COBIT IT Assessment/Audit Tool

Introduction

The goal of information technology certification programs is to provide alignment for IT infrastructure with the business goals of the corporation.

- Optimal configurations : for performance and return-on-investment
- Scalability and flexibility : to meet rapidly growing or changing business conditions
- Performance engineering : for mission-critical systems
- Predictable cost control : for projects, operations and planning
- Risk analysis : for information technology and capital investment

This Assessment/Audit tool contains, within 4 areas of IT control, a total of 34 high-level control objectives:

- Planning and Organization - IT Controls
- Acquisition and Implementation - IT Controls
- Delivery and Support - IT Controls
- Monitoring - IT Controls

This Assessment/Audit tool's detailed-control statements are graded on a scale of 0-5:

- 0 Non-existent - no recognizable process
- 1 Initial - no standardized process
- 2 Repeatable - standardized process in place
- 3 Defined - policy/procedures are standardized and documented
- 4 Managed - compliance monitors are in place and utilized
- 5 Optimized -processes are refined

Overview

- **Three levels of IT management** ... domains, processes, activities + tasks
- **Utilize IT resources** ... people, application systems, technology, facilities, data
- **To produce information measured by criteria** ... quality ... fiduciary ... security

Domains are management groupings within an organization's structure (e.g. division)

Processes are joined activities/tasks with natural control breaks (e.g. department)

Activities are joined tasks with a defined life-cycle designed to produce a measurable result(s)

Tasks are discrete actions required to achieve measurable business goals
note: activities and tasks require a different type of control than domains and processes

This assessment tool is based on the COBIT Framework for certification of performance in the management, security and control of information technology - as developed by the IT Governance Institute (www.ITgovernance.org and www.isaca.org). COBIT is an acronym which stands for "**C**ontrol **Ob**jectives for **I**nformation and related **T**echnology"

Planning and Organization ... IT Controls

High-Level Control Statements

PO1	Define a Strategic IT Plan
PO2	Define the Information Architecture
PO3	Define the Technological Direction
PO4	Define the IT Organization and Relationships
PO5	Manage the IT Investment
PO6	Communicate Management Aims and Direction
PO7	Manage Human Resources
PO8	Ensure Compliance with External Requirements
PO9	Assess Risks
PO10	Manage Projects
PO11	Manage Quality

Detailed Control Objectives

1.0 Define a Strategic IT Plan

- 1.1 IT as Part of the Organization's Long- and Short-Range Plan
- 1.2 IT Long-Range Plan
- 1.3 IT Long-Range Planning - Approach and Structure
- 1.4 IT Long-Range Plan Changes
- 1.5 Short-Range Planning for the IT Function
- 1.6 Communication of IT Plans
- 1.7 Monitoring and Evaluation of IT Plans
- 1.8 Assessment of Existing Systems

2.0 Define the Information Architecture

- 2.1 Information Architecture Model
- 2.2 Corporate Data Dictionary and Data Syntax Rules
- 2.3 Data Classification Scheme
- 2.4 Security Levels

3.0 Determine Technological Direction

- 3.1 Technological Infrastructure Planning
- 3.2 Monitoring Future Trends and Regulations
- 3.3 Technological Infrastructure Contingency
- 3.4 Hardware and Software Acquisition Plans
- 3.5 Technology Standards

4.0 Define the IT Organization and Relationships

- 4.1 IT Planning or Steering Committee
- 4.2 Organizational Placement of the IT Function
- 4.3 Review of Organizational Achievements
- 4.4 Roles and Responsibilities
- 4.5 Responsibility for Quality Assurance
- 4.6 Responsibility for Logical and Physical Security
- 4.7 Ownership and Custodianship
- 4.8 Data and System Ownership
- 4.9 Supervision
- 4.10 Segregation of Duties
- 4.11 IT Staffing
- 4.12 Job or Position Descriptions for IT Staff
- 4.13 Key IT Personnel
- 4.14 Contracted Staff Policies and Procedures
- 4.15 Relationships

5.0 Manage the IT Investment

- 5.1 Annual IT Operating Budget
- 5.2 Cost and Benefit Monitoring
- 5.3 Cost and Benefit Justification

6.0 Communicate Management Aims and Direction

- 6.1 Positive Information Control Environment
- 6.2 Management's Responsibility for Policies
- 6.3 Communication of Organizational Policies
- 6.4 Policy Implementation Resources
- 6.5 Maintenance of Policies
- 6.6 Compliance with Policies, Procedures and Standards
- 6.7 Quality Commitment
- 6.8 Security and Internal Control Framework Policy
- 6.9 Intellectual Property Rights
- 6.10 Issue-Specific Policies
- 6.11 Communication of IT Security Awareness

7.0 Manage Human Resources

- 7.1 Personnel Recruitment and Promotion
- 7.2 Personnel Qualifications
- 7.3 Roles and Responsibilities
- 7.4 Personnel Training
- 7.5 Cross-Training or Staff Back-up
- 7.6 Personnel Clearance Procedures
- 7.7 Employee Job Performance Evaluation
- 7.8 Job Change and Termination

8.0 Ensure Compliance with External Requirements

- 8.1 External Requirements Review
- 8.2 Practices and Procedures for Complying with External Requirements
- 8.3 Safety and Ergonomic Compliance
- 8.4 Privacy, Intellectual Property and Data Flow
- 8.5 Electronic Commerce
- 8.6 Compliance with Insurance Contracts

9.0 Assess Risks

- 9.1 Business Risk Assessment
- 9.2 Risk Assessment Approach

- 9.3 Risk Identification
- 9.4 Risk Measurement
- 9.5 Risk Action Plan
- 9.6 Risk Acceptance
- 9.7 Safeguard Selection
- 9.8 Risk Assessment Commitment

10.0 Manage Projects

- 10.1 Project Management Framework
- 10.2 User Department Participation in Project Initiation
- 10.3 Project Team Membership and Responsibilities
- 10.4 Project Definition
- 10.5 Project Approval
- 10.6 Project Phase Approval
- 10.7 Project Master Plan
- 10.8 System Quality Assurance Plan
- 10.9 Planning of Assurance Methods
- 10.10 Formal Project Risk Management
- 10.11 Test Plan
- 10.12 Training Plan
- 10.13 Post-Implementation Review Plan

11.0 Manage Quality

- 11.1 General Quality Plan
- 11.2 Quality Assurance Approach
- 11.3 Quality Assurance Planning
- 11.4 Quality Assurance Review of Adherence to IT Standards and Procedures
- 11.5 System Development Life Cycle Methodology
- 11.6 System Development Life Cycle Methodology for Major Changes to Existing Technology
- 11.7 Updating of the System Development Life Cycle Methodology
- 11.8 Coordination and Communication
- 11.9 Acquisition and Maintenance Framework for the Technology Infrastructure
- 11.10 Third-Party Implementor Relationships
- 11.11 Program Documentation Standards
- 11.12 Program Testing Standards
- 11.13 System Testing Standards
- 11.14 Parallel/Pilot Testing
- 11.15 System Testing Documentation
- 11.16 Quality Assurance Evaluation of Adherence to Development Standards
- 11.17 Quality Assurance Review of the Achievement of IT Objectives
- 11.18 Quality Metrics

- 11.19 Reports of Quality Assurance Reviews

Acquisition and Implementation - IT Controls

High-Level Control Statements

A11	Identify Automated Solutions
A12	Acquire and Maintain Application Software
A13	Acquire and Maintain Technology Infrastructure
A14	Develop and Maintain Procedures
A15	Install and Accredit Systems
A16	Manage Changes

Detailed Control Objectives

1.0 Identify Automated Solutions

- 1.1 Definition of Information Requirements
- 1.2 Formulation of Alternative Courses of Action
- 1.3 Formulation of Acquisition Strategy
- 1.4 Third-Party Service Requirements
- 1.5 Technological Feasibility Study
- 1.6 Economic Feasibility Study
- 1.7 Information Architecture
- 1.8 Risk Analysis Report
- 1.9 Cost-Effective Security Controls
- 1.10 Audit Trails Design
- 1.11 Ergonomics
- 1.12 Selection of System Software
- 1.13 Procurement Control
- 1.14 Software Product Acquisition
- 1.15 Third-Party Software Maintenance
- 1.16 Contract Application Programming
- 1.17 Acceptance of Facilities
- 1.18 Acceptance of Technology

2.0 Acquire and Maintain Application Software

- 2.1 Design Methods
- 2.2 Major Changes to Existing Systems
- 2.3 Design Approval
- 2.4 File Requirements Definition and Documentation
- 2.5 Program Specifications
- 2.6 Source Data Collection Design
- 2.7 Input Requirements Definition and Documentation
- 2.8 Definition of Interfaces
- 2.9 User-Machine Interface
- 2.10 Processing Requirements Definition and Documentation
- 2.11 Output Requirements Definition and Documentation
- 2.12 Controllability
- 2.13 Availability as a Key Design Factor
- 2.14 IT Integrity Provisions in Application Program Software
- 2.15 Application Software Testing
- 2.16 User Reference and Support Materials
- 2.17 Reassessment of System Design

3.0 Acquire and Maintain Technology Infrastructure

- 3.1 Assessment of New Hardware and Software
- 3.2 Preventive Maintenance for Hardware
- 3.3 System Software Security
- 3.4 System Software Installation
- 3.5 System Software Maintenance
- 3.6 System Software Change Controls
- 3.7 Use and Monitoring of System Utilities

4.0 Develop and Maintain Procedures

- 4.1 Operational Requirements and Service Levels
- 4.2 User Procedures Manual
- 4.3 Operations Manual
- 4.4 Training Materials

5.0 Install and Accredit Systems

- 5.1 Training
- 5.2 Application Software Performance Sizing
- 5.3 Implementation Plan
- 5.4 System Conversion
- 5.5 Data Conversion
- 5.6 Testing Strategies and Plans
- 5.7 Testing of Changes
- 5.8 Parallel/Pilot Testing Criteria and Performance
- 5.9 Final Acceptance Test

- 5.10 Security Testing and Accreditation
- 5.11 Operational Test
- 5.12 Promotion to Production
- 5.13 Evaluation of Meeting User Requirements
- 5.14 Management's Post-Implementation review

6.0 Manage Changes

- 6.1 Change Request Initiation and Control
- 6.2 Impact Assessment
- 6.3 Control of Changes
- 6.4 Emergency Changes
- 6.5 Documentation and Procedures
- 6.6 Authorized Maintenance
- 6.7 Software Release Policy
- 6.8 Distribution of Software

Delivery and Support ... IT Controls

High-Level Control Statements

DS1	Define and Manage Service Levels
DS2	Manage Third-Party Services
DS3	Manage Performance and Capacity
DS4	Ensure Continuous Service
DS5	Ensure Systems Security
DS6	Identify and Allocate Costs
DS7	Educate and Train Users
DS8	Assist and Advise Customers
DS9	Manage the Configuration
DS10	Manage Problems and Incidents
DS11	Manage Data
DS12	Manage Facilities

DS13 Manage Operations

Detailed Control Objectives

1.0 Define and Manage Service Levels

- 1.1 Service Level Agreement Framework
- 1.2 Aspects of Service Level Agreements
- 1.3 Performance Procedures
- 1.4 Monitoring and Reporting
- 1.5 Review of Service Level Agreements and Contracts
- 1.6 Chargeable Items
- 1.7 Service Improvement Program

2.0 Manage Third-Party Services

- 2.1 Supplier Interfaces
- 2.2 Owner Relationships
- 2.3 Third-Party Contracts
- 2.4 Third-Party Qualifications
- 2.5 Outsourcing Contracts
- 2.6 Continuity of Services
- 2.7 Security Relationships
- 2.8 Monitoring

3.0 Manage Performance and Capacity

- 3.1 Availability and Performance Requirements
- 3.2 Availability Plan
- 3.3 Monitoring and Reporting
- 3.4 Modeling Tools
- 3.5 Proactive Performance Management
- 3.6 Workload Forecasting
- 3.7 Capacity Management of Resources
- 3.8 Resources Availability
- 3.9 Resources Schedule

4.0 Ensure Continuous Service

- 4.1 IT Continuity Framework
- 4.2 IT Continuity Plan Strategy and Philosophy
- 4.3 IT Continuity Plan Contents
- 4.4 Minimizing IT Continuity Requirements
- 4.5 Maintaining the Continuity Plan

- 4.6 Testing the IT Continuity Plan
- 4.7 IT Continuity Plan Training
- 4.8 IT Continuity Plan Distribution
- 4.9 User Department Alternative Processing Back-up Procedures
- 4.10 Critical IT Resources
- 4.11 Back-up Site and Hardware
- 4.12 Off-site Back-up Storage
- 4.13 Wrap-up Procedures

5.0 Ensure Systems Security

- 5.1 Manage Security Measures
- 5.2 Identification, Authentication and Access
- 5.3 Security of Online Access to Data
- 5.4 User Account Management
- 5.5 Management Review of User Accounts
- 5.6 User Control of User Accounts
- 5.7 Security Surveillance
- 5.8 Data Classification
- 5.9 Central Identification and Access Rights Management
- 5.10 Violation and Security Activity Reports
- 5.11 Incident Handling
- 5.12 Re-accreditation
- 5.13 Counter-party Trust
- 5.14 Transaction Authorization
- 5.15 Non-Repudiation
- 5.16 Trusted Path
- 5.17 Protection of Security Functions
- 5.18 Cryptographic Key Management
- 5.19 Malicious Software Prevention, Detection and Correction
- 5.20 Firewall Architectures and Connections with Public Networks
- 5.21 Protection of Electronic Value

6.0 Identify and Allocate Costs

- 6.1 Chargeable Items
- 6.2 Costing Procedures
- 6.3 User Billing and Charge-back Procedures

7.0 Educate and Train Users

- 7.1 Identification of Training Needs
- 7.2 Training Organization
- 7.3 Security Principles and Awareness Training

8.0 Assist and Advise Customers

- 8.1 Help Desk
- 8.2 Registration of Customer Queries
- 8.3 Customer Query Escalation
- 8.4 Monitoring of Clearance
- 8.5 Trend Analysis and Reporting

9.0 Manage the Configuration

- 9.1 Configuration Recording
- 9.2 Configuration Baseline
- 9.3 Status Accounting
- 9.4 Configuration Control
- 9.5 Unauthorized Software
- 9.6 Software Storage
- 9.7 Configuration Management Procedures
- 9.8 Software Accountability

10. Manage Problems and Incidents

- 10.1 Problem Management System
- 10.2 Problem Escalation
- 10.3 Problem Tracking and Audit Trail
- 10.4 Emergency and Temporary Access Authorizations
- 10.5 Emergency Processing Priorities

11.0 Manage Data

- 11.1 Data Preparation Procedures
- 11.2 Source Document Authorization Procedures
- 11.3 Source Document Data Collection
- 11.4 Source Document Error Handling
- 11.5 Source Document Retention
- 11.6 Data Input Authorization Procedures
- 11.7 Accuracy, Completeness and Authorization Checks
- 11.8 Data Input Error Handling
- 11.9 Data Processing Integrity
- 11.10 Data Processing Validation and Editing
- 11.11 Data Processing Error Handling
- 11.12 Output Handling and Retention
- 11.13 Output Distribution
- 11.14 Output Balancing and Reconciliation
- 11.15 Output Review and Error Handling
- 11.16 Security Provision for Output Reports

- 11.17 Protection of Sensitive Information During Transmission and Transport
- 11.18 Protection of Disposed Sensitive Information
- 11.19 Storage Management
- 11.20 Retention Periods and Storage Terms
- 11.21 Media Library Management System
- 11.22 Media Library Management Responsibilities
- 11.23 Back-up and Restoration
- 11.24 Back-up Jobs
- 11.25 Back-up Storage
- 11.26 Archiving
- 11.27 Protection of Sensitive Messages
- 11.28 Authentication and Integrity
- 11.29 Electronic Transaction Integrity
- 11.30 Continued Integrity of Stored Data

12.0 Manage Facilities

- 12.1 Physical Security
- 12.2 Low Profile of the IT Site
- 12.3 Visitor Escort
- 12.4 Personnel Health and Safety
- 12.5 Protection Against Environmental Factors
- 12.6 Uninterruptible Power Supply

13.0 Manage Operations

- 13.1 Processing Operations Procedures and Instructions Manual
- 13.2 Start-up Process and Other Operations Documentation
- 13.3 Job Scheduling
- 13.4 Departures from Standard Job Schedules
- 13.5 Processing Continuity
- 13.6 Operations Logs
- 13.7 Safeguard Special Forms and Output Devices
- 13.8 Remote Operations

Monitoring - IT Controls

High-Level Control Statements

M1 Monitor the Processes

M2 Assess Internal Control Adequacy

M3 Obtain Independent Assurance

M4 Provide for Independent Audit

Detailed Control Objectives

1.0 Monitor the Processes

- 1.1 Collecting Monitoring Data
- 1.2 Assessing Performance
- 1.3 Assessing Customer Satisfaction
- 1.4 Management Reporting

2.0 Assess Internal Control Adequacy

- 2.1 Internal Control Monitoring
- 2.2 Timely Operation of Internal Controls
- 2.3 Internal Control Level Reporting
- 2.4 Operational Security and Internal Control Assurance

3.0 Obtain Independent Assurance

- 3.1 Independent Security and Internal Control Certification/Accreditation of IT Services
- 3.2 Independent Security and Internal Control Certification/Accreditation of Third-Party Service Providers
- 3.3 Independent Effectiveness Evaluation of IT Services
- 3.4 Independent Effectiveness Evaluation of Third-Party Service Providers
- 3.5 Independent Assurance of Compliance with Laws, Regulatory Requirements & Contractual Commitments
- 3.6 Independent Assurance of Compliance with Laws, Regulatory Requirements and Contractual Commitments of Third-Party Providers
- 3.7 Competence of Independent Assurance Function
- 3.8 Proactive Audit Involvement

4.0 Provide for Independent Audit

- 4.1 Audit Charter
- 4.2 Independence
- 4.3 Professional Ethics and Standards
- 4.4 Competence
- 4.5 Planning

- 4.6 Performance of Audit Work
- 4.7 Reporting
- 4.8 Follow-up Activities

Portfolio Management for Information Services

There are pretty good presidents that successful organizations manage information technology and telecommunication services/projects as investments rather than expenses. The management model often seen, in successful corporations, is the **portfolio investment** approach which utilizes explicit decision criteria with cost/benefit/risk assessment to select and control the management of information and telecommunications technology.

Portfolio investment tools systematically reduce inherent risk while maximizing the benefits of operations and implementation-projects. The core activity is to correctly assess the costs of continued funding for existing operations versus the costs of developing new performance capabilities.

The dynamic tension between continuation of an existing operations' model and the uncertainty of a new model usually surfaces during annual budget decision making. In theory, the disciplined processes of portfolio investment should help to make apparent to all the explicit links between business needs and the technology under consideration.

Of course, this is only clear in the somewhat perfect world of theory. Portfolio management is complex and probably significantly different in each successful enterprise ... in contrast ... what will stand out during an audit are the **attributes of non-centralized management processes** incorrectly used to select, control and evaluate information technology and telecommunications. Problems that will stand out during an audit include:

- significant unmanaged risk
- unexamined low-value or redundant projects which consume scarce resources
- mismatches between systems maintenance and strategic priorities for improving mission performance
- design flaws that can unexpectedly increase complexity
- outsourcing decisions that put an organization at risk

What can go wrong with outsourcing?

- outsourcing may replace a better intra-corporate alternative
- actual cost may exceed the quoted price
- enterprise control over operations schedules and projects may be lost

- technical competence of the outsource vendor may turn out to be marginal
- outsource vendor may go out of business during the contract period

What are a few key findings which signal a problem during an IT audit?

- illegal software is found
- examples of de-motivated employees are noted
- excessive costs and/or delays for IT operations and projects
- examples of missed business opportunities
- examples of competitive disadvantage
- reduced credibility of Information Services ... management ... operations' level users ... etc.

What are attributes of good IT management?

- risk management
- business process re-engineering and improvement
- TQM
- service level management
- business change management
- marketing IT services internally
- IT portfolio management

What are detection level controls attributed to good IT management?

- benchmarking
- cost/benefit analysis
- activity cost accounting
- user billing and charge-back procedures
- software license audits
- incident response capability

Assessment/Audit Criteria for Business Application Systems

The assessment/audit of business application systems should consider the required actions and deliverables at each phase in the life-cycle of these resources:

- systems planning
- systems analysis
- systems design

- systems construction
- systems implementation
- systems operation and maintenance

Systems Planning - Actions	Systems Planning - Deliverables
• define scope and objective of system • develop initial estimates of cost, benefits, savings, project time line • perform fact-finding analysis • identify and evaluate alternative systems • prepare cost-benefit analysis for each alternative • identify best solution	• statement of objectives • feasibility study • recommendations to proceed or not • preliminary implementation plan

Systems Analysis - Actions	Systems Analysis - Deliverables
• perform data analysis • perform process analysis • understand how current system works • understand system user requirements • define system and business requirements • define security, control and audit requirements	• data and process analysis documents • system and business requirements document

Systems Design - Actions	Systems Design - Deliverables
• identify the business and systems functions to be performed by the proposed system as required by the users • define the input, processing and output activities that support each business function • define the data requirements and identify their sources • group the activities into subsystems or modules, each accomplishing one or more business functions and develop	• description of the new systems' requirements explained as input, processing and output activities • description of data elements and data dictionaries • logical and physical databases • edit and validation criteria • design and program specifications • data file characteristics

specifications • finalize the control, security and audit requirements	

Systems Construction - Actions	Systems Construction - Deliverables
• develop computer program instructions • conduct program walk-thorough • conduct program/unit testing • conduct system testing of all programs/interfaces	• computer program source code • program/unit test plan and test results • systems test plan and test results • file conversion plan and approach

Systems Implementation - Actions	Systems Implementation - Deliverables
• develop operations and user manuals • develop training materials • conduct user and IS training • conduct acceptance testing of the entire system by users to test all functions, errors, reports, screens and interfaces • identify and document the discrepancies between expected test results and actual test results • convert and validate data files and programs • conduct pilot conversions or parallel operations	• operations manual • user manual • training manual • user acceptance test plan and test results • data file conversion results

Systems Ops & Maintenance - Actions	Systems Ops and Maintenance - Deliverables
• move programs to production • provide production support • conduct post implementation review	• production problem report • list of system enhancements • list of system changes

Model for the Audit of Information Technology and Telecommunications Resources

Actions performed by an Auditor include:

- attend project meetings to understand progress and problems
- evaluate system and manual controls present in the system
- review conformance with project milestones, deliverables and standards
- compare projected financial time frames and costs with actual numbers
- review software contracts to ensure legal and financial risks are minimized
- review financial stability of vendors/outsourcers to ensure they can fulfill obligations
- confirm that system development is in compliance with plan and program changes are approved
- design various tools and techniques to facilitate audit routines and self-assessment programs

Two components of an audit are:

- audit the methodology ..... philosophy, guidelines, policies, responsibilities, time line, deliverables
- audit the project/process ... is compliance with the methodology, identify reasons for deviations

Areas of **primary focus** for an audit:

- original feasibility study
- analysis of user/system requirements
- system design specifications

Areas of **least importance** for an audit's focus are:

- program development
- program unit/model testing

Project/Process Controls which an audit should look for:

- system development methodology
- system maintenance methodology
- system acquisition methodology
- system-based control guidelines
- system security guidelines

- project management controls
- software documentation guidelines
- software verification and validation guidelines
- quality assurance guidelines
- software testing guidelines
- software productivity tools
- software configuration management guidelines

The primary controls which stabilize a system are: **preventive, detective and corrective.**

Examples of **preventive controls** to look for during an audit:

- system development/acquisition/maintenance methodology is present and provides control for process/project costs and schedules, provides standard procedures, seeks to improve software quality and ensures effective among process/project (inter and intra systems) team members
- experienced and qualified staff
- documentation ... user manuals, policies, reports, error messages
- detailed system specifications ... narratives, layouts, flowcharts, etc.
- management involvement
- active user participation for requirements, specifications, test plans, control procedures, etc.
- user sign-offs
- project management controls ... milestone events, feasibility study, business/technical specifications
- controls and audit trails

Examples of **detective controls** to look for during an audit:

- software access security controls
- software testing standards
- post-implementation reviews
- desk checks and code reviews
- structured walkthroughs and/or peer reviews

Examples of **corrective controls** to look for during an audit:

- system-generated reports
- exception reports
- error reports
- statistics
- system documentation manuals
- user manuals with error correction procedures

Project Management Theory

Information Technology Audit & Certification Criteria

Information Technology Governance is the structure of relationships and processes within an enterprise which add value to a corporation's goals while balancing risk with return-on-investment.

The management of existing information technology and the implementation of new information technology resources require increased consideration for both **security** and **control** in the management of these resources.

Change Management

Change Management involves changing steady-state.

Reasons to initiate change management for information technology are often related to new activity patterns in supply delivery, the competitive environment in a services/product market or the strategy to expand into new markets. Change management often addresses a need to increase:

- effectiveness
- efficiency
- confidentiality
- integrity
- availability
- compliance
- reliability

System structure influences behaviour ... that is ... placed in the same system, people - however different - will tend to produce similar results ... a system tends to cause its own behaviour.

System Structure
generates

Patterns of Behaviour
resulting in
Events

Some Principles associated with Steady-State and Change

- today's problems come from yesterday's solutions
- the harder you push on a steady-state system ... the harder it will push back
- behaviour generally grows better before it grows worse
- the easiest way out of a problem most often will lead back into the problem
- often an objective will reveal that the cure can be worse than the disease ... that is ... the long-term consequences of applying non-systemic solutions to a systemic problem will, over time, increase the need for more and more non-systemic solutions
- faster is slower
- cause and effect are not closely related in time and space
- the areas of highest **leverage** are often the least obvious ... when found ... leverage permits small changes to produce big results ... to find the area of highest leverage look for underlying structures rather

than events ... real leverage lies in understanding the "dynamic complexity" not the "detail complexity" ... proper use of leverage relies on: balancing feedback, reinforcing feedback and delay

- you can have your cake and eat it too - but not at the same time

- dividing an elephant in half does not produce two small elephants ... that is ... there is an integrity in the "whole" of a system that will not be duplicated if it is split

- there is no "blame" ... there is no "outside" entity to blame ... you and the cause of your problems are part of a single system ... "outside" entities require relationship

Learning is a key element in the management of change

Organizations, work groups and teams learn only through individuals who learn.

Individual learning does not guarantee organizational learning but without it no organizational learning will occur.

Learning is necessary for personal mastery (seeing current reality clearly with an orientation for what is important) which fosters self-respect and self-actualization. From this platform, an individual will be able to challenge the corporation's goals-for-growth and technological development.
Essentially, it is not the organization that pulls the individual toward learning ... it's the individual that pulls the organization toward learning in an environment of change ... this is important because organizational process that represent the current reality or steady-state are resistant to change.

Operations Level Processing and Change

There are two Process Types - both represent complex systems

1. **Primitive Cultures**
   - flexibility and effectiveness are most essential to survival
   - use Reinforcing Feedback for Change Management

2. **Diversified Cultures**
   - stability of status quo is most essential to survival

- use Balancing Feedback for Change Management

Primary Controls for Operations Level Processing

1. Reinforcing Feedback

- growth, change
- is always a growth engine
- Reinforcing processes rarely occur in isolation ... eventually, limits are encountered which can slow growth, stop it, divert it or even reverse it ... these limits represent balancing feedback in active or dynamic interaction with a growth process

2. Balancing Feedback

- stabilizer of status quo
- operates whenever there is goal oriented behaviour
- seeks to stabilize a process or system ... it provides self-correction that attempts to maintain goal(s)
- what makes balancing processes so difficult to deal with in an environment of change is that the goals are often implicit and no one recognizes that the balancing process(es) exist at all. The result is that true problematic behaviour goes undetected during problem solving because it looks like nothing is happening. Examples include human resources resistance to change ... when dealing with the realities of balancing feedback - it is better to focus on implicit norms and power relationships rather than the explicit issues in which they are embedded.

3. Delay

- the time/space, that is interruption, between an action and its consequences
- delay should be used from a perspective of objectivity and control

Reinforcing Feedback, Balancing Feedback and Delay are the building blocks of System Archetypes.

Domains of Order

1. Steady-state variables never change
2. Periodic Behaviour variables have repeating loops
3. Chaotic or Lorenz variables change randomly and are unpredictable but sequence of change is predicable

Chaotic or Lorenz Order

- often termed: random systems -or- non-repeating periodic states
- variables change randomly and are unpredictable but the sequence of change is predictable
- bifurcation - the point where the sequence or frequency of change occurs
- bifurcation constant for all true random states is a bifurcation rate of 4.669
- chaotic systems demonstrate a sensitive dependence on initial conditions
- small changes in initial conditions can drastically change the long term behaviour of a systems
- terms: bifurcation, fractile, Koch Curve, Mandelbret Set
- Koch Curve (fractile) is a line of infinite length that surrounds a finite area ... the inner area of this structure remains less than a circle drawn around the original structure (usually a triangle)

Stabilizers of "Self"

- family of origin
- significant other
- tiered friendships
- meaningful work
- spiritual beliefs

Hierarchy of Needs

- self-actualization
- aesthetic needs
- need to understand
- estimate needs
- belonging and love needs
- safety needs
- physiology needs (food, housing)
- this is a sequential hierarchy ... in situations of persuasion or conflict it is necessary to pick the appropriate level for action and then move the subject in sequential steps, satisfying needs, toward the apex of the hierarchy

Types of Barriers encountered in Change Management

- From a logistics perspective, barriers may be: focused, centralized or distributed
- From a practical perspective, barriers may be: physical, monetary, vision, conceptual

Change Management for Information Technology requires:

Criteria Development

- quality
- fiduciary
- security

Resource Development

- people
- systems / applications
- technology
- facilities
- data

Process Development

- critical success factors
- key goal indicators
- key performance indicators
- maturity model for assessment/audit

Project Management Theory

Project Estimating

Management approval for a project and subsequent project controls all are based on project estimating. If the original estimations are off, the budget, time line, and outcomes are off.

Project estimation has four inter-related components:

- Time .... complexity, training needs, communication, time line, costs
- Scope .... size, difficulty, reliability, methodology, tools
- Quality .... requirement attributes: e.g. well-defined, well-understood, financials
- Resources ... human resources (#, experience, credentials, performance, resources

Both quantitative and qualitative methods are used to estimate project variables ... no single method is complete ... look for a combination of methods. All methods are subject to unreliable variables and bias. The best way to address this issue in an audit is to perform select detailed-decomposition of project tasks (e.g. work breakdown structure) ... this action tends to remove ambiguities.

Qualitative methods include:

- rules-of-thumb methods
- heuristics methods
- trial-and-error methods

note: heuristic methods include: authority, reactance, consensus, commitment, liking, similarity ... these are qualitative issues which marketing pitches utilize to sell products ... they are important to project acceptance.

Quantitative methods include:

- comparisons
- simple ratios
- regression analysis (parametric modelling)

note: regression analysis expresses a quantifiable relationship between dependent and independent variables

Project Planning and Scheduling

All project plans should specify key points within the project, deliverables at these points and threats represented by delay and/or failure of deliverables. The project plan should be a document which describes how the project work will be conducted, what is to be accomplished, when specific segments are performed, who is responsible and how much it will cost. The plan should define responsibilities and accountability for all people involved in the project.

Components of project planning and scheduling are:

- milestones .... major checkpoints for progress review on a defined time line
- deliverables .... tangible work products for review
- baseline .... starting point at beginning ... ending point for future changes
- assumptions .... provides vision/understanding of the projects goals and direction
- risks, dependencies ... defines uncertainty
- contingencies .... for known, high probability risk
- diagrams, graphics .... showing tasks and their sequence

Project Monitoring and Control

Projects can often extend over lengthy periods of time and have varied levels of participation by many people. Monitoring and control of such processes is essential and often includes such components as:

- policies, procedures and forms for change management
- logs
- checklists
- status reports

In general, formal control techniques which include documentation are preferred over informal communications like phone conversations and face-to-face encounters ... the intent is to provide a culture in which evidence is available for what has been said and participants understand the where and how to resolve questions and/or disputes.

Change Management

Lots of things change during a project: requirements change, design approaches change, business needs change, team members turnover. The initial baseline for the project includes estimates for cost and time; if an event impacts the baseline estimates then the project plan should incorporate the event into the plan and recalculate the baseline assumptions. Techniques utilized for change management include:

- logging all changes with a sequential control number
- assigning the change task to appropriate staff/team members
- checking progress periodically
- reporting the status of change (e.g. completed, deferred, pending, etc.)

Communications is an integral component of change management ... it should include attributes of timeliness and complete communications to all parties impacted by the change ... selection of communication media should be tied to the importance of the message being communicated.

Project Progress Reporting

Management needs to assess the progress of projects. This requires meaningful and measurable checkpoints reported in a format that permits determination of whether or not the project is on schedule and within its budget. Reporting mechanisms should fit the size of the project. The project team should work in a culture which supports truthful and meaningful reporting ... watch out for what is termed the 90% complete syndrome ... inexperienced people will often report their tasks as 90% complete each week because they micro-focus on immediate time frame deliverables ... a better approach is to build status monitoring which defines the current status in terms of the time and effort estimate for completion of all tasks. It is fair to accomplish status monitoring by status reports, meetings or a combination.

Course Corrections

No project stands still and very few projects proceed according to plan. Thus, mid-point reviews and other checkpoints are needed to bring a project back on schedule. During these course corrections, timely and proactive action by project management will prevent delays or worse:

Assess the situation

Identify problems

Isolate causes for problems

Allocate resources to fix problems

Project Communication

Project communication can take place in several ways:

Meetings

Reports

Etc.

Communication provides reporting guidelines to project members and is a core mechanism for maintaining quality and consistency in reporting. Meeting guidelines for a project should include:

Each meeting should have a defined purpose

Agendas should be available to members prior to the meeting

The "right" people required for action should constitute the meeting's membership

Meeting groups should be relatively small (less than 10 is optimum)

Meeting length should be no longer than 1 hour

Each meeting should have a definite conclusion

Set clear goals for the next meeting

Project Quality Assurance

Quality assurance is the forecasting and prevention of quality related problems. A project quality requirements document will incorporate the customer's quality requirements. Every effort is taken to meet the customer quality requirements.

Quality assurance is concerned with the forecasting and prevention of quality problems. It is pro-active. Examples of quality assurance tools include:

- Policies and procedures
- Supervision
- Periodic checking and rechecking
- Reviews
- Checklists
- Walk-through

Quality control is concerned with detecting and reporting quality-related problems. It is reactive. Examples of quality control tools include:

- Inspection tools
- Measurement gauges
- Acceptance tests
- Statistical process-control charts with upper and lower control limits

Software quality assurance is a planned systematic pattern for all actions necessary to provide adequate confidence that the product, or process by which the product is developed, conforms to established requirements.

The primary goal of software quality assurance is to enhance the quality of software. To this end, a new concept, Clean Room engineering, is evolving where by software is developed to assure quality. With Clean Room engineering, programmers do not compile their code. Instead they spend more time on design using a "box structure" method, and on analyzing their own work. When the programmers are confident of their work, it is submitted to another group, who then compiles and tests the code. Clean Room processing shows a lower error rate in the finished product and an increase in productivity across the life cycle of the application.

Examining the Need for Software Quality Assurance

Some symptoms leading to establishment of a software quality assurance function include:

- when the software is exhibiting many errors
- when users stop using the software
- when customers express quality-related problems
- when sales and profits are declining

Under these conditions, a software quality assurance function should be established to assure quality. For example, periodic checks of test results (e.g., number of errors found) may reveal an error rate outside the control limits. A reviewer may discover that this is due to the fact that inspections had been eliminated when the project fell behind schedule.

Software Quality Assurance Planning and Standards

Software Quality Assurance Plans

A software quality assurance plan should identify what quality measurements are to be taken, when they are to be examined, by whom, their purpose, and who has the authority to make changes in any of the processes. The purpose could be (1) to determine if there is a problem, (2) to evaluate the effect of change in the development process, and (3) to examine the effectiveness of testing, inspection or other analysis techniques. Major elements of a software quality assurance plan include: organization for quality responsibilities; identification of applicable software products; standards, practices, and conventions; metrics, reviews, and audits; documentation, error reporting and corrective action; records management; security access control; training; tools, techniques and methodologies; and risk management.

Software Quality Assurance Standards

Measurement of the software development process and product is valuable to the developing organization. Metrics may be used to measure quality. To this end, standards improve the quality of software. If a standard requires an activity, the standard should contain requirements for the documentation, review, and monitoring of this activity. Inclusion of statements such as the following would be useful: (1)(each life cycle activity shall be documented; (2) reports shall be prepared on each activity's implementation; (3) a quality assurance review shall verify that the life cycle activity plan is acceptable and was accomplished (or if not, why not); (4) tracking mechanisms shall provide information to project management and reviewers.

Software Quality Factors

Software quality factors or attributes are the requirements that software must meet, such as usability, efficiency, reliability, maintainability, and portability.

Usability refers to the ease with which software can be used. It determines the user-friendliness of software.

Efficiency and economy are related. An economical and efficient system uses the minimum number of information resources to achieve the output level required by the system's users. System economy and efficiency must always be considered within the context of system effectiveness.

Reliability is the probability that a given software system operates for some time period, without system failure due to a software fault, on the computer for which it was designed, given that it is used within design limits.

Maintainability is the characteristic of code facilitating an incorporation of changes.

Portability refers to the degree of movement of the software between the same vendor's computer models or across different vendors. It is the ability to operate on multiple computer types from different vendors.

Software Quality Assurance

The thrust of software assurance (SQA) is product assurance. Which is important – process or product? Quality of process is related to the quality of the product. New software quality assurance focuses on evaluating the processes by which products are developed or manufactured.

The major objectives of the SQA process are to ensure that the software development and software assurance processes comply with software assurance plans and standards, and to recommend process improvements. The process uses the system requirements, and information about the purpose and criticality of the software, to evaluate the outputs of the software development and software assurance processes. It begins before the software requirements process and ends when its objectives have been met. A software quality assurance plan and review/audit reports are produced during the SQA process.

Software Quality Assurance Activities

Major activities in software quality assurance include:

- project management
- software verification and validation (SV&V)
- software configuration management (SCM)
- software quality assurance (SQA).

Some activities in project management include

- Were reviews of project status and assurance task reports conducted?
- Based on metrics data, were any processes found to be outside statistical control limits?
- If so, what actions were taken?

Some activities in SV&V include

- Were the test strategies consistent with the requirements and were they adequate to evaluate the design?
- Were the test strategies sufficient to determine whether the software is safe and effective?
- How were the safety features of the software tested and analyzed?
- Do test results show that safety-related functions were tested adequately and that they perform reliably?

Some activities in SCM include

- At what point was the first SCM baseline established?
- What evidence is there that the software delivered is the software that has been verified and validated?

Some activities in SQA include

- What actions were taken in response to the metrics results?
- Do SQA reports indicate that standards were properly followed?
- What error analysis techniques were used? How did the developer or vendor respond to the findings?
- Were software design and code-inspections performed?
- Were technical reviews held, and problems identified in the reviews satisfactorily resolved in the product?

Software Error Analysis

Software error analysis includes the techniques used to locate, analyze, and estimate errors and data related to errors. It includes the use of error detection techniques, analysis of single errors, data collection, metrics, statistical process-control techniques, error prediction models, and reliability models.

Error detection techniques are techniques of software development, software quality assurance, software verification, validation and testing used to locate anomalies in software products. An anomaly is any condition which departs from the expected. Errors, defects, faults, and failures are considered anomalies. Once an anomaly is detected, analysis is performed to determine if the anomaly is an actual error, and if so, to identify precisely the nature and cause of the error, so that it can be properly resolved.

Often, emphasis is placed only on resolving a single error. However, the single error could be representative of other similar errors originating from the same incorrect assumptions, or it could indicate the presence of serious problems in the development process. Correcting only the single error and not addressing underlying problems may cause further complications later in the system life cycle.

Software Hazard Analysis

The overall objective of the software hazard analysis process is to ensure that software hazards and hazards related to interfaces between the software and the computer system have either been eliminated or their risk has been mitigate. This process uses the system requirements to identify software hazards and evaluate the risk of software hazards, and then eliminates or reduces the risk of hazards. It begins before the software requirements process and ends when its objectives have been met. The software hazard analysis process produces a software safety plan and documents that report on the results of the software hazard analysis.

A pre-requisite for any critical system is an analysis of the hazards or threats against which the system must protect. Software hazard analysis (e.g., software fault tree analysis) is an integral part of the system hazard analysis, and both should be conducted in order to assure that all hazards have been identified. Both types of hazard analysis are essential in designing a system for fail-safe operation (e.g., protection against division by zero). In response to a software hazard analysis, some software requirements, software design features, or software code may be changed.

Other examples of items that must be identified in the software hazard analysis include:

- input conditions which could lead to the software causing an unsafe state
- failure modes related to instructions in code which could lead to an unsafe subsystem failure.

System Functionality, Usability, and Complexity

System functionality can become irrelevant, if the system is unusable. Poor interface design between the system user and the machine (computer) may have more consequences than the lack of functions. For example, if the formats of input and output are incomprehensible, the user will stop using the system and revert to previous methods.

In a sense, functionality itself can determine usability; if the functions provided do not match task requirements, a system will not be usable. There is a growing body of evidence that shows that providing extensive functionality is not enough; people must understand what the functions do and how to use them.

Poor usability may jeopardize the utility of a system if it causes some users to give up on it. Usability is at least as important as functionality. Training, accessibility, and availability of terminals, and the culture of the workplace all have an impact. These are organizational factors beyond the designers control. Slow response time during periods of heavy usage may also be beyond the designers control.

Factors that the designer can control include;

- providing functions that match task requirements
- determining the details of screen design
- menu interaction techniques, system feedback
- dynamics of user-system interaction
- input and output formats.

Even experienced users are penalized by poor design. Experienced users can be found spending time and effort in correcting errors. Investing in usability is as important as investing in functionality. Not considering usability can lead to system failure. As an integral part of system design, usability contributes to overall system functionality by making it accessible to users and facilitates effective use of functional capabilities.

For high-volume, structured tasks, improved usability can have significant, measurable effects. For computer systems used for large-scale transactions, seemingly small improvements in usability can translate into large cost savings. Savings as little as one second per transaction can mean a savings of thousands of dollars as well as significantly improved productivity.

One of the reasons that the development of software complexity measures is so difficult is that programming behaviours are poorly understood. A behaviour must be understood before what makes it difficult can be determined. Most complexity measures have been designed without regard to the programmer, task, problem area, or environment and ignore the stylistic characteristics of the program. In fact, program complexity can be dependent on nonprogrammer factors.

The large majority of software complexity measures have been developed with little regard for the programmer, the programming task, or the programming environment. It should include studying programmers and the means they use to construct, understand, and modify programs.

Tools are available to predict program length, program development time, number of errors, and future cost of program maintenance. Any measure can be made. The properties of a software metric criticality determine the ways in which it can be used. This dependency must be kept in mind by those who are developing metrics and those who use metrics.

Software Project Management & Metrics

Course Corrections

No project stands still and very few projects proceed according to plan. Thus, mid-point reviews and other check points are needed to bring a project back on schedule. During these course corrections, timely and proactive action by project management will prevent delays or worse:

Assess the situation

Identify problems

Isolate causes for problems

Allocate resources to fix problems

Project Communication

Project communication can take place in several ways:

Meetings

Reports

Etc.

Communication provides reporting guidelines to project members and is a core mechanism for maintaining quality and consistency in reporting. Meeting guidelines for a project should include:

Each meeting should have a defined purpose

Agendas should be available to members prior to the meeting

The "right" people required for action should constitute the meeting's membership

Meeting groups should be relatively small (less than 10 is optimum)

Meeting length should be no longer than 1 hour

Each meeting should have a definite conclusion

Set clear goals for the next meeting

Quality Assurance

1. Overview

Quality assurance is the forecasting and prevention of quality related problems. A project quality requirements document will incorporate the customer's quality requirements. Every effort is taken to meet the customer quality requirements. Examples of quality assurance tools include:

Policies and procedures

Supervision

Periodic checking and rechecking

Reviews

Checklists

Walk-throughs

Quality assurance is concerned with the forecasting and prevention of quality problems. It is pro-active.
Quality control is concerned with detecting and reporting quality-related problems. It is reactive.

Examples of quality control tools include:

Inspection tools

Measurement gauges

Acceptance tests

Statistical process-control charts with upper and lower control limits

Software quality assurance is a planned systematic pattern for all actions necessary to provide adequate confidence that the product, or process by which the product is developed, conforms to established requirements.

The primary goal of software quality assurance is to enhance the quality of software. To this end, a new concept, Clean Room engineering, is evolving where by software is developed to assure quality. With Clean Room engineering, programmers do not compile their code. Instead they spend more time on design using a "box structure" method, and on analyzing their own work. When the programmers are confident of their work, it is submitted to another group, who then compiles and tests the code. Clean Room processing shows a lower error rate in the finished product and an increase in productivity across the life cycle of the application.

2. Examining the Need for Software Quality Assurance

Some symptoms leading to establishment of a software quality assurance function include (1) when the software is exhibiting many errors, (2) when users stop using the software, (3) when customers express quality-related problems, (4) when sales and profits are declining. Under these conditions, a software quality assurance function should be established to assure quality.

For example, periodic checks of test results (e.g., number of errors found) may reveal an error rate outside the control limits. A reviewer may discover that this is due to the fact that inspections had been eliminated when the project fell behind schedule.

3. Software Quality Assurance Planning and Standards

Software Quality Assurance Plans

A software quality assurance plan should identify what quality measurements are to be taken, when they are to be examined, by whom, their purpose, and who has the authority to make changes in any of the processes. The purpose could be (1) to determine if there is a problem, (2) to evaluate the effect of change in the development process, and (3) to examine the effectiveness of testing, inspection or other analysis techniques. Major elements of a software quality assurance plan include: organization for quality responsibilities; identification of applicable software products; standards, practices, and conventions; metrics, reviews, and audits; documentation, error reporting and corrective action; records management; security access control; training; tools, techniques and methodologies; and risk management.

Software Quality Assurance Standards

Measurement of the software development process and product is valuable to the developing organization. Metrics may be used to measure quality. To this end, standards improve the quality of software. If a standard requires an activity, the standard should contain requirements for the documentation, review, and monitoring of this activity. Inclusion of statements such as the following would be useful: (1)(each life cycle activity shall be documented; (2) reports shall be prepared on each activity's implementation; (3) a quality assurance review shall verify that the life cycle activity plan is acceptable and was accomplished (or if not, why not); (4) tracking mechanisms shall provide information to project management and reviewers.

4. Software Quality Factors

Software quality factors or attributes are the requirements that software must meet, such as usability, efficiency, reliability, maintainability, and portability.

Usability refers to the ease with which software can be used. It determines the user-friendliness of software.

Efficiency and economy are related. An economical and efficient system uses the minimum number of information resources to achieve the output level required by the system's users. System economy and efficiency must always be considered within the context of system effectiveness.

Reliability is the probability that a given software system operates for some time period, without system failure due to a software fault, on the computer for which it was designed, given that it is used within design limits.

Maintainability is the characteristic of code facilitating an incorporation of changes.

Portability refers to the degree of movement of the software between the same vendor's computer models or across different vendors. It is the ability to operate on multiple computer types from different vendors.

5. Software Quality Assurance

The thrust of software assurance (SQA) is product assurance. Which is important – process or product? Quality of process is related to the quality of the product. New software quality assurance focuses on evaluating the processes by which products are developed or manufactured.

The major objectives of the SQA process are to ensure that the software development and software assurance processes comply with software assurance plans and standards, and to recommend process improvements. The process uses the system requirements, and information about the purpose and criticality of the software, to evaluate the outputs of the software development and software assurance processes. It begins before the software requirements process and ends when its objectives have been met. A software quality assurance plan and review/audit reports are produced during the SQA process.

6. Software Quality Assurance Activities

Major activities in software quality assurance include project management, software verification and validation (SV&V), software configuration management (SCM), and software quality assurance (SQA).

Some activities in project management include (1) Were reviews of project status and assurance task reports conducted? (2) Based on metrics data, were any processes found to be outside statistical control limits? (3) If so, what actions were taken?

Some activities in SV&V include (1) Were the test strategies consistent with the requirements and were they adequate to evaluate the design? (2) Were the test strategies sufficient to determine whether the software is safe and effective? (3) How were the safety features of the software tested and analyzed? (4) Do test results show that safety-related functions were tested adequately and that they perform reliably?

Some activities in SCM include (1) At what point was the first SCM baseline established? (2) What evidence is there that the software delivered is the software that has been verified and validated?

Some activities in SQA include (1) What actions were taken in response to the metrics results? (2) Do SQA reports indicate that standards were properly followed? (3) What error analysis techniques were used? How did the developer or vendor respond to the findings? (4) Were software design and code-inspections performed? (5) Were technical reviews held, and problems identified in the reviews satisfactorily resolved in the product?

7. Software Error Analysis

Software error analysis includes the techniques used to locate, analyze, and estimate errors and data related to errors. It includes the use of error detection techniques, analysis of single errors, data collection, metrics, statistical process-control techniques, error prediction models, and reliability models.

Error detection techniques are techniques of software development, software quality assurance, software verification, validation and testing used to locate anomalies in software products. An anomaly is any condition which departs from the expected. Errors, defects, faults, and failures are considered anomalies. Once an anomaly is detected, analysis is performed to determine if the anomaly is an actual error, and if so, to identify precisely the nature and cause of the error, so that it can be properly resolved.

Often, emphasis is placed only on resolving a single error. However, the single error could be representative of other similar errors originating from the same incorrect assumptions, or it could indicate the presence of serious problems in the development process. Correcting only the single error and not addressing underlying problems may cause further complications later in the system life cycle.

8. Software Hazard Analysis

The overall objective of the software hazard analysis process is to ensure that software hazards and hazards related to interfaces between the software and the computer system have either been eliminated or their risk has been mitigate. This process uses the system requirements to identify software hazards and evaluate the risk of software hazards, and then eliminates or reduces the risk of hazards. It begins before the software requirements process and ends when its objectives have been met. The software hazard analysis process produces a software safety plan and documents that report on the results of the software hazard analysis.

A pre-requisite for any critical system is an analysis of the hazards or threats against which the system must protect. Software hazard analysis (e.g., software fault tree analysis) is an integral part of the system hazard analysis, and both should be conducted in order to assure that all hazards have been identified. Both types of hazard analysis are essential in designing a system for fail-safe for schedulerration (e.g., protection against division by zero). In response to a software hazard analysis, some software requirements, software design features, or software code may be changed.

Other examples of items that must be identified in the software hazard analysis include (1) input conditions which could lead to the software causing an unsafe state and (2) failure modes related to instructions in code which could lead to an unsafe subsystem failure.

9. System Functionality, Usability, and Complexity

System functionality can become irrelevant, if the system is unusable. Poor interface design between the system user and the machine (computer) may have more consequences than the lack of functions. For example, if the formats of input and output are incomprehensible, the user will stop using the system and revert to previous methods.

In a sense, functionality itself can determine usability; if the functions provided do not match task requirements, a system will not be usable. There is a growing body of evidence that shows that providing extensive functionality is not enough; people must understand what the functions do and how to use them.

Poor usability may jeopardize the utility of a system if it causes some users to give up on it. Usability is at least as important as functionality. Training, accessibility, and availability of terminals, and the culture of the workplace all have an impact. These are organizational factors beyond the designers control. Slow response time during periods of heavy usage may also be beyond the designers control.

Factors that the designer can control include; providing functions that match task requirements, determining the details of screen design, menu interaction techniques, system feedback, dynamics of user-system interaction, and input and output formats.

Even experienced users are penalized by poor design. Experienced users can be found spending time and effort in correcting errors. Investing in usability is as important as investing in functionality. Not considering usability can lead to system failure. As an integral part of system design, usability contributes to overall system functionality by making it accessible to users and facilitates effective use of functional capabilities.

For high-volume, structured tasks, improved usability can have significant, measurable effects. For computer systems used for large-scale transactions, seemingly small improvements in usability can translate into large cost savings. Savings as little as one second per transaction can mean a savings of thousands of dollars as well as significantly improved productivity.

One of the reasons that the development of software complexity measures is so difficult is that programming behaviours are poorly understood. A behaviour must be understood before what makes it difficult can be determined. Most complexity measures have been designed without regard to the programmer, task, problem area, or environment and ignore the stylistic characteristics of the program. In fact, program complexity can be dependent on nonprogrammer factors.

The large majority of software complexity measures have been developed with little regard for the programmer, the programming task, or the programming environment. It should include studying programmers and the means they use to construct, understand, and modify programs.

Tools are available to predict program length, program development time, number of errors, and future cost of program maintenance. Any measure can be made. The properties of a software metric criticality determine the ways in which it can be used. This dependency must be kept in mind by those who are developing metrics and those who use metrics.

SOFTWARE ANOMALIES

1. Cost Impact of Software Defects

Ideally, software development processes should be so advanced that no errors will enter a software system during development. Current practices can only help to reduce the number of errors, not prevent *all* errors. However, even if the best practices were available, it would be risky to assume that no errors may enter a system as long as people are involved in the software development and maintenance processes.

The use of error analysis allows for early error detection and correction. When an error made early in the system life cycle goes undetected, problems and costs can accrue rapidly. An incorrectly stated requirement may lead to incorrect assumptions in the design, in turn causing subsequent errors in the code. It may be difficult to catch all errors during testing, since exhaustive testing of the software under all circumstances with all possible input data sets, is not possible. Therefore, even a critical error may remain undetected and may be delivered along with the final product. This undetected error may subsequently cause a system failure resulting in costs not only to fix the error but also for the system failure itself (e.g., plant shutdown, loss of life).

Sometimes, the cost of fixing an error may result in a decision not to fix the error. This is particularly true if the error is found late in the system life cycle. For example, when an error has caused a failure during system test and the location of the error is found to be in the requirements or design correcting that error can be expensive. Sometimes the error is allowed to remain and a fix deferred until the next version of the software. Persons responsible for these decisions may justify them simply on the basis of cost or on an analysis showing that the error, even when exposed, will not cause a critical failure. Decision makers must have confidence in the analysis used to identify the impact of the error, especially for software used in high integrity systems.

A strategy for avoiding the high costs of fixing errors late in the system life cycle is to prevent the situation from occurring altogether, by detecting and correcting errors as early as possible. Studies have shown that it is much more expensive to correct software requirements deficiencies late in the development effort than it is to have correct requirements from the beginning. In fact, the cost to correct a defect found late in the system life cycle may be more than one hundred times the cost to detect and correct the problem when the defect was born. In addition to the lower cost of fixing individual errors, another cost benefit of performing error analysis early in the development is that the error propagation rate will be lower, resulting in fewer errors to correct in later phases. Thus, while error analysis at all phases is important, there is no better time, in terms of cost benefit, to conduct error analysis than during the software requirements phase.

2. Defect Amplification and Removal

Removing of errors takes place after an anomaly has been discovered using any error detection technique. Analysis of an anomaly will not only aid in the removal of errors related to the anomaly, but will also help detect other similar errors which have not yet manifested themselves. In addition, information obtained from this analysis can provide valuable feedback that may improve subsequent efforts and development processes in future projects.

An anomaly is any condition departing from the expected. Errors, defects, faults, and failures are considered anomalies. The handling of an anomaly generally follows three steps: identification, investigation, and resolution. An anomaly may or may not be an error, it may be the result of misinterpreting test results, or the anomaly may be caused by a problem external to the software under analysis (e.g., the modem used for testing was not configured properly).

STEPS IN HANDLING SOFTWARE ANOMALIES		
Identification (collects information	Investigation (identifies cause of the	Resolution (steps taken to correct

about errors)	error)	the error)

Identification. As soon as an anomaly is detected, information about it should be recorded to help identify, analyze, and correct the anomaly. Typically, this information is presented in an anomaly, or problem report. While the formats may differ, reports should include the following types of information: locator, date and time, activity, phase error encountered, status of product, repeatability of error, symptom, location of symptom, severity level (e.g., critical, serious, major, moderate, no problem).

Investigation. Following the identification stage, all errors should be investigated to obtain further information on the nature and cause in order to propose solutions for resolution action or corrective action. Information that should be recorded during this stage include the following: phase error introduced, type of error found, location of error, cause of error, units affected, priority level (e.g., resolve error immediately, error will be placed in normal queue, error will be fixed last, error will not be fixed).

Resolution. Error resolution consists of the steps to correct the error. The policy of the project determines if the person who investigates the error will also correct the error. The procedures for distribution and retention of the error information is also identified by the policy. Typically, the recipients of the error information are the project manager, software quality assurance manager, database manager, and the customer (user). The amount of formalism (e.g., whether the plan needs to be documented) depends on the scope, risk, and size of the project. For small errors in small projects, the scheme may not be necessary. The resolution plan should contain the item to be fixed, estimated date or time to start and complete the fix, and personnel responsible for performing the fix and for follow-up.

FORMAL TECHNICAL REVIEWS

Overview

Formal reviews are conducted at the end of each life cycle phase or at the end of the planning period on the results or products of the phase or period. They may also be conducted when a serious problem or concern arises. Two types of formal reviews are available: management reviews and technical reviews.

TYPES OF FORMAL REVIEWS	
Management Reviews	Technical Reviews

(examines the project plan and status)	(examines the product)

Management reviews formally evaluate a project plan or project status relative to that plan. Management reviews have two purposes. The first is to ensure the adequacy and completeness of each planning document for meeting project requirements. The second is to ensure that project activities are progressing according to the planning documents, identify the need for corrective action to the plan or the project, and ensure proper allocation of resources. The results of the management reviews are summarized in a management review report and are auditable and traceable to and from the appropriate planning documents.

In contrast, the formal **technical review** examines the product, and the result of any assurance activities already conducted n the product. The purpose of technical reviews is to evaluate the software elements such as requirements and design to ensure conformity to its specifications, compliance of the development of the software elements with its plans, and the integrity of changes to the software elements. The results of the technical reviews are summarized in technical review reports which are auditable and traceable to and from the appropriate planning documents.

Review Techniques
Four popular review techniques will be discussed in this section, including inspections, tractability analysis (tracing), reviews, and walk-throughs.

Inspections

A Fagan inspection is an evaluation technique in which software requirements, design, code, or other products are examined by a person or group other than the author to detect faults, violations of development standards, and other problems. An inspection begins with the distribution of the item to be inspected (e.g., specification, some code, test data). Each participant is required to analyze the item on his own. During the inspection, which is the meeting of all the participants, the item is jointly analyzed to find as many errors as possible. All errors found are recorded, but no attempt is made to correct the errors at that time. However, at some point in the future, it must be verified that the errors found have actually been corrected. Inspections may also be performed in the design and implementation phases. *The types of errors detected include: weak modularity, failure to handle exceptions, nonexisting error traps, incomplete requirements, infeasible requirements, conflicting requirements, and incorrect specification of resources.*

Advantages of Inspections	Disadvantages of Inspections
Provides comprehensive statistics on classes of errors Increases product quality Effective for projects of all sizes	Inspectors must be independent of programmers Programmers may feel inspection is a personal attack on their work Time consuming task

Traceability Analysis

There are several types of traceability analysis, including requirements trace, design trace, code trace, and test trace. Traceability analysis is the process of verifying that each specified requirement has been implemented in the design/code, that all aspects of the design/code have their basis in the specified requirements, and that testing produces results compatible with the specified requirements. *The types of errors detected include: omitted functions, higher-order requirement improperly translated, software specification incompatible with other system specifications, omission or misinterpretation of specified requirements, detailed design not in conformity to general design, failure to conform to standards, code not in conformity to detailed design, and software not performing functions and producing outputs in conformity with requirement specifications.*

Advantages of Traceability Analysis	Disadvantages of Traceability Analysis
Highly effective for detecting errors during design and implementation phases Valuable aid in verifying completeness, consistency, and testability of software Aids in retesting software when a system requirement has been changed	No significant disadvantages

Reviews

A review is a meeting at which the requirements, design, code, or other products of a software development project are presented to the user, sponsor, or other interested parties for comment and approval, often as a prerequisite for concluding a given phase of the software development process. Usually held at end of a phase, it may be called when problems arise. It is often referred to as "formal review" versus desktop review of materials (i.e., desk checking).

Advantages of Reviews	**Disadvantages of Reviews**
Provides opportunity to change course of a project before start of next phase Because scope of review is usually broad, gives opportunity to recognize global problems	If participants do not have materials ahead of time and spend time preparing, review will accomplish little or nothing Attention focus on major issues

Walk-throughs

A walk-through is an evaluation technique in which a designer or programmer leads one or more other members of the development team through a segment of design or code, while the other members ask questions and make comments about technique, style, and identify possible errors, violations of development standards, and other problems. Walk-throughs are similar to reviews, but are less formal. Other essential differences include the following: (1) participants are fellow programmers rather than representatives of other functions, (2) frequently no preparation is required, (3) standards are usually ignored, (4) checklists are rarely used, and (5) follow-up is often Ignored. *The types of errors detected include: interface, logic, data, and syntax.*

Advantages of Walk-throughs	**Disadvantages of Walk-throughs**
Less intimidating than formal reviews Identifies the most error=prone sections of the program, so more attention can be paid to these sections during	Designers or programmers may feel walk-through is an attack on their character or work

testing Very effective in finding logic design and coding errors	

The Review Meeting

Success of a technical review requires that all participants carefully examine the inputs to the technical review prior to the review meeting. In both the management and technical reviews, experts on specific topics (e.g., design experts for design reviews) should be present to lend their knowledge to the review.

Review Reporting and Record-keeping

The review leader performs the administrative functions of the review and issues the technical review report. The recorder documents the findings, decisions, and recommendations of the review team. Review team members are expected to be prepared for the meeting and ensure that the review objectives are met. Recommendations made by the review team should be such that management can act on them quickly. Management is responsible for responding to recommendations promptly.

Review Guidelines

The review leader plans for the review by identifying the review team, schedules a time and place for the review, and distributes all inputs to the review team. An overview of the project is conducted for the review team by a qualified project member. Each review team member studies the software element and related materials. The technical review consists of the review team:

Determining whether or not the software element corresponds to the specifications and standards to which it must adhere, and recording any deviations

Listing issues, recommendations, and responsible individuals for resolving the issues

Identifying other issues that need to be addressed

Documenting the meeting, deficiencies found in the software element, and recommendations for management

Review Checklist

Formal reviews may include reviewing SQA, SV&V, and SCM results in order to examine the product. This review may also help detect whether or not these activities were performed in accordance with their respective plans. A checklist provides guidelines for reviewing both the product and plans for assurance activities.

The following checklist contains questions a reviewer can ask during Software Requirements Review: (1) Do the interface requirements enable compatibility of external interfaces (hardware and software)? (2) Do the functional requirements cover all abnormal situations? (3) Are the specific models, algorithms, and numerical techniques compatible? (4) Are the requirements for the man-machine interface adequate? (5) Is there justification for the design/implementation constraints? (6) Will the design, operation, and maintenance of software be feasible? (7) Is each unique requirement defined more than once? Are there any redundant statements? (8) Are there requirements for fault tolerance and graceful degradation? (9) Are the requirements clear and specific enough to be the basis for detailed design specifications and functional test cases? (10) Are the requirements verifiable (i.e., can the software be checked to see whether requirements have been fulfilled)?

The following checklist contains questins a reviewer can ask during Software Design Review: (1) Are design features consistent with the requirements? (2) Does the design implement required program behavior with respect to each program interface? (3) Are all required processing steps included? (4) Are all possible outcomes of each decision point designated? (5) Does the design take into account all expected situations and conditions? (6) Does the design configuration ensure integrity of changes? (7) Does each program have a single function? (8) Is the design structured so that it comprises relatively small, hierachically related programs or sets of programs, each performing a particular, unique function? (9) Does the design avoid unnecessarily complex designs and design representations?

The following checklist contains questions a reviewer can ask during Source Code Review: (1) Is the code a complete and precise implementation of the design? (2) Was the code integrated and debugged to satisfy the design specification? (3) Does the code create the required database, including the initial data? (4) Are there any unreferenced or undefined variables, constants, or data types? (5) Is the code logically consistent with the design? (6) Does the code conform to specified standards? (7) Are all variables properly specified and used? (8) Are all comments accurate? (9) Are all programs invoked with the correct number of parameters? (10) Are cross-references or data dictionaries included to show variable and constant access by the program? (11) Does code consist of programs with only one entry point and one exit point? (12) Is the code written in a language with well-defined syntax and semantics? (13) Is the code free of unintended infinite loops? (14) Does the code avoid recursion? (15) Does the code protect against detectable run-time errors (e.g., range array index values, division by zero, out of range variable values, and stack overflow)? (16) Do loops only have one entrance? (17) Does the code contain or reference history of all code modifications and the reason for them? (18) Do the comment statements adequately describe each routine, using clear English language? (19) Was a nemonic naming convention used? Does the naming reflect the type of variable? (20) Is the valid range of each variable defined?

The following checklist contains questions a reviewer can ask during Test Readiness Review: (1) Have recommended changes been made to the code as result of source code review or, integration test? (2) Is the error rate sufficiently low to warrant beginning the next type of testing? (3) Are all test cases and procedures complete? (4) Is the test facility ready? Are schedules approved, personnel and physical requirements specified? (5) Have all test tools been checked? (6) Have all test procedures been checked?

SOFTWARE QUALITY METRICS

Metrics

Software quality **metrics** are applicable to all software products and include **completeness**, **correctness**, and **reliability**.

Overall Software Quality Metrics
Completeness Correctness Reliability

A measure of **completeness** is cause and effect graphing, which aids in identifying requirements that are incomplete and ambiguous. Also, it explores inputs and expected outputs of a program and identifies ambiguities. For test purposes, it is useful for evaluating test cases for probability of detecting faults. Some examples of characteristics related to completeness include the number of ambiguities identified and the number of ambiguities remaining.

Some examples of characteristics related to **correctness** include: (1) number of problem reports per system development life cycle phase, (2) number of reported problems per time period, (3) number of open real problems per time period, (4) number of closed real problems per time period, (5) number of unevaluated problem reports, (6) age of open real problem reports, (7) age of unevaluated problem reports, (8) age of real closed problem reports, and (9) rate of error discovery.

Required software **reliability** can be used to provide a reliability rating for a product through examination of the processes used to develop it. At the early planning phases of a project, reliability can be used to measure the trade-offs of cost and degree of reliability.

Measures

Basically, two types of measures exist: **Halstead's software science** and**McCabe's complexity measure.**

Halstead's software science. Cyclomatic complexity may be used to determine the structural complexity of a code module. It is calculated in a manner similar to the static complexity of the design. The difference is that the cyclomatic complexity is calculated from a flowgraph of the module, with an edge added from the exit node to the entry node. Cyclomatic complexity is calculated by counting the number of regions in the graph.

The **Halstead software science** measure of observed program length can be used as a readability indicator. Readability metrics include: (1) number of grammatically incorrect comments, (2) number of misspellings, (3) total number of characters, (4) total number of comments, (5) number of comment characters, and (6) number of code characters. Halstead originally did not count declaration statements, input/output statements, or statement labels. However, later researchers do count the operands in these types of statements.

McCabe's complexity measure. Cyclomatic complexity can be used to measure the completeness of the testing that a programmer must satisfy. Specifically, branch coverage and path coverages are verified for completeness. McCabe's technique can be used to: (1) develop a set of test cases which test every outcome of each decision, and execute a minimal number of distinct paths, and (2) assess the testability of software base on its complexity.

**** Which Measure is What? ****

The Halstead software science measures understandability of a software.

The McCabe's complexity measures testability of a software.

**** Metric Versus Measure ****

A metric is the definition, algorithm or mathematical function used to make a quantitative assessment of product or process.

A measure is the numerical value obtained by either direct or indirect measurement; may also be the input, output, or value of a metric.

SOFTWARE RELIABILITY, AVAILABILITY, AND SAFETY

Software Reliability

The reliability of a computer system is defined as the probability that the system will be able to process work correctly and completely without its being terminated or corrupted. A system does not have to fail (crash) for it to be unreliable. The computer configuration, the individual components comprising the system, and the system's usage are all contributing factors to the overall system reliability. As the reliability of these elements varies, so will the level of system reliability.
Reliability is a fundamental element to the security of computer systems. A failure can decrease or destroy the security of the system. Undesirable events such as denial of information, unauthorized disclosure of information, or loss of money and resources can result from lack of reliability.

Reliability is also related to safety and quality. Reliability is critical to a system where there is a potential for loss of life, health, destruction of property, or damage to the environment. Examples include: health care systems, scheduling of safety inspections, manufacturing process control systems, and air traffic control systems. From a quality perspective, reliability deals with the assessment of impact of lack of reliability on the entire organization. Reliability is part of quality attributes.

Software Availability

The availability of a computer system is a measure of the amount of time that the system is actually capable of accepting and performing a user's work. The terms reliability and availability are closely related and often used (although incorrectly) as synonyms. For example, a system that fails frequently, but is restarted quickly would have high availability, even though its reliability is low. To distinguish between the two, reliability can be thought of as the quality of service and availability as the quantity of service. In other words, availability can be viewed as a component of reliability. An example of availability is the availability of communication ports.

Software Safety and Ergonomics

Software Safety. Software safety is important, since lack of safety considerations in a computer-based application system can cause danger or injury to people, damage to equipment and property. It could also create financial or other loss to people using the system or people affected by the system. For example, an incorrectly programmed and incompletely tested medical diagnosis and treatment prescription system could kill a patient or injure people receiving the treatment. Another example: a process control system in a pharmaceutical company where drugs are incorrectly formulated by a computer system could kill or injure patients due to errors in software. Similarly, incorrect and obsolete documentation, especially after a program change was made, could lead to improperly functioning software, loss of life, failed missions, and lost time.

The overall purpose of the software safety evaluation review is to assess how well the product meets its software quality objectives. Quality objectives include reliability, safety, functionality, maintainability, and reviewability. To perform this evaluation, the reviewers need sufficient information about the product requirements, its development, and its overall quality. The following general types of questions can guide the reviewer regarding the product evaluation:

> How thoroughly has the developer or vendor analyzed the safety of critical functions of the software?
>
> How well has the developer or vendor established the appropriateness of the functions, algorithms, and knowledge on which the software is based?
>
> How carefully has the developer or vendor implemented the safety and performance requirements of the software?

Specific questions for the Requirements and Design Phases include: (1) Are the safety requirements consistent with the hazard analysis? (2) How was failure analysis of the software conducted? (3) Are there requirements for self-supervision of the software? (4) Is there modularity in the design? (5) Have critical components been isolated? (6) Were design reviews performed and documented?

Specific questions for the Implementation Phase include: (1) Do the test cases produce results identical with the expected output? (2) Does the operations procedure manual adequately describe diagnostic procedures and tools? (3) Does the operations procedure manual adequately describe emergency procedures and fault recovery procedures?

Specific questions for maintenance phase include: (1) Is there a formal procedure to start maintenance when problems are found? (2) Is there a formal change control procedure?

Ergonomics. Computer terminal screen design, chair and table height, and lighting conditions are part of ergonomics to make working with a computer easier. Health risks due to restricted range of body movements and the possibility of low frequency radiation such as eye stress, skin rashes, back pain, and visual disorders are involved in computer use.

TOOLS FOR PROJECT MANAGEMENT

Overview

Seven project management tools are discussed in this section, including program evaluation techniques, critical path methods, line-of-balance method, graphical evaluation and review techniques, work breakdown structure, charts and tables, and budgets. There is a time and place for each ones of these tools. The project manager needs to know when to use what tools to obtain the greatest benefit.

Project Management Tools

Program evaluation and review techniques (uses probabilities and three-time estimates, focus is on the time)
Critical path method (uses probabilities and a single-time estimate, focus is on cost)
Work breakdown structure (does not use probabilities, provides a conceptual organization of a project)
Charts and tables (do not use probabilities, focus is on presentation of data)
Budgets (use financial data, focus is on variance analysis)

Program Evaluation and Review Techniques

Overview

Project management frequently uses network diagrams to plan the project, evaluate alternatives, and control large and complex projects toward completion. Program evaluation and review techniques (PERT) require extremely careful plans from the very outset of the project. This allows management to allocate resources to critical areas before they become critical. This will alert a manager to trouble areas or bottlenecks before they become a major problem and the source of a project overrun. PERT also helps to allocate resources, but has no influence on the excellence of the end product.

PERT improves communication upward to the manager and the customer (client). PERT lets the supervisor believe that the project manager is doing a superior job, regardless of how well the project manager is actually performing.

PERT Assumptions

Interrelationships of activities are depicted in a network of directed arcs (arcs with arrows, which denote the sequence of the activities they represent). The **nodes,** called events, represent instants in time when certain activities have been completed and others can then be started. All inwardly-directed activities at a node must be completed before any outwardly-directed activity of that node can be started. A **path** is defined as an unbroken chain of activities from the origin node to some other node. The origin node is the beginning of the project. An **event** is said to have occurred when all activities on all paths directed into the node representing that event have been completed.
Another assumption of PERT is that all activities are started as soon as possible. This assumption may not hold true when scarce resources must be allocated to individual activities.

PERT Applications

The development of a critical path network is accomplished by establishing the major milestones that must be reached. Construction of the network diagram requires identification and recording of the project's internal time dependencies – dependencies that might otherwise go unnoticed until a deadline slips by or impacts other activities. A new activity can be added by identifying its successor and predecessor.

An ordered sequence of events to be achieved would constitute a valid model of the program. The network provides a detailed, systematized plan and time schedule before the project begins. As the project progresses, the time estimates can be refined. A top-down approach is taken when developing the network. The total project is fully planned and all components of the plan are included.

Applications of PERT and CPM

- Construction and maintenance of chemical plant facilities, highways, dams, buildings, railroads, and irrigation systems
- Planning of retooling programs for high volume products in plants such as automotive and appliance plants
- Introduction of a new product
- Installation of a computer system
- Acquisition of a company

Critical path scheduling helps coordinate the timing of activities on paper and helps avert costly emergencies. The network diagram must be developed in detail as much as possible so that discrepancies, omissions, and work coordination problems can be resolved inexpensively, at least to the extent that they can be foreseen.

Project diagrams of large projects can be constructed by sections. Within each section the task is accomplished one arrow at a time by asking and answering the following questions for each job:

What immediately preceded this job?

What immediately succeeds (follows) this job?

What can be concurrent with this job?

If the maximum time available for a job equals its duration, the job is called "critical". A delay in a critical job will cause a comparable delay in the project completion time. A project contains at least one contiguous path of critical jobs through the project diagram from beginning to end. Such a path is called a "critical path".

Typically, only about 10 to 15 percent of the jobs in a large project are critical. The primary purpose of determining the "critical path" is to identify those activities that must be finished as scheduled if the new program or project is to be completed on time. The "critical path" of those activities cannot be delayed without jeopardizing the entire program or project.

If the maximum time available for a job exceeds its duration, the job is called a **floater**. Some floaters can be displaced in time or delayed to a certain extent without interfering with other jobs or the completion of the project. Others, if displaced, will start a chain reaction of displacements downstream in the project.

The technological ordering is impossible if a cycle error exists in the job data (i.e., job ‘a’ preceded ‘b’, ‘b’ precedes ‘c’, and ‘c’ precedes ‘a’). The time required to traverse each arrow path is the sum of the times associated with all jobs on the path. The critical path (or paths) is the longest path in time from start to finish; it indicates the minimum time necessary to complete the entire project.

In order to accurately portray all predecessor relationships, "dummy jobs" must often be added to the project graph. The critical path is the bottleneck route, only by finding ways to shorten jobs along the critical path can the overall project time be reduced; the time required to perform non-critical jobs is irrelevant from the viewpoint of total project time.

The PERT Approach
The status of a project at any time is a function of several variables such as resources, performance, and time. Resources are in the form of dollars, or what "dollars" represent – manpower, materials, energy, and methods of production; technical performance of systems, subsystems, and components. An optimum schedule is one that would properly balance resources, performance, and time.

Information concerning the inherent difficulties and variability in the activity being estimated are reflected in the three numbers: the optimistic, pessimistic, and most likely elapsed time estimates should be obtained for each activity. The purpose of the analysis is to estimate, for each network event, the expected times (mean or average) and calendar time of occurrence (PTY).

When PERT is used on a project, the three time estimates (optimistic, most likely, and pessimistic) are combined to determine the expected duration and the variance for each activity.

Optimistic: An estimate of the minimum time an activity will take. This is based on everything "going right the first time". It can be obtained under unusual, good luck situations.

Most likely: An estimate of the normal time an activity will take, a result which would occur most often if the activity could be repeated a number of times under similar circumstances.

Pessimistic: An estimate of the maximum time an activity will take, a result which can occur, only if unusually bad luck is experienced.

The expected times determine the critical path, and the variances for the activities on this path are summed to obtain the duration variance for the project. A probability distribution for the project completion time can be constructed from this information. However, the variances of activities which do not lie on the critical path are not considered when developing the project variance, and this fact can lead to serious errors in the estimate of project duration.

An estimate of the length of an activity is an uncertain one. A stochastic model can be used to reflect this uncertainty. This model measures the possible variation in activity duration. This may take the form of a distribution showing the various probabilities that an activity will be completed in its various possible completion times. Alternatively, this may be non-distribution such as range or standard deviation.

The expected time = 1/6 (a + 4m + b)

where ‘a’ is optimistic time, ‘m’ is most likely time, and ‘b’ is pessimistic time. The expected activity times derived from a three estimate, PERT-type calculation provides a more accurate estimate and allows the activity time variance to be calculated and included in the estimates of project duration.

The latest calendar time at which an event must be accomplished so as not to cause a slippage in meeting a calendar time for accomplishing the objective event is referred to as the ‘latest time’, and denoted as TL. The difference between the latest and expected times, TL – PTY, is defined as **"slack"**. Slack can be taken as a measure of scheduling flexibility that is present in a work flow plan, and the slack for an event also represents the time interval in which it might be reasonably be scheduled. Slack exists in a system as a consequence of multiple path junctures that arise when two or more activities contribute to a third.

- A slack time is a free time associated with each activity as it represents unused resources that can be diverted to the critical path.
- Non-critical paths have slack time while critical paths have no such slack time.

A slack is extra time available for all events and activities not on the critical path. A negative slack condition can prevail when a calculated end date does not achieve a program date objective established earlier.

The manager must determine valid means of shortening lead times along the critical path by applying new resources or additional funds that are obtained from those activities that can "afford" it because of their slack condition. "Safety factor" is another name for "slack". Alternatively, the manager can reevaluate the sequencing of activities along the critical path. If necessary, those activities which were formerly connected in a series can be organized on a parallel or concurrent basis, with the associated trade-off risks involved. Alternatively, the manager may choose to change the scope of work of a critical path alternative in order to achieve a given schedule objective.

When some events have **zero slack**, it is an indication that the expected and latest times for these events are identical. If the zero-slack events are joined together, they will form a path that will extend from the present to the final event. This path can be looked upon as "the critical path". Should any event on the critical path slip beyond its expected date of accomplishment, then the final event can be expected to slip a similar amount. The paths having the greatest slack can be examined for possible performance or resource trade-offs.

When jobs or operations follow one after another, there is no slack. Sub-critical events, where the criteria for defining a sub-critical event relates to the amount of slack involved in the event. Those events having as much as five weeks slack have been deemed sub-critical.

The PERT analysis permits a quantitative evaluation of conceivable alternatives. Each job in the project is represented by an arrow that depicts: (1) the existence of the job, and (2) the direction of time-flows from the tail to the head of the arrow. The arrows are then connected to show graphically the sequence in which the jobs in the project must be performed. The junctions where arrows meet are called events. These are points in time when certain jobs are completed and others must begin.

The difference between a job’s early start and its late start (or between early finish and late finish) is called total slack (TS). Total slack represents the maximum amount of time a job may be delayed beyond its early start without necessarily delaying the project’s completion time.

Key Concepts to Remember – PERT Time Dimensions

ES = Earliest start time for a particular activity

EF = Earliest finish time for a particular activity

EF = ES + t, where 't' is expected activity time for the activity

LS = Latest start time for a particular activity

LF = Latest finish time for a particular activity

LS = LF – t, where 't' is expected activity time for the activity

Slack = LS – ES = LF – EF

The manager examines the work demand and indicates if sufficient resources are available to accomplish all jobs by their early finish. If resources are insufficient, activities are rescheduled within their late finish, using project priority, and available slack. Later, the manager is asked for additional resources or for a decision to delay an activity beyond its late finish.

Critical jobs are those on the longest path throughout the project. That is, critical jobs directly affect the total project time. If the target date (T) equals the early finish date for the whole project (F), then all critical jobs will have zero total slack. There will be at least one path going from start to finish that includes critical jobs only, i.e., the critical path. There could be two or more critical paths in the network, but only one at a time.

If T is greater (later) than F, then the critical jobs will have total slack equal to T minus F. This is a minimum value; since the critical path includes only critical jobs, it included those with the smallest TS. All non-critical jobs will have greater total slack.

Another kind of slack is **free slack** (FS). It is the amount a job can be delayed without delaying the early start of any other job. A job with positive total slack may or may not also have free slack, but the latter never exceeds the former. For purposes of computation, the free slack of a job is defined as the difference between the job's EF time and the earliest of the ES times of all its immediate successors.

When a job has zero total slack, its scheduled start time is automatically fixed (i.e., ES + LS); and to delay the calculated start time is to delay the whole project. Jobs with positive total slack, however, allow the scheduler some discretion in establishing their start times. This flexibility can be applied to smoothing work schedules.

Peak load may be relieved by shifting jobs on the peak days to their late starts. Slack allows this kind of juggling without affecting project time.
Possible data errors in PERT:

The estimated job time may be in error

The predecessor relationship may contain cycle errors (job ‘a’ is a predecessor for ‘b’, ‘b’ is a predecessor for ‘c’, and ‘c’ is a predecessor for ‘a’)

The list of prerequisites for a job may include more than the immediate prerequisites; (e.g., job ‘a’ is a predecessor of ‘b’, ‘b’ is a predecessor of ‘c’, and ‘a’ and ‘b’ both are predecessors of ‘c’)

Some predecessor relationships may be overlooked

Some predecessor relationships may be listed that are spurious

The errors in the PERT calculated project’s mean and standard deviation will tend to be large if many non-critical paths each have a duration approximately equal to the duration of the critical path. However, the more slack time there is in each of the non-critical paths, the smaller will be the error.

One way to minimize errors and omissions is to continually back-check the data and challenge the assumptions. The following table presents advantages and disadvantages of PERT:

Advantages of PERT	**Limitations of PERT**
• Greatly improved control over complex development work and production programs • Capacity to distill large amounts of data in brief, orderly fashion • Requires a great deal of planning to create a valid network • Represents the advent of the management-by-exception principle	• There is little interconnection between the different activities pursued • Requires constant updating and reanalysis of schedules and activities • Requires greater amount of detail work • Does not contain the quantity information, only time information is

exception principle People in different locations can relate their efforts to the total task requirements of a large program "Downstream" savings are achieved by earlier and more positive action on the part of management in the early stages of the project	available

PERT Implementation Issues

The people and organization of a project are more important considerations than the use of a particular planning and control technique

Consideration should be given to managerial issues such as project organization, personalities of project members, and operating schemes

There is a big difference between the criteria of success for the task to be accomplished and the criteria of success for the management system

The project manager is a miniature general manager. He usually lacks the commensurate authority and depends on various management techniques to carry out his job

The project management approach is the preferred method to deal with one-time defined projects

The qualifications of a person making time estimates must include a thorough understanding of the work to be done

Precise knowledge of the task sequencing is required or planned in the performance of activities

PERT/Cost

Once the network has been established, based upon the project work breakdown structure, costs can be estimated. If the breakdown has been made satisfactorily, it will serve as both an estimating and actual cost accumulation vehicle. PERT/cost adds the consideration of resource costs to the schedule produced by the PERT procedure. The basic PERT handles the problem of time uncertainty while PERT/cost addresses cost uncertainty. Cost uncertainty as it relates to time can be handled by different cost estimates for three-time differences. The ultimate objective is not only to improve planning and control, but also to assess possibilities for ''trading off'' time and cost, i.e., adding or subtracting from one at the expense of the other.

There is an "optimum" time-cost point for any activity or job as indicated by the "U" shape of the curve drawn between total direct cost (on y-axis) versus time (on x-axis). It is assumed that total costs will increase with any effort to accelerate or delay the job away from this point in the case where resource application varies. Crashing the project involves shortening the critical path or paths by operating on those activities which have the lowest time-cost slopes. At least three approaches are available to develop the cost estimates:

A single cost estimate of expected cost

Three cost estimates

Optimum time-cost curves

A **single cost estimate** of expected cost is based upon the assumption of the individual cost elements. The **three-cost estimate** approach determines the "expected cost". The advantage of the three-cost estimate over the single cost estimate is that the result is subject to probability analysis. With this expected cost, the manager cannot assume that he has the optimum time-cost mix.

The third approach to estimation is the **optimum time-cost curve concept**. This is differential costing with time as the variability factor. The intention of this approach is to optimize time and costs by using optimum estimated costs. It assumes there is a direct relationship between time and costs on any activity. This relationship can be expressed by a continuous curve. The method is also based upon the concept that activities are subject to time-cost trade-offs. The optimum time-cost curve method is difficult to put into practice due too the need to develop continuous time-cost curves.

Tools for Project Management

Overview

Seven project management tools are discussed in this section, including program evaluation techniques, critical path methods, line-of-balance method, graphical evaluation and review techniques, work breakdown structure, charts and tables, and budgets. There is a time and place for each ones of these tools. The project manager needs to know when to use what tools to obtain the greatest benefit.

Project Management Tools

Program evaluation and review techniques (uses probabilities and three-time estimates, focus is on the time)
Critical path method (uses probabilities and a single-time estimate, focus is on cost)
Work breakdown structure (does not use probabilities, provides a conceptual organization of a project)
Charts and tables (do not use probabilities, focus is on presentation of data)
Budgets (use financial data, focus is on variance analysis)

Program Evaluation and Review Techniques

Overview

Project management frequently uses network diagrams to plan the project, evaluate alternatives, and control large and complex projects toward completion. Program evaluation and review techniques (PERT) require extremely careful plans from the very outset of the project. This allows management to allocate resources to critical areas before they become critical. This will alert a manager to trouble areas or bottlenecks before they become a major problem and the source of a project overrun. PERT also helps to allocate resources, but has no influence on the excellence of the end product.
PERT improves communication upward to the manager and the customer (client). PERT lets the supervisor believe that the project manager is doing a

superior job, regardless of how well the project manager is actually performing.

PERT Assumptions

Interrelationships of activities are depicted in a network of directed arcs (arcs with arrows, which denote the sequence of the activities they represent). The **nodes,** called events, represent instants in time when certain activities have been completed and others can then be started. All inwardly-directed activities at a node must be completed before any outwardly-directed activity of that node can be started. A **path** is defined as an unbroken chain of activities from the origin node to some other node. The origin node is the beginning of the project. An **event** is said to have occurred when all activities on all paths directed into the node representing that event have been completed.

Another assumption of PERT is that all activities are started as soon as possible. This assumption may not hold true when scarce resources must be allocated to individual activities.

PERT Applications

The development of a critical path network is accomplished by establishing the major milestones that must be reached. Construction of the network diagram requires identification and recording of the project's internal time dependencies – dependencies that might otherwise go unnoticed until a deadline slips by or impacts other activities. A new activity can be added by identifying its successor and predecessor.

An ordered sequence of events to be achieved would constitute a valid model of the program. The network provides a detailed, systematized plan and time schedule before the project begins. As the project progresses, the time estimates can be refined. A top-down approach is taken when developing the network. The total project is fully planned and all components of the plan are included.

Applications of PERT and CPM
• Construction and maintenance of chemical plant facilities, highways, dams, buildings, railroads, and irrigation systems
• Planning of retooling programs for high volume products in plants such as automotive and appliance plants
• Introduction of a new product
• Installation of a computer system
• Acquisition of a company

Critical path scheduling helps coordinate the timing of activities on paper and helps avert costly emergencies. The network diagram must be developed in detail as much as possible so that discrepancies, omissions, and work coordination problems can be resolved inexpensively, at least to the extent that they can be foreseen.

Project diagrams of large projects can be constructed by sections. Within each section the task is accomplished one arrow at a time by asking and answering the following questions for each job:

1. What immediately preceded this job?
2. What immediately succeeds (follows) this job?
3. What can be concurrent with this job?

If the maximum time available for a job equals its duration, the job is called "critical". A delay in a critical job will cause a comparable delay in the project completion time. A project contains at least one contiguous path of critical jobs through the project diagram from beginning to end. Such a path is called a "critical path".

Typically, only about 10 to 15 percent of the jobs in a large project are critical. The primary purpose of determining the "critical path" is to identify those activities that must be finished as scheduled if the new program or project is to be completed on time. The "critical path" of those activities cannot be delayed without jeopardizing the entire program or project.

If the maximum time available for a job exceeds its duration, the job is called a **floater**. Some floaters can be displaced in time or delayed to a certain extent without interfering with other jobs or the completion of the project. Others, if displaced, will start a chain reaction of displacements downstream in the project.

The technological ordering is impossible if a cycle error exists in the job data (i.e., job ‘a’ preceded ‘b’, ‘b’ precedes ‘c’, and ‘c’ precedes ‘a’). The time required to traverse each arrow path is the sum of the times associated with all jobs on the path. The critical path (or paths) is the longest path in time from start to finish; it indicates the minimum time necessary to complete the entire project.

In order to accurately portray all predecessor relationships, "dummy jobs" must often be added to the project graph. The critical path is the bottleneck route, only by finding ways to shorten jobs along the critical path can the overall project time be reduced; the time required to perform non-critical jobs is irrelevant from the viewpoint of total project time.

The PERT Approach

The status of a project at any time is a function of several variables such as resources, performance, and time. Resources are in the form of dollars, or what "dollars" represent – manpower, materials, energy, and methods of production; technical performance of systems, subsystems, and components. An optimum schedule is one that would properly balance resources, performance, and time.

Information concerning the inherent difficulties and variability in the activity being estimated are reflected in the three numbers: the optimistic, pessimistic, and most likely elapsed time estimates should be obtained for each activity. The purpose of the analysis is to estimate, for each network event, the expected times (mean or average) and calendar time of occurrence (PTY).

When PERT is used on a project, the three time estimates (optimistic, most likely, and pessimistic) are combined to determine the expected duration and the variance for each activity.

Optimistic: An estimate of the minimum time an activity will take. This is based on everything "going right the first time". It can be obtained under unusual, good luck situations.

Most likely: An estimate of the normal time an activity will take, a result which would occur most often if the activity could be repeated a number of times under similar circumstances.

Pessimistic: An estimate of the maximum time an activity will take, a result which can occur, only if unusually bad luck is experienced.

The expected times determine the critical path, and the variances for the activities on this path are summed to obtain the duration variance for the project. A probability distribution for the project completion time can be constructed from this information. However, the variances of activities which do not lie on the critical path are not considered when developing the project variance, and this fact can lead to serious errors in the estimate of project duration.

An estimate of the length of an activity is an uncertain one. A stochastic model can be used to reflect this uncertainty. This model measures the possible variation in activity duration. This may take the form of a distribution showing the various probabilities that an activity will be completed in its various possible completion times. Alternatively, this may be non-distribution such as range or standard deviation.

The expected time = 1/6 (a + 4m + b)

where 'a' is optimistic time, 'm' is most likely time, and 'b' is pessimistic time. The expected activity times derived from a three estimate, PERT-type calculation provides a more accurate estimate and allows the activity time variance to be calculated and included in the estimates of project duration.

<u>Example:</u> A company is planning a multi-phase construction project. The time estimates for a particular phase of the project are:
Optimistic 2 months
Most likely 4 months
Pessimistic 9 months
Question: Using PERT, what is the expected completion time for this particular phase?
Answer: The expected completion time would be 4.5 months, as shown below:
The expected time = 1/6 (a + 4m + b) = 1/6 (2 + 4x4 + 9) = 27/6 = 4.5.

The latest calendar time at which an event must be accomplished so as not to cause a slippage in meeting a calendar time for accomplishing the objective event is referred to as the 'latest time', and denoted as TL. The difference between the latest and expected times, TL – PTY, is defined as **"slack"**. Slack can be taken as a measure of scheduling flexibility that is present in a work flow plan, and the slack for an event also represents the time interval in which it might be reasonably be scheduled. Slack exists in a system as a consequence of multiple path junctures that arise when two or more activities contribute to a third.

- A slack time is a free time

it represents unused resources that can be diverted to the critical path.

- Non-critical paths have slack time while critical paths have no such slack time.

A slack is extra time available for all events and activities not on the critical path. A negative slack condition can prevail when a calculated end date does not achieve a program date objective established earlier.

The manager must determine valid means of shortening lead times along the critical path by applying new resources or additional funds that are obtained from those activities that can "afford" it because of their slack condition. "Safety factor" is another name for "slack". Alternatively, the manager can reevaluate the sequencing of activities along the critical path. If necessary, those activities which were formerly connected in a series can be organized on a parallel or concurrent basis, with the associated trade-off risks involved. Alternatively, the manager may choose to change the scope of work of a critical path alternative in order to achieve a given schedule objective.

When some events have **zero slack**, it is an indication that the expected and latest times for these events are identical. If the zero-slack events are joined together, they will form a path that will extend from the present to the final event. This path can be looked upon as "the critical path". Should any event on the critical path slip beyond its expected date of accomplishment, then the final event can be expected to slip a similar amount. The paths having the greatest slack can be examined for possible performance or resource trade-offs.

When jobs or operations follow one after another, there is no slack. Sub-critical events, where the criteria for defining a sub-critical event relates to the amount of slack involved in the event. Those events having as much as five weeks slack have been deemed sub-critical.

The PERT analysis permits a quantitative evaluation of conceivable alternatives. Each job in the project is represented by an arrow that depicts: (1) the existence of the job, and (2) the direction of time-flows from the tail to the head of the arrow. The arrows are then connected to show graphically the sequence in which the jobs in the project must be performed. The junctions where arrows meet are called events. These are points in time when certain jobs are completed and others must begin.

The difference between a job’s early start and its late start (or between early finish and late finish) is called total slack (TS). Total slack represents the maximum amount of time a job may be delayed beyond its early start without necessarily delaying the project’s completion time.

Key Concepts to Remember – PERT Time Dimensions

ES = Earliest start time for a particular activity
EF = Earliest finish time for a particular activity
EF = ES + t, where 't' is expected activity time for the activity
LS = Latest start time for a particular activity
LF = Latest finish time for a particular activity
LS = LF – t, where 't' is expected activity time for the activity
Slack = LS – ES = LF – EF

The manager examines the work demand and indicates if sufficient resources are available to accomplish all jobs by their early finish. If resources are insufficient, activities are rescheduled within their late finish, using project priority, and available slack. Later, the manager is asked for additional resources or for a decision to delay an activity beyond its late finish.

Critical jobs are those on the longest path throughout the project. That is, critical jobs directly affect the total project time. If the target date (T) equals the early finish date for the whole project (F), then all critical jobs will have zero total slack. There will be at least one path going from start to finish that includes critical jobs only, i.e., the critical path. There could be two or more critical paths in the network, but only one at a time.

If T is greater (later) than F, then the critical jobs will have total slack equal to T minus F. This is a minimum value; since the critical path includes only critical jobs, it included those with the smallest TS. All non-critical jobs will have greater total slack.

Another kind of slack is **free slack** (FS). It is the amount a job can be delayed without delaying the early start of any other job. A job with positive total slack may or may not also have free slack, but the latter never exceeds the former. For purposes of computation, the free slack of a job is defined as the difference between the job's EF time and the earliest of the ES times of all its immediate successors.

When a job has zero total slack, its scheduled start time is automatically fixed (i.e., ES + LS); and to delay the calculated start time is to delay the whole project. Jobs with positive total slack, however, allow the scheduler some discretion in establishing their start times. This flexibility can be applied to smoothing work schedules.

Peak load may be relieved by shifting jobs on the peak days to their late starts. Slack allows this kind of juggling without affecting project time.
<u>Possible data errors in PERT:</u>

- The estimated job time may be in error

- The predecessor relationship may contain cycle errors (job 'a' is a predecessor for 'b', 'b' is a predecessor for 'c', and 'c' is a predecessor for 'a')
- The list of prerequisites for a job may include more than the immediate prerequisites; (e.g., job 'a' is a predecessor of 'b', 'b' is a predecessor of 'c', and 'a' and 'b' both are predecessors of 'c')
- Some predecessor relationships may be overlooked
- Some predecessor relationships may be listed that are spurious
- The errors in the PERT calculated project's mean and standard deviation will tend to be large if many non-critical paths each have a duration approximately equal to the duration of the critical path. However, the more slack time there is in each of the non-critical paths, the smaller will be the error.

One way to minimize errors and omissions is to continually back-check the data and challenge the assumptions. The following table presents advantages and disadvantages of PERT:

Advantages of PERT	**Limitations of PERT**
• Greatly improved control over complex development work and production programs • Capacity to distill large amounts of data in brief, orderly fashion • Requires a great deal of planning to create a valid network • Represents the advent of the management-by-exception principle • People in different locations can relate their efforts to the total task requirements of a large program • "Downstream" savings are achieved by earlier and more positive action on the part of management in the early stages of the project	• There is little interconnection between the different activities pursued • Requires constant updating and reanalysis of schedules and activities • Requires greater amount of detail work • Does not contain the quantity information, only time information is available

PERT Implementation Issues

- The people and organization of a project are more important considerations than the use of a particular planning and control technique
- Consideration should be given to managerial issues such as project organization, personalities of project members, and operating schemes
- There is a big difference between the criteria of success for the task to be accomplished and the criteria of success for the management system
- The project manager is a miniature general manager. He usually lacks the commensurate authority and depends on various management techniques to carry out his job
- The project management approach is the preferred method to deal with one-time defined projects
- The qualifications of a person making time estimates must include a thorough understanding of the work to be done
- Precise knowledge of the task sequencing is required or planned in the performance of activities

PERT/Cost

Once the network has been established, based upon the project work breakdown structure, costs can be estimated. If the breakdown has been made satisfactorily, it will serve as both an estimating and actual cost accumulation vehicle. PERT/cost adds the consideration of resource costs to the schedule produced by the PERT procedure. The basic PERT handles the problem of time uncertainty while PERT/cost addresses cost uncertainty. Cost uncertainty as it relates to time can be handled by different cost estimates for three-time differences. The ultimate objective is not only to improve planning and control, but also to assess possibilities for ‘‘trading off’’ time and cost, i.e., adding or subtracting from one at the expense of the other.

There is an "optimum" time-cost point for any activity or job as indicated by the "U" shape of the curve drawn between total direct cost (on y-axis) versus time (on x-axis). It is assumed that total costs will increase with any effort to accelerate or delay the job away from this point in the case where resource application varies. Crashing the project involves shortening the critical path or paths by operating on those activities which have the lowest time-cost slopes. At least three approaches are available to develop the cost estimates:

1. A single cost estimate of expected cost
2. Three cost estimates
3. Optimum time-cost curves

A **single cost estimate** of expected cost is based upon the assumption of the individual cost elements. The **three-cost estimate** approach determines the "expected cost". The advantage of the three-cost estimate over the single cost estimate is that the result is subject to probability analysis. With this expected cost, the manager cannot assume that he has the optimum time-cost mix.

The third approach to estimation is the **optimum time-cost curve concept**. This is differential costing with time as the variability factor. The intention of this approach is to optimize time and costs by using optimum estimated costs. It assumes there is a direct relationship between time and costs on any activity. This relationship can be expressed by a continuous curve. The method is also based upon the concept that activities are subject to time-cost trade-offs. The optimum time-cost curve method is difficult to put into practice due too the need to develop continuous time-cost curves.

Critical Path Method

Overview

The critical path method (CPM) is a powerful, but basically simple technique for analyzing, planning, and scheduling large, complex projects. In essence, the tool provides a means for determining (1) which jobs or activities, of the many that comprise a project, are "critical" in their effect on total project time, and (2) how best to schedule all jobs in the project in order to meet a target date at minimum cost. CPM is an extension of PERT>
<u>Characteristics of project for analysis by CPM:</u>

- The project consists of a well-defined collection of jobs or activities which, when completed, mark the end of the project
- The jobs may be started and stopped independently of each other, within a given sequence
- The jobs are ordered in a technological sequence (for example, the foundation of a house must be constructed before the walls are erected)

CPM focuses attention on those jobs that are critical to the project time, it provides an easy way to determine the effects of shortening various jobs in the project, and it enables the project manager to evaluate the costs of a "crash" program.

Time estimates for both normal and crash options are used in the CPM method. Crash time is the time required by the path if maximum effort and resources are diverted to the task along this path. A balance can be obtained when a project manager knows what the normal time and the crash time would be.

As needed, it is a costly practice to "crash" all jobs in a project in order to reduce total project time. If some way is found to shorten one or more of the critical jobs, then no only will the whole project time be shortened but the critical path itself may shift and some previously non-critical jobs may become critical. It is physically possible to shorten the time required by critical jobs by assigning more people to the jobs; working overtime; using different equipment, materials, and technology.

When CPM is used in a project to develop a crashing strategy, two or more paths through the network may have nearly the same length. If the activity duration is allowed to vary, a decrease in the length of the critical path may not result in an equivalent decrease in the project duration, because of the variance inherent in the parallel or alternate paths. These variations of activity times can even allow the alternate path to become a critical path. Thus, simply allowing the activity times to vary slightly from their estimates in order to make the length of the paths different can cause serious errors in a CPM crashing strategy and lead to wasted resources and cost overruns.

Characteristics of CPM Networks

- CPM networks attempt to build the entire project "on paper" at a very early stage of the project – even when the scope is not defined, vaguely defined, or incorrectly defined. In a way, CPM is to the project management what modeling or simulation is to economic studies, production problems, plant design, and transportation problems.
- CPM provides a graphic view of the entire project with completion dates, support activities, and costs affixed to every stage of the project.

The critical path techniques are as valuable on short- and middle-range planning jobs as they are on major and extremely complex projects.

- CPM's single time estimate fails to consider the effects of variability in path completion times on the crashing strategy.
- The CPM chart is an excellent tool for communicating scope as well as details of the job to other persons directly and indirectly concerned with the development and completion of its various phases.

- The CPM chart serves as a permanent record and reminder of the substance of this communication to all management levels.
- The CPM chart shows the timing of management decisions.
- CPM enables the manager to measure progress (or lack of it) against plans and to take appropriate action quickly when needed. And the underlying simplicity of CPM and its ability to focus attention on crucial problem areas of large projects makes it an ideal tool for the senior manager.

CPM versus PERT

CPM and PERT methods are essentially similar in general approach and have much in common. However, important differences in implementation details exist. They are independently derived and based on different concepts. Both techniques define the duration of a project and the relationships among the project's component activities. An important feature of the PERT approach was its statistical treatment of the uncertainty in activity time estimates, which involves the collection of three separate time estimates and the calculation of probability estimates of meeting specified schedule dates.
<u>CPM differs from PERT in two areas:</u>

1. The use of only one time estimate for each activity (and thus no statistical treatment of uncertainty)
2. The inclusion, as an integral part of the overall scheme, of a procedure for time/cost tradeoff to minimize the sum of direct and indirect project costs

<u>Common features of PERT and CPM</u>

1. They both use a network diagram for project representation, in which diagram circles represent activities, with arrows indicating precedence
2. They both calculate early and late start and finish times and slack time

The following table provides a comparison between CPM and PERT:

CPM	**PERT**
1. Uses a single deterministic time estimate to emphasize minimum project costs while downgrading consideration of time restraints 2. 2. It is the choice of cost-conscious managers	1. Uses three time estimates to define a probabilistic distribution of activity times which emphasizes minimum project duration while downgrading consideration of cost restraints 2. It tends to be used by time-conscious managers

While these two techniques are based on different assumptions, they are related to each other because of the obvious relationship between time and cost. <u>*The 'ideal' work technique would combine the concepts of CPM's crashing strategy with PERT's probability distribution of activity times to derive the optimum project duration and cost.*</u>

- **Work Breakdown Structure**

  The work breakdown structure (WBS) was first intended as the common link between schedules and costs in the PERT cost application. Later, it became an important tool for conceptual organization of any project. The WBS provides the necessary logic and formalization of task statements. The WBS prepares the work packages, which usually represent the lowest division of the end items.

WBS allows a project manager to decompose project tasks and activities into a manageable size to facilitate project planning and control. WBS is a hierarchical definition of tasks to be performed and accounted for in a project. Major task areas are defined, and each major area is further identified by defining subtasks. Each subtask may be further broken down so that the lowest level of task definition for each item becomes the basic unit for costing and scheduling. The elements of the WBS must be related to people; someone must always be held responsible and accountable for each element.

The WBS is a checklist of every activity that must be performed to develop an end product. This checklist becomes the foundation for resource allocation, time schedules, and budget plans.

The WBS provides the basis for project organization, cost estimation, task scheduling, and cost reporting. Its goal is to identify stand-alone activities that can be scheduled and costed. The task-segmentation process provides additional understanding of the tasks and increased costing accuracy. The WBS should also correlate with the lowest-level organizational structure for the project. This provides a traceability of resource requirements associated with each task and results in allocation of estimated resource requirements within the organization.

There is a definite interrelationship between the WBS and project schedules. Each WBS element has a definite time-phasing relative to other WBS elements. The scheduling of activities is an iterative process that provides the framework for cost estimating and the basis for the overall project plan.

- **Charts and Tables**

Overview

The basic purpose of a chart or graph is to give a visual comparison between two or more things. For example, changes in budget from one year to the next may be represented in a graph. One significant reason for visualizing a comparison is to reinforce its comprehension.

Charts and graphs are used to dramatize a statement, a fact, a point of view, or an idea. Visual aids assist in the quick comprehension of both simple and complex data, statistics, or problems. A chart should explain itself in silence; it should be completely understood without the assistance of a caption. The caption must act only as a reinforcement to its comprehension.

Various charts such as tabular charts, column charts, bar charts, milestone charts, pie charts, line charts, layer charts, and input/output charts along with decision trees and decision tables are discussed briefly.

Types and Charts and Tables

- Tabular chart (used to represent items of interest)
- Column chart (used for comparison of things)
- Gantt (bar) chart (used to show duration of actions)
- Milestone chart (used to show achievements)
- Pie chart (used to represent a 100 percent of total)
- Line chart (used for comparison of things)
- Layer chart (used for accumulation of individual facts)
- Input/output charts (actual results are compared with forecasts)
- Decision trees (display the sequential nature of decision making)
- Decision tables (documents conditions, rules, and actions)

The **tabular chart** is used to represent items of interest. It requires a fair amount of study in order to grasp the full meaning of the figures. This is because it takes longer to digest the meaning of an itemization of compiled figures than if the same figures are presented graphically. The **column chart** is most commonly used for demonstrating a comparison between two or more things. The column chart is vertical.

The **Gantt chart** is a bar chart and is essentially a column chart on its side, and is used for the same purpose. The bar chart is horizontal. The bar chart is a tool which allows a manager to evaluate whether existing resources can handle work demand or whether activities should be postponed. The Gantt chart is used to show task scheduling where each task has start and completion dates.

Gantt chart is a graphical illustration of a scheduling technique. The structure of the chart shows output plotted against units of time. It does not include cost information. It highlights activities over the life of a project and contrasts actual times with projected times using a horizontal (bar) chart. It gives a quick picture of a project's progress in knowing the status of actual time lines and projected time lines. The following table presents advantages and disadvantages of PERT and Gantt charts.

The following table presents the advantages and disadvantages of PERT and Gantt charts.

Type	PERT	Gantt Chart
Advantages	○ A good planning aid. ○ Interdependencies between activities can be shown. ○ Network diagram is flexible to change. ○ Activity times are probabilistic. ○ A good scheduling tool for large, non-routine projects. ○ A good tool in predicting resource needs, problem areas, and the impact of delays on project completion.	○ A good planning tool. ○ A graphical scheduling technique, simple to develop, use and understand. ○ Useful for large projects. ○ Shows a sequence of steps or tasks. ○ Actual completion times can be compared with planned times.

Type	PERT	Gantt Chart
Disadvantages	○ Difficult to apply to repetitive assembly-line operations where scheduling is dependent on the pace of machines. ○ Large and complex projects are difficult to draw manually. ○ Requires computer hardware and software to draw a complex network. ○ Requires training to use the computer program.	○ Interrelationships among activities are not shown on the chart. ○ Inflexible to change. ○ Activity times are deterministic. ○ Difficult to show

		very complex situations. ○ Cannot be used as a procedure-documenting tool. ○ Does not show the critical path in a chain of activities.

A **milestone chart** represents a major activity or task to be accomplished (e.g., a design phase in a computer system development project).

**** Gantt Chart Versus Milestone Chart ****

- Gantt charts show the duration of tasks.
- Milestone charts show achievements.
- Both Gantt charts and milestone charts show activities against time.

The **pie chart** is used to represent a 100 percent total of two or more items. The **line chart** is exceptionally impressive when comparing several things, but could present a visual problem if the comparisons are too many or too close in relation to one another. Advantages are that it is simple to draw. Disadvantages are that, if the lines are close to each other, it is difficult to distinguish some of the plotted points.

The **layer chart** is linear in appearance but has a different representation. It depicts the accumulation of individual facts stacked one over the other to create the overall total. This chart is more complex than the others, since it illustrates much more. In addition to showing the comparison of layers that add up to the total, this type of chart also shows how each group of layers relates to subsequent groups. The layer chart requires more work to prepare than the other charts. There is more arithmetic involved, and it requires a good deal of concentration to draw the chart.

Input/output charts reflect an actual expenditure (inputs) and accomplishment (outputs) versus a plotted forecast. Each area of a project is assigned a task, and funding and accomplishments for each task are forecasted. Actual results are compared with forecasts.

Decision trees are a graphical representation of possible decisions, events or states of nature resulting from each decision with its associated probabilities, and the outcomes of the events or states of nature. The decision problem displays the sequential nature of the decision-making situation. The decision tree has nodes, branches, and circles to represent junction boxes, connectors between the nodes, and state-of-nature nodes, respectively.

Decision tables (truth tables) are a tool which documents rules used to select one or more actions based on one or more conditions. These conditions and their corresponding actions can be presented either in a matrix or tabular form.

Budgets

Budgets have been a management tool for a long time. Budgets are the financial expression of project tasks. Either incremental budget or zero-based budgeting concepts can be practiced for managing a project. Incremental budgeting starts with data from a similar, previous project and a factor is built-in to increase or decrease the budget for the new project. Zero-based budgeting starts fresh and does not depend on a previous project's data. Variance analysis highlights the differences between actual and budget. Both positive and negative variance needs to be analyzed to understand the root cause of the difference.

INFORMATION SYSTEMS AUDIT TECHNIQUES

PROJECT MANAGEMENT REVIEW

The major objective of the project management process, a part of software assurance process, is to establish the organizational structure of the project and assign responsibilities. The process uses the system requirements documentation and information about the purpose of the software, criticality of the software, required deliverables, and available time and other resources, to plan and manage the software development and maintenance processes. The project management process begins before software development starts and ends when its objectives have been met. The project management process overlaps and often reiterates other software assurance processes. It establishes/approves standards, implements monitoring and reporting practices, develops high-level policy for quality, and cites laws and regulations for compliance.

Review the following activities performed by the project manager in project planning area: (1) set objectives or goals – determine the desired outcome for the project: (a) analyze and document the system and software requirements; define the relationships between the system and software activities, (b) determine management requirements and constraints (resource and schedule limitations), and (c) define success criteria; always include delivery of software that satisfies the requirements, on time and within budget, (2) plan for corrective action, (3) develop project strategies – decide on major organizational goals (e.g., quality) and develop a general program of action for reaching those goals, (4) develop policies for the project – make standing decision on important recurring matters to provide a guide for decision (end of page 830).

Software Management Tools

SOFTWARE ANOMALIES

Cost Impact of Software Defects

Ideally, software development processes should be so advanced that no errors will enter a software system during development. Current practices can only help to reduce the number of errors, not prevent *all* errors. However, even if the best practices were available, it would be risky to assume that no errors may enter a system as long as people are involved in the software development and maintenance processes.

The use of error analysis allows for early error detection and correction. When an error made early in the system life cycle goes undetected, problems and costs can accrue rapidly. An incorrectly stated requirement may lead to incorrect assumptions in the design, in turn causing subsequent errors in the code. It may be difficult to catch all errors during testing, since exhaustive testing of the software under all circumstances with all possible input data sets, is not possible. Therefore, even a critical error may remain undetected and may be delivered along with the final product. This undetected error may subsequently cause a system failure resulting in costs not only to fix the error but also for the system failure itself (e.g., plant shutdown, loss of life).

Sometimes, the cost of fixing an error may result in a decision not to fix the error. This is particularly true if the error is found late in the system life cycle. For example, when an error has caused a failure during system test and the location of the error is found to be in the requirements or design correcting that error can be expensive. Sometimes the error is allowed to remain and a fix deferred until the next version of the software. Persons responsible for these decisions may justify them simply on the basis of cost or on an analysis showing that the error, even when exposed, will not cause a critical failure. Decision makers must have confidence in the analysis used to identify the impact of the error, especially for software used in high integrity systems.

A strategy for avoiding the high costs of fixing errors late in the system life cycle is to prevent the situation from occurring altogether, by detecting and correcting errors as early as possible. Studies have shown that it is much more expensive to correct software requirements deficiencies late in the development effort than it is to have correct requirements from the beginning. In fact, the cost to correct a defect found late in the system life cycle may be more than one hundred times the cost to detect and correct the problem when the defect was born. In addition to the lower cost of fixing individual errors, another cost benefit of performing error analysis early in the development is that the error propagation rate will be lower, resulting in fewer errors to correct in later phases. Thus, while error analysis at all phases is important, there is no better time, in terms of cost benefit, to conduct error analysis than during the software requirements phase.

Defect Amplification and Removal

Removing of errors takes place after an anomaly has been discovered using any error detection technique. Analysis of an anomaly will not only aid in the removal of errors related to the anomaly, but will also help detect other similar errors which have not yet manifested themselves. In addition, information obtained from this analysis can provide valuable feedback that may improve subsequent efforts and development processes in future projects.

An anomaly is any condition departing from the expected. Errors, defects, faults, and failures are considered anomalies. The handling of an anomaly generally follows three steps: identification, investigation, and resolution. An anomaly may or may not be an error, it may be the result of misinterpreting test results, or the anomaly may be caused by a problem external to the software under analysis (e.g., the modem used for testing was not configured properly).

STEPS IN HANDLING SOFTWARE ANOMALIES		
Identification (collects information about errors)	Investigation (identifies cause of the error)	Resolution (steps taken to correct the error)

Identification. As soon as an anomaly is detected, information about it should be recorded to help identify, analyze, and correct the anomaly. Typically, this information is presented in an anomaly, or problem report. While the formats may differ, reports should include the following types of information: locator, date and time, activity, phase error encountered, status of product, repeatability of error, symptom, location of symptom, severity level (e.g., critical, serious, major, moderate, no problem).

Investigation. Following the identification stage, all errors should be investigated to obtain further information on the nature and cause in order to propose solutions for resolution action or corrective action. Information that should be recorded during this stage include the following: phase error introduced, type of error found, location of error, cause of error, units affected, priority level (e.g., resolve error immediately, error will be placed in normal queue, error will be fixed last, error will not be fixed).

Resolution. Error resolution consists of the steps to correct the error. The policy of the project determines if the person who investigates the error will also correct the error. The procedures for distribution and retention of the error information is also identified by the policy. Typically, the recipients of the error information are the project manager, software quality assurance manager, database manager, and the customer (user). The amount of formalism (e.g., whether the plan needs to be documented) depends on the scope, risk, and size of the project. For small errors in small projects, the scheme may not be necessary. The resolution plan should contain the item to be fixed, estimated date or time to start and complete the fix, and personnel responsible for performing the fix and for follow-up.

Formal Technical Software Reviews

Overview

Formal reviews are conducted at the end of each life cycle phase or at the end of the planning period on the results or products of the phase or period. They may also be conducted when a serious problem or concern arises. Two types of formal reviews are available: management reviews and technical reviews.

TYPES OF FORMAL REVIEWS	
Management Reviews (examines the project plan and status)	Technical Reviews (examines the product)

Management reviews formally evaluate a project plan or project status relative to that plan. Management reviews have two purposes. The first is to ensure the adequacy and completeness of each planning document for meeting project requirements. The second is to ensure that project activities are progressing according to the planning documents, identify the need for corrective action to the plan or the project, and ensure proper allocation of resources. The results of the management reviews are summarized in a management review report and are auditable and traceable to and from the appropriate planning documents.

In contrast, the formal **technical review** examines the product, and the result of any assurance activities already conducted n the product. The purpose of technical reviews is to evaluate the software elements such as requirements and design to ensure conformity to its specifications, compliance of the development of the software elements with its plans, and the integrity of changes to the software elements. The results of the technical reviews are summarized in technical review reports which are auditable and traceable to and from the appropriate planning documents.

Review Techniques

Four review techniques:

- inspections
- traceability analysis (tracing)
- reviews
- walk-throughs

Inspections

A Fagan inspection is an evaluation technique in which software requirements, design, code, or other products are examined by a person or group other than the author to detect faults, violations of development standards, and other problems. An inspection begins with the distribution of the item to be inspected (e.g., specification, some code, test data). Each participant is required to analyze the item on his own. During the inspection, which is the meeting of all the participants, the item is jointly analyzed to find as many errors as possible. All errors found are recorded, but no attempt is made to correct the errors at that time. However, at some point in the future, it must be verified that the errors found have actually been corrected. Inspections may also be performed in the design and implementation phases.
Types of errors detected include:

- weak modularity
- failure to handle exceptions
- non-existing error traps
- incomplete requirements
- infeasible requirements
- conflicting requirements
- incorrect specification of resources.

Advantages of Inspections	**Disadvantages of Inspections**
• Provides comprehensive statistics on classes of errors • Increases product quality • Effective for projects of all sizes	• Inspectors must be independent of programmers • Programmers may feel inspection is a personal attack on their work • Time consuming task

Traceability Analysis

There are several types of traceability analysis, including requirements trace, design trace, code trace, and test trace. Traceability analysis is the process of verifying that each specified requirement has been implemented in the design/code, that all aspects of the design/code have their basis in the

specified requirements, and that testing produces results compatible with the specified requirements.
Types of errors detected include:

- omitted functions
- higher-order requirement improperly translated
- software specification incompatible with other system specifications
- omission or misinterpretation of specified requirements
- detailed design not in conformity to general design
- failure to conform to standards
- code not in conformity to detailed design
- and software not performing functions and producing outputs in conformity with requirement specifications.

Advantages of Traceability Analysis	**Disadvantages of Traceability Analysis**
• Highly effective for detecting errors during design and implementation phases • Valuable aid in verifying completeness, consistency, and testability of software • Aids in retesting software when a system requirement has been changed	• No significant disadvantages

Reviews

A review is a meeting at which the requirements, design, code, or other products of a software development project are presented to the user, sponsor, or other interested parties for comment and approval, often as a prerequisite for concluding a given phase of the software development process. Usually held at end of a phase, it may be called when problems arise. It is often referred to as "formal review" versus desktop review of materials (i.e., desk checking).

Advantages of Reviews	**Disadvantages of Reviews**
• Provides opportunity to change	• If participants do not have

course of a project before start of next phase • Because scope of review is usually broad, gives opportunity to recognize global problems	materials ahead of time and spend time preparing, review will accomplish little or nothing • Attention focus on major issues

Walk-throughs

A walk-through is an evaluation technique in which a designer or programmer leads one or more other members of the development team through a segment of design or code, while the other members ask questions and make comments about technique, style, and identify possible errors, violations of development standards, and other problems. Walk-throughs are similar to reviews, but are less formal. Other essential differences include the following: (1) participants are fellow programmers rather than representatives of other functions, (2) frequently no preparation is required, (3) standards are usually ignored, (4) checklists are rarely used, and (5) follow-up is often Ignored.

Types of errors detected include:

- interface
- logic
- data
- syntax.

Advantages of Walk-throughs	**Disadvantages of Walk-throughs**
• Less intimidating than formal reviews • Identifies the most error=prone sections of the program, so more attention can be paid to these sections during testing • Very effective in finding logic design and coding errors	• Designers or programmers may feel walk-through is an attack on their character or work

The Review Meeting

Success of a technical review requires that all participants carefully examine the inputs to the technical review prior to the review meeting. In both the management and technical reviews, experts on specific topics (e.g., design experts for design reviews) should be present to lend their knowledge to the review.

Review Reporting and Record-keeping

The review leader performs the administrative functions of the review and issues the technical review report. The recorder documents the findings, decisions, and recommendations of the review team. Review team members are expected to be prepared for the meeting and ensure that the review objectives are met. Recommendations made by the review team should be such that management can act on them quickly. Management is responsible for responding to recommendations promptly.

Review Guidelines

The review leader plans for the review by identifying the review team, schedules a time and place for the review, and distributes all inputs to the review team. An overview of the project is conducted for the review team by a qualified project member. Each review team member studies the software element and related materials. The technical review consists of the review team:

- Determining whether or not the software element corresponds to the specifications and standards to which it must adhere, and recording any deviations
- Listing issues, recommendations, and responsible individuals for resolving the issues
- Identifying other issues that need to be addressed
- Documenting the meeting, deficiencies found in the software element, and recommendations for management

Review Checklist

Formal reviews whether or not activities were performed in accordance with their respective plans. A checklist provides guidelines for reviewing both the software product and plans for assurance activities.

S*oftware Requirements Review*: (1) Do the interface requirements enable compatibility of external interfaces (hardware and software)? (2) Do the functional requirements cover all abnormal situations? (3) Are the specific models, algorithms, and numerical techniques compatible? (4) Are the requirements for the man-machine interface adequate? (5) Is there justification for the design/implementation constraints? (6) Will the design, operation, and maintenance of software be feasible? (7) Is each unique requirement defined more than once? Are there any redundant statements? (8) Are there requirements for fault tolerance and graceful degradation? (9) Are the requirements clear and specific enough to be the basis for detailed design specifications and functional test cases? (10) Are the requirements verifiable (i.e., can the software be checked to see whether requirements have been fulfilled)?

Software Design Review: (1) Are design features consistent with the requirements? (2) Does the design implement required program behaviour with respect to each program interface? (3) Are all required processing steps included? (4) Are all possible outcomes of each decision point designated? (5) Does the design take into account all expected situations and conditions? (6) Does the design configuration ensure integrity of changes? (7) Does each program have a single function? (8) Is the design structured so that it comprises relatively small, hierarchically related programs or sets of programs, each performing a particular, unique function? (9) Does the design avoid unnecessarily complex designs and design representations?

Source Code Review: (1) Is the code a complete and precise implementation of the design? (2) Was the code integrated and debugged to satisfy the design specification? (3) Does the code create the required database, including the initial data? (4) Are there any un-referenced or undefined variables, constants, or data types? (5) Is the code logically consistent with the design? (6) Does the code conform to specified standards? (7) Are all variables properly specified and used? (8) Are all comments accurate? (9) Are all programs invoked with the correct number of parameters? (10) Are cross-references or data dictionaries included to show variable and constant access by the program? (11) Does code consist of programs with only one entry point and one exit point? (12) Is the code written in a language with well-defined syntax and semantics? (13) Is the code free of unintended infinite loops? (14) Does the code avoid recursion? (15) Does the code protect against detectable run-time errors (e.g., range array index values, division by zero, out of range variable values, and stack overflow)? (16) Do loops only have one entrance? (17) Does the code contain or reference history of all code modifications and the reason for them? (18) Do the comment statements adequately describe each routine, using clear English language? (19) Was a nemonic naming convention used? Does the naming reflect the type of variable? (20) Is the valid range of each variable defined?

Test Readiness Review: (1) Have recommended changes been made to the code as result of source code review or, integration test? (2) Is the error rate sufficiently low to warrant beginning the next type of testing? (3) Are all test cases and procedures complete? (4) Is the test facility ready? Are schedules approved, personnel and physical requirements specified? (5) Have all test tools been checked? (6) Have all test procedures been checked?

Software Quality Metrics

Metric -vs- Measure

- A metric is the definition, algorithm or mathematical function used to make a quantitative assessment of product or process.
- A measure is the numerical value obtained by either direct or indirect measurement; may also be the input, output, or value of a metric.

Software quality **metrics** are applicable to all software products and include:

- completeness
- correctness
- reliability.

A measure of **completeness** is cause and effect graphing, which aids in identifying requirements that are incomplete and ambiguous. Also, it explores inputs and expected outputs of a program and identifies ambiguities. For test purposes, it is useful for evaluating test cases for probability of detecting faults. Some examples of characteristics related to completeness include the number of ambiguities identified and the number of ambiguities remaining.

Some examples of characteristics related to **correctness** include: (1) number of problem reports per system development life cycle phase, (2) number of reported problems per time period, (3) number of open real problems per time period, (4) number of closed real problems per time period, (5) number of unevaluated problem reports, (6) age of open real problem reports, (7) age of unevaluated problem reports, (8) age of real closed problem reports, and (9) rate of error discovery.

Required software **reliability** can be used to provide a reliability rating for a product through examination of the processes used to develop it. At the early planning phases of a project, reliability can be used to measure the trade-offs of cost and degree of reliability.

Measures of Software Quality

Two types of measures:

- Halstead’s software science .... measures the understandability of a software product
- McCabe’s complexity measure .... measures the testability of a software product

Halstead’s software science. Cyclomatic complexity may be used to determine the structural complexity of a code module. It is calculated in a manner similar to the static complexity of the design. The difference is that the cyclomatic complexity is calculated from a flowgraph of the module, with an edge added from the exit node to the entry node. Cyclomatic complexity is calculated by counting the number of regions in the graph.

The **Halstead software science** measure of observed program length can be used as a readability indicator. Readability metrics include: (1) number of grammatically incorrect comments, (2) number of misspellings, (3) total number of characters, (4) total number of comments, (5) number of comment characters, and (6) number of code characters. Halstead originally did not count declaration statements, input/output statements, or statement labels. However, later researchers do count the operands in these types of statements.

McCabe's complexity measure. Cyclomatic complexity can be used to measure the completeness of the testing that a programmer must satisfy. Specifically, branch coverage and path coverages are verified for completeness. McCabe's technique can be used to: (1) develop a set of test cases which test every outcome of each decision, and execute a minimal number of distinct paths, and (2) assess the testability of software base on its complexity.

Controls for Software Reliability, Integrity, Availability and Safety

1. **Software Reliability**

   The reliability of a computer system is defined as the probability that the system will be able to process work correctly and completely without its being terminated or corrupted. A system does not have to fail (crash) for it to be unreliable. The computer configuration, the individual components comprising the system, and the system's usage are all contributing factors to the overall system reliability. As the reliability of these elements varies, so will the level of system reliability.

   Reliability is a fundamental element to the security of computer systems. A failure can decrease or destroy the security of the system. Undesirable events such as denial of information, unauthorized disclosure of information, or loss of money and resources can result from lack of reliability.

   Reliability is also related to safety and quality. Reliability is critical to a system where there is a potential for loss of life, health, destruction of property, or damage to the environment. Examples include: health care systems, scheduling of safety inspections, manufacturing process control systems, and air traffic control systems. From a quality perspective, reliability deals with the assessment of impact of lack of reliability on the entire organization. Reliability is part of quality attributes.

2. **Software Availability**

   The availability of a computer system is a measure of the amount of time that the system is actually capable of accepting and performing a user's work. The terms reliability and availability are closely related and often used (although incorrectly) as synonyms. For example, a system that fails frequently, but is restarted quickly would have high availability, even though its reliability is low. To distinguish between the two, reliability can be thought of as the quality of service and availability as the quantity of service. In other words, availability can be viewed as a component of reliability. An example of availability is the availability of communication ports.

3. **Software Safety and Ergonomics**

Software Safety. Software safety is important, since lack of safety considerations in a computer-based application system can cause danger or injury to people, damage to equipment and property. It could also create financial or other loss to people using the system or people affected by the system. For example, an incorrectly programmed and incompletely tested medical diagnosis and treatment prescription system could kill a patient or injure people receiving the treatment. Another example: a process control system in a pharmaceutical company where drugs are incorrectly formulated by a computer system could kill or injure patients due to errors in software. Similarly, incorrect and obsolete documentation, especially after a program change was made, could lead to improperly functioning software, loss of life, failed missions, and lost time.

The overall purpose of the software safety evaluation review is to assess how well the product meets its software quality objectives. Quality objectives include reliability, safety, functionality, maintainability, and reviewability. To perform this evaluation, the reviewers need sufficient information about the product requirements, its development, and its overall quality. The following general types of questions can guide the reviewer regarding the product evaluation:

- How thoroughly has the developer or vendor analyzed the safety of critical functions of the software?
- How well has the developer or vendor established the appropriateness of the functions, algorithms, and knowledge on which the software is based?
- How carefully has the developer or vendor implemented the safety and performance requirements of the software?

Specific questions for the Requirements and Design Phases include: (1) Are the safety requirements consistent with the hazard analysis? (2) How was failure analysis of the software conducted? (3) Are there requirements for self-supervision of the software? (4) Is there modularity in the design? (5) Have critical components been isolated? (6) Were design reviews performed and documented?

Specific questions for the Implementation Phase include: (1) Do the test cases produce results identical with the expected output? (2) Does the operations procedure manual adequately describe diagnostic procedures and tools? (3) Does the operations procedure manual adequately describe emergency procedures and fault recovery procedures?

Specific questions for maintenance phase include: (1) Is there a formal procedure to start maintenance when problems are found? (2) Is there a formal change control procedure?

Ergonomics. Computer terminal screen design, chair and table height, and lighting conditions are part of ergonomics to make working with a computer easier. Health risks due to restricted range of body movements and the possibility of low frequency radiation such as eye stress, skin rashes, back pain, and visual disorders are involved in computer use.

TOOLS FOR PROJECT MANAGEMENT

Overview

Seven project management tools are discussed in this section, including program evaluation techniques, critical path methods, line-of-balance method, graphical evaluation and review techniques, work breakdown structure, charts and tables, and budgets. There is a time and place for each ones of these tools. The project manager needs to know when to use what tools to obtain the greatest benefit.

Project Management Tools

Program evaluation and review techniques (uses probabilities and three-time estimates, focus is on the time)
Critical path method (uses probabilities and a single-time estimate, focus is on cost)
Work breakdown structure (does not use probabilities, provides a conceptual organization of a project)
Charts and tables (do not use probabilities, focus is on presentation of data)
Budgets (use financial data, focus is on variance

analysis)

Program Evaluation and Review Techniques

Overview

Project management frequently uses network diagrams to plan the project, evaluate alternatives, and control large and complex projects toward completion. Program evaluation and review techniques (PERT) require extremely careful plans from the very outset of the project. This allows management to allocate resources to critical areas before they become critical. This will alert a manager to trouble areas or bottlenecks before they become a major problem and the source of a project overrun. PERT also helps to allocate resources, but has no influence on the excellence of the end product.

PERT improves communication upward to the manager and the customer (client). PERT lets the supervisor believe that the project manager is doing a superior job, regardless of how well the project manager is actually performing.

PERT Assumptions

Interrelationships of activities are depicted in a network of directed arcs (arcs with arrows, which denote the sequence of the activities they represent). The **nodes,** called events, represent instants in time when certain activities have been completed and others can then be started. All inwardly-directed activities at a node must be completed before any outwardly-directed activity of that node can be started. A **path** is defined as an unbroken chain of activities from the origin node to some other node. The origin node is the beginning of the project. An **event** is said to have occurred when all activities on all paths directed into the node representing that event have been completed.

Another assumption of PERT is that all activities are started as soon as possible. This assumption may not hold true when scarce resources must be allocated to individual activities.

PERT Applications

The development of a critical path network is accomplished by establishing the major milestones that must be reached. Construction of the network diagram requires identification and recording of the project's internal time dependencies – dependencies that might otherwise go unnoticed until a deadline slips by or impacts other activities. A new activity can be added by identifying its successor and predecessor.

An ordered sequence of events to be achieved would constitute a valid model of the program. The network provides a detailed, systematized plan and time schedule before the project begins. As the project progresses, the time estimates can be refined. A top-down approach is taken when developing the network. The total project is fully planned and all components of the plan are included.

Applications of PERT and CPM

- Construction and maintenance of chemical plant facilities, highways, dams, buildings, railroads, and irrigation systems
- Planning of retooling programs for high volume products in plants such as automotive and appliance plants
- Introduction of a new product
- Installation of a computer system
- Acquisition of a company

Critical path scheduling helps coordinate the timing of activities on paper and helps avert costly emergencies. The network diagram must be developed in detail as much as possible so that discrepancies, omissions, and work coordination problems can be resolved inexpensively, at least to the extent that they can be foreseen.

Project diagrams of large projects can be constructed by sections. Within each section the task is accomplished one arrow at a time by asking and answering the following questions for each job:

1. What immediately preceded this job?
2. What immediately succeeds (follows) this job?
3. What can be concurrent with this job?

If the maximum time available for a job equals its duration, the job is called "critical". A delay in a critical job will cause a comparable delay in the project completion time. A project contains at least one contiguous path of critical jobs through the project diagram from beginning to end. Such a path is called a "critical path".

Typically, only about 10 to 15 percent of the jobs in a large project are critical. The primary purpose of determining the "critical path" is to identify those activities that must be finished as scheduled if the new program or project is to be completed on time. The "critical path" of those activities cannot be delayed without jeopardizing the entire program or project.

If the maximum time available for a job exceeds its duration, the job is called a **floater**. Some floaters can be displaced in time or delayed to a certain extent without interfering with other jobs or the completion of the project. Others, if displaced, will start a chain reaction of displacements downstream in the project.

The technological ordering is impossible if a cycle error exists in the job data (i.e., job 'a' preceded 'b', 'b' precedes 'c', and 'c' precedes 'a'). The time required to traverse each arrow path is the sum of the times associated with all jobs on the path. The critical path (or paths) is the longest path in time from start to finish; it indicates the minimum time necessary to complete the entire project.

In order to accurately portray all predecessor relationships, "dummy jobs" must often be added to the project graph. The critical path is the bottleneck route, only by finding ways to shorten jobs along the critical path can the overall project time be reduced; the time required to perform non-critical jobs is irrelevant from the viewpoint of total project time.

The PERT Approach

The status of a project at any time is a function of several variables such as resources, performance, and time. Resources are in the form of dollars, or what "dollars" represent – manpower, materials, energy, and methods of production; technical performance of systems, subsystems, and components. An optimum schedule is one that would properly balance resources, performance, and time.

Information concerning the inherent difficulties and variability in the activity being estimated are reflected in the three numbers: the optimistic, pessimistic, and most likely elapsed time estimates should be obtained for each activity. The purpose of the analysis is to estimate, for each network event, the expected times (mean or average) and calendar time of occurrence (PTY).
When PERT is used on a project, the three time estimates (optimistic, most likely, and pessimistic) are combined to determine the expected duration and the variance for each activity.

Optimistic: An estimate of the minimum time an activity will take. This is based on everything "going right the first time". It can be obtained under unusual, good luck situations.

Most likely: An estimate of the normal time an activity will take, a result which would occur most often if the activity could be repeated a number of times under similar circumstances.

Pessimistic: An estimate of the maximum time an activity will take, a result which can occur, only if unusually bad luck is experienced.

The expected times determine the critical path, and the variances for the activities on this path are summed to obtain the duration variance for the project. A probability distribution for the project completion time can be constructed from this information. However, the variances of activities which do not lie on the critical path are not considered when developing the project variance, and this fact can lead to serious errors in the estimate of project duration.

An estimate of the length of an activity is an uncertain one. A stochastic model can be used to reflect this uncertainty. This model measures the possible variation in activity duration. This may take the form of a distribution showing the various probabilities that an activity will be completed in its various possible completion times. Alternatively, this may be non-distribution such as range or standard deviation.

The expected time = 1/6 (a + 4m + b)
where ‘a’ is optimistic time, ‘m’ is most likely time, and ‘b’ is pessimistic time. The expected activity times derived from a three estimate, PERT-type calculation provides a more accurate estimate and allows the activity time variance to be calculated and included in the estimates of project duration.

Example: A company is planning a multi-phase construction project. The time estimates for a particular phase of the project are:
Optimistic 2 months
Most likely 4 months
Pessimistic 9 months

Question: Using PERT, what is the expected completion time for this particular phase?
Answer: The expected completion time would be 4.5 months, as shown below:
The expected time = 1/6 (a + 4m + b) = 1/6 (2 + 4x4 + 9) = 27/6 = 4.5.

The latest calendar time at which an event must be accomplished so as not to cause a slippage in meeting a calendar time for accomplishing the objective event is referred to as the 'latest time', and denoted as TL. The difference between the latest and expected times, TL – PTY, is defined as **"slack"**. Slack can be taken as a measure of scheduling flexibility that is present in a work flow plan, and the slack for an event also represents the time interval in which it might be reasonably be scheduled. Slack exists in a system as a consequence of multiple path junctures that arise when two or more activities contribute to a third.

- A slack time is a free time associated with each activity as it represents unused resources that can be diverted to the critical path.
- Non-critical paths have slack time while critical paths have no such slack time.

A slack is extra time available for all events and activities not on the critical path. A negative slack condition can prevail when a calculated end date does not achieve a program date objective established earlier.

The manager must determine valid means of shortening lead times along the critical path by applying new resources or additional funds that are obtained from those activities that can "afford" it because of their slack condition. "Safety factor" is another name for "slack". Alternatively, the manager can re-evaluate the sequencing of activities along the critical path. If necessary, those activities which were formerly connected in a series can be organized on a parallel or concurrent basis, with the associated trade-off risks involved. Alternatively, the manager may choose to change the scope of work of a critical path alternative in order to achieve a given schedule objective.

When some events have **zero slack**, it is an indication that the expected and latest times for these events are identical. If the zero-slack events are joined together, they will form a path that will extend from the present to the final event. This path can be looked upon as "the critical path". Should any event on the critical path slip beyond its expected date of accomplishment, then the final event can be expected to slip a similar amount. The paths having the greatest slack can be examined for possible performance or resource trade-offs.

When jobs or operations follow one after another, there is no slack. Sub-critical events, where the criteria for defining a sub-critical event relates to the amount of slack involved in the event. Those events having as much as five weeks slack have been deemed sub-critical.

The PERT analysis permits a quantitative evaluation of conceivable alternatives. Each job in the project is represented by an arrow that depicts: (1) the existence of the job, and (2) the direction of time-flows from the tail to the head of the arrow. The arrows are then connected to show graphically the sequence in which the jobs in the project must be performed. The junctions where arrows meet are called events. These are points in time when certain jobs are completed and others must begin.

The difference between a job's early start and its late start (or between early finish and late finish) is called total slack (TS). Total slack represents the maximum amount of time a job may be delayed beyond its early start without necessarily delaying the project's completion time.

Key Concepts to Remember – PERT Time Dimensions
ES = Earliest start time for a particular activity
EF = Earliest finish time for a particular activity
EF = ES + t, where 't' is expected activity time for the activity
LS = Latest start time for a particular activity
LF = Latest finish time for a particular activity
LS = LF – t, where 't' is expected activity time for the activity
Slack = LS – ES = LF – EF

The manager examines the work demand and indicates if sufficient resources are available to accomplish all jobs by their early finish. If resources are insufficient, activities are rescheduled within their late finish, using project priority, and available slack. Later, the manager is asked for additional resources or for a decision to delay an activity beyond its late finish.

Critical jobs are those on the longest path throughout the project. That is, critical jobs directly affect the total project time. If the target date (T) equals the early finish date for the whole project (F), then all critical jobs will have zero total slack. There will be at least one path going from start to finish that includes critical jobs only, i.e., the critical path. There could be two or more critical paths in the network, but only one at a time.

If T is greater (later) than F, then the critical jobs will have total slack equal to T minus F. This is a minimum value; since the critical path includes only critical jobs, it included those with the smallest TS. All non-critical jobs will have greater total slack.

Another kind of slack is **free slack** (FS). It is the amount a job can be delayed without delaying the early start of any other job. A job with positive total slack may or may not also have free slack, but the latter never exceeds the former.

For purposes of computation, the free slack of a job is defined as the difference between the job's EF time and the earliest of the ES times of all its immediate successors.

When a job has zero total slack, its scheduled start time is automatically fixed (i.e., ES + LS); and to delay the calculated start time is to delay the whole project. Jobs with positive total slack, however, allow the scheduler some discretion in establishing their start times. This flexibility can be applied to smoothing work schedules.

Peak load may be relieved by shifting jobs on the peak days to their late starts. Slack allows this kind of juggling without affecting project time.
<u>Possible data errors in PERT:</u>

- The estimated job time may be in error
- The predecessor relationship may contain cycle errors (job 'a' is a predecessor for 'b', 'b' is a predecessor for 'c', and 'c' is a predecessor for 'a')
- The list of prerequisites for a job may include more than the immediate prerequisites; (e.g., job 'a' is a predecessor of 'b', 'b' is a predecessor of 'c', and 'a' and 'b' both are predecessors of 'c')
- Some predecessor relationships may be overlooked
- Some predecessor relationships may be listed that are spurious
- The errors in the PERT calculated project's mean and standard deviation will tend to be large if many non-critical paths each have a duration approximately equal to the duration of the critical path. However, the more slack time there is in each of the non-critical paths, the smaller will be the error.

One way to minimize errors and omissions is to continually back-check the data and challenge the assumptions. The following table presents advantages and disadvantages of PERT:

Advantages of PERT	**Limitations of PERT**
• Greatly improved control over complex development work and production programs • Capacity to distill large amounts of data in brief, orderly fashion • Requires a great deal of planning to create a valid network • Represents the advent of the management-by-exception principle • People in different locations can relate their efforts to the total	• There is little interconnection between the different activities pursued • Requires constant updating and reanalysis of schedules and activities • Requires greater amount of detail work • Does not contain the quantity information, only time information is available

task requirements of a large program • "Downstream" savings are achieved by earlier and more positive action on the part of management in the early stages of the project	

PERT Implementation Issues

- The people and organization of a project are more important considerations than the use of a particular planning and control technique
- Consideration should be given to managerial issues such as project organization, personalities of project members, and operating schemes
- There is a big difference between the criteria of success for the task to be accomplished and the criteria of success for the management system
- The project manager is a miniature general manager. He usually lacks the commensurate authority and depends on various management techniques to carry out his job
- The project management approach is the preferred method to deal with one-time defined projects
- The qualifications of a person making time estimates must include a thorough understanding of the work to be done
- Precise knowledge of the task sequencing is required or planned in the performance of activities

PERT/Cost

Once the network has been established, based upon the project work breakdown structure, costs can be estimated. If the breakdown has been made satisfactorily, it will serve as both an estimating and actual cost accumulation vehicle. PERT/cost adds the consideration of resource costs to the schedule produced by the PERT procedure. The basic PERT handles the problem of time uncertainty while PERT/cost addresses cost uncertainty. Cost uncertainty as it relates to time can be handled by different cost estimates for three-time differences. The ultimate objective is not only to improve planning and control, but also to assess possibilities for ''trading off" time and cost, i.e., adding or subtracting from one at the expense of the other.

There is an "optimum" time-cost point for any activity or job as indicated by the "U" shape of the curve drawn between total direct cost (on y-axis) versus time (on x-axis). It is assumed that total costs will increase with any effort to accelerate or delay the job away from this point in the case where resource application varies. Crashing the project involves shortening the critical path or paths by operating on those activities which have the lowest time-cost slopes. At least three approaches are available to develop the cost estimates:

1. A single cost estimate of expected cost
2. Three cost estimates
3. Optimum time-cost curves

A **single cost estimate** of expected cost is based upon the assumption of the individual cost elements. The **three-cost estimate** approach determines the "expected cost". The advantage of the three-cost estimate over the single cost estimate is that the result is subject to probability analysis. With this expected cost, the manager cannot assume that he has the optimum time-cost mix.

The third approach to estimation is the **optimum time-cost curve concept**. This is differential costing with time as the variability factor. The intention of this approach is to optimize time and costs by using optimum estimated costs. It assumes there is a direct relationship between time and costs on any activity. This relationship can be expressed by a continuous curve. The method is also based upon the concept that activities are subject to time-cost trade-offs. The optimum time-cost curve method is difficult to put into practice due too the need to develop continuous time-cost curves.

Critical Path Method

Overview

The critical path method (CPM) is a powerful, but basically simple technique for analyzing, planning, and scheduling large, complex projects. In essence, the tool provides a means for determining (1) which jobs or activities, of the many that comprise a project, are "critical" in their effect on total project time, and (2) how best to schedule all jobs in the project in order to meet a target date at minimum cost. CPM is an extension of PERT>
<u>Characteristics of project for analysis by CPM:</u>

- The project consists of a well-defined collection of jobs or activities which, when completed, mark the end of the project
- The jobs may be started and stopped independently of each other, within a given sequence
- The jobs are ordered in a technological sequence (for example, the foundation of a house must be constructed before the walls are erected)

CPM focuses attention on those jobs that are critical to the project time, it provides an easy way to determine the effects of shortening various jobs in the project, and it enables the project manager to evaluate the costs of a "crash" program.

Time estimates for both normal and crash options are used in the CPM method. Crash time is the time required by the path if maximum effort and resources are diverted to the task along this path. A balance can be obtained when a project manager knows what the normal time and the crash time would be.

As needed, it is a costly practice to "crash" all jobs in a project in order to reduce total project time. If some way is found to shorten one or more of the critical jobs, then no only will the whole project time be shortened but the critical path itself may shift and some previously non-critical jobs may become critical. It is physically possible to shorten the time required by critical jobs by assigning more people to the jobs; working overtime; using different equipment, materials, and technology.

When CPM is used in a project to develop a crashing strategy, two or more paths through the network may have nearly the same length. If the activity duration is allowed to vary, a decrease in the length of the critical path may not result in an equivalent decrease in the project duration, because of the variance inherent in the parallel or alternate paths. These variations of activity times can even allow the alternate path to become a critical path. Thus, simply allowing the activity times to vary slightly from their estimates in order to make the length of the paths different can cause serious errors in a CPM crashing strategy and lead to wasted resources and cost overruns.

Characteristics of CPM Networks

- CPM networks attempt to build the entire project "on paper" at a very early stage of the project – even when the scope is not defined, vaguely defined, or incorrectly defined. In a way, CPM is to the project management what modelling or simulation is to economic studies, production problems, plant design, and transportation problems.
- CPM provides a graphic view of the entire project with completion dates, support activities, and costs affixed to every stage of the project.

The critical path techniques are as valuable on short- and middle-range planning jobs as they are on major and extremely complex projects.

- CPM's single time estimate fails to consider the effects of variability in path completion times on the crashing strategy.
- The CPM chart is an excellent tool for communicating scope as well as details of the job to other persons directly and indirectly concerned with the development and completion of its various phases.

- The CPM chart serves as a permanent record and reminder of the substance of this communication to all management levels.
- The CPM chart shows the timing of management decisions.
- CPM enables the manager to measure progress (or lack of it) against plans and to take appropriate action quickly when needed. And the underlying simplicity of CPM and its ability to focus attention on crucial problem areas of large projects makes it an ideal tool for the senior manager.

CPM versus PERT

CPM and PERT methods are essentially similar in general approach and have much in common. However, important differences in implementation details exist. They are independently derived and based on different concepts. Both techniques define the duration of a project and the relationships among the project's component activities. An important feature of the PERT approach was its statistical treatment of the uncertainty in activity time estimates, which involves the collection of three separate time estimates and the calculation of probability estimates of meeting specified schedule dates.

CPM differs from PERT in two areas:

1. The use of only one time estimate for each activity (and thus no statistical treatment of uncertainty)
2. The inclusion, as an integral part of the overall scheme, of a procedure for time/cost tradeoffs to minimize the sum of direct and indirect project costs

Common features of PERT and CPM

1. They both use a network diagram for project representation, in which diagram circles represent activities, with arrows indicating precedence
2. They both calculate early and late start and finish times and slack time

The following table provides a comparison between CPM and PERT:

CPM	**PERT**
1. Uses a single deterministic time estimate to emphasize minimum project costs while downgrading consideration of time restraints 2. 2. It is the choice of cost-conscious managers	1. Uses three time estimates to define a probabilistic distribution of activity times which emphasizes minimum project duration while downgrading consideration of cost restraints 2. It tends to be used by time-conscious managers

While these two techniques are based on different assumptions, they are related to each other because of the obvious relationship between time and cost. ***The 'ideal' work technique would combine the concepts of CPM's crashing strategy with PERT's probability distribution of activity times to derive the optimum project duration and cost.***

- **Work Breakdown Structure**

  The work breakdown structure (WBS) was first intended as the common link between schedules and costs in the PERT cost application. Later, it became an important tool for conceptual organization of any project. The WBS provides the necessary logic and formalization of task statements. The WBS prepares the work packages, which usually represent the lowest division of the end items.

  WBS allows a project manager to decompose project tasks and activities into a manageable size to facilitate project planning and control. WBS is a hierarchical definition of tasks to be performed and accounted for in a project. Major task areas are defined, and each major area is further identified by defining subtasks. Each subtask may be further broken down so that the lowest level of task definition for each item becomes the basic unit for costing and scheduling. The elements of the WBS must be related to people; someone must always be held responsible and accountable for each element.

  The WBS is a checklist of every activity that must be performed to develop an end product. This checklist becomes the foundation for resource allocation, time schedules, and budget plans.

  The WBS provides the basis for project organization, cost estimation, task scheduling, and cost reporting. Its goal is to identify stand-alone activities that can be scheduled and costed. The task-segmentation process provides additional understanding of the tasks and increased costing accuracy. The WBS should also correlate with the lowest-level organizational structure for the project. This provides a traceability of resource requirements associated with each task and results in allocation of estimated resource requirements within the organization.

  There is a definite interrelationship between the WBS and project schedules. Each WBS element has a definite time-phasing relative to other WBS elements. The scheduling of activities is an iterative process that provides the framework for cost estimating and the basis for the overall project plan.

- **Charts and Tables**

  Overview

  The basic purpose of a chart or graph is to give a visual comparison between two or more things. For example, changes in budget from one year to the next may be represented in a graph. One significant reason for visualizing a comparison is to reinforce its comprehension.

  Charts and graphs are used to dramatize a statement, a fact, a point of view, or an idea. Visual aids assist in the quick comprehension of both simple and complex data, statistics, or problems. A chart should explain itself in silence; it should be completely understood without the assistance of a caption. The caption must act only as a reinforcement to its comprehension.

  Various charts such as tabular charts, column charts, bar charts, milestone charts, pie charts, line charts, layer charts, and input/output charts along with decision trees and decision tables are discussed briefly.

Types and Charts and Tables
○ Tabular chart (used to represent items of interest) ○ Column chart (used for comparison of things) ○ Gantt (bar) chart (used to show duration of actions) ○ Milestone chart (used to show achievements) ○ Pie chart (used to represent a 100 percent of total) ○ Line chart (used for comparison of things) ○ Layer chart (used for accumulation of individual facts) ○ Input/output charts (actual results are compared with forecasts) ○ Decision trees (display the sequential nature of decision making) ○ Decision tables (documents conditions, rules, and actions)

The **tabular chart** is used to represent items of interest. It requires a fair amount of study in order to grasp the full meaning of the figures. This is because it takes longer to digest the meaning of an itemization of compiled figures than if the same figures are presented graphically. The **column chart** is most commonly used for demonstrating a comparison between two or more things. The column chart is vertical.

The **Gantt chart** is a bar chart and is essentially a column chart on its side, and is used for the same purpose. The bar chart is horizontal. The bar chart is a tool which allows a manager to evaluate whether existing resources can handle work demand or whether activities should be postponed. The Gantt chart is used to show task scheduling where each task has start and completion dates.

Gantt chart is a graphical illustration of a scheduling technique. The structure of the chart shows output plotted against units of time. It does not include cost information. It highlights activities over the life of a project and contrasts actual times with projected times using a horizontal (bar) chart. It gives a quick picture of a project's progress in knowing the status of actual time lines and projected time lines. The following table presents advantages and disadvantages of PERT and Gantt charts.

The following table presents the advantages and disadvantages of PERT and Gantt charts.

Type	**PERT**	**Gantt Chart**
Advantages	○ A good planning aid. ○ Interdependencies between activities can be shown. ○ Network diagram is flexible to change. ○ Activity times are probabilistic. ○ A good scheduling tool for large, non-routine projects. ○ A good tool in predicting resource needs, problem areas, and the impact of delays on project completion.	○ A good planning tool. ○ A graphical scheduling technique, simple to develop, use and understand. ○ Useful for large projects. ○ Shows a sequence of steps or tasks. ○ Actual completion times can be compared with planned times.
Type	**PERT**	**Gantt Chart**
Disadvantages	○ Difficult to apply to repetitive assembly	○ Interrelation ships among

	line operations where scheduling is dependent on the pace of machines. o Large and complex projects are difficult to draw manually. o Requires computer hardware and software to draw a complex network. o Requires training to use the computer program.	activities are not shown on the chart. o Inflexible to change. o Activity times are deterministic. o Difficult to show very complex situations. o Cannot be used as a procedure-documenting tool. o Does not show the critical path in a chain of activities.

A **milestone chart** represents a major activity or task to be accomplished (e.g., a design phase in a computer system development project).

**** Gantt Chart Versus Milestone Chart ****

- Gantt charts show the duration of tasks.
- Milestone charts show achievements.
- Both Gantt charts and milestone charts show activities against time.

The **pie chart** is used to represent a 100 percent total of two or more items. The **line chart** is exceptionally impressive when comparing several things, but could present a visual problem if the comparisons are too many or too close in relation to one another. Advantages are that it is simple to draw. Disadvantages are that, if the lines are close to each other, it is difficult to distinguish some of the plotted points.

The **layer chart** is linear in appearance but has a different representation. It depicts the accumulation of individual facts stacked one over the other to create the overall total. This chart is more complex than the others, since it illustrates much more. In addition to showing the comparison of layers that add up to the total, this type of chart also shows how each group of layers relates to subsequent groups. The layer chart requires more work to prepare than the other charts. There is more arithmetic involved, and it requires a good deal of concentration to draw the chart.

Input/output charts reflect an actual expenditure (inputs) and accomplishment (outputs) versus a plotted forecast. Each area of a project is assigned a task, and funding and accomplishments for each task are forecasted. Actual results are compared with forecasts.

Decision trees are a graphical representation of possible decisions, events or states of nature resulting from each decision with its associated probabilities, and the outcomes of the events or states of nature. The decision problem displays the sequential nature of the decision-making situation. The decision tree has nodes, branches, and circles to represent junction boxes, connectors between the nodes, and state-of-nature nodes, respectively.

Decision tables (truth tables) are a tool which documents rules used to select one or more actions based on one or more conditions. These conditions and their corresponding actions can be presented either in a matrix or tabular form.

Budgets

Budgets have been a management tool for a long time. Budgets are the financial expression of project tasks. Either incremental budget or zero-based budgeting concepts can be practiced for managing a project. Incremental budgeting starts with data from a similar, previous project and a factor is built-in to increase or decrease the budget for the new project. Zero-based budgeting starts fresh and does not depend on a previous project's data. Variance analysis highlights the differences between actual and budget. Both positive and negative variance needs to be analyzed to understand the root cause of the difference.

INFORMATION SYSTEMS AUDIT TECHNIQUES

PROJECT MANAGEMENT REVIEW

The major objective of the project management process, a part of software assurance process, is to establish the organizational structure of the project and assign responsibilities. The process uses the system requirements documentation and information about the purpose of the software, criticality of the software, required deliverables, and available time and other resources, to plan and manage the software development and maintenance processes. The project management process begins before software development starts and ends when its objectives have been met. The project management process overlaps and often reiterates other software assurance processes. It establishes/approves standards, implements monitoring and reporting practices, develops high-level policy for quality, and cites laws and regulations for compliance.

Review the following activities performed by the project manager in project planning area: (1) set objectives or goals – determine the desired outcome for the project: (a) analyze and document the system and software requirements; define the relationships between the system and software activities, (b) determine management requirements and constraints (resource and schedule limitations), and (c) define success criteria; always include delivery of software that satisfies the requirements, on time and within budget, (2) plan for corrective action, (3) develop project strategies – decide on major organizational goals (e.g., quality) and develop a general program of action for reaching those goals, (4) develop policies for the project – make standing decision on important recurring matters to provide a guide for decision (end of page 830).