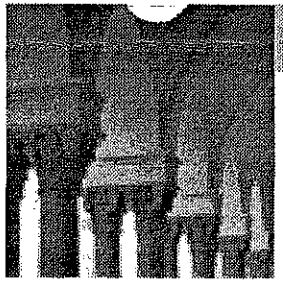
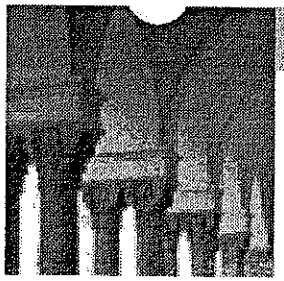


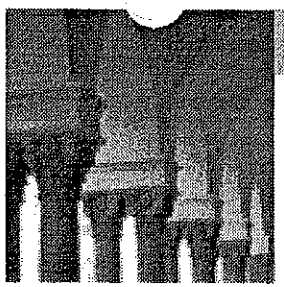
Objectives

- Understand the differences between the Software Development Lifecycle (SDLC) models
- Explain the detailed activities that take place in each project phases
- Understand the project manager's role throughout the SDLC

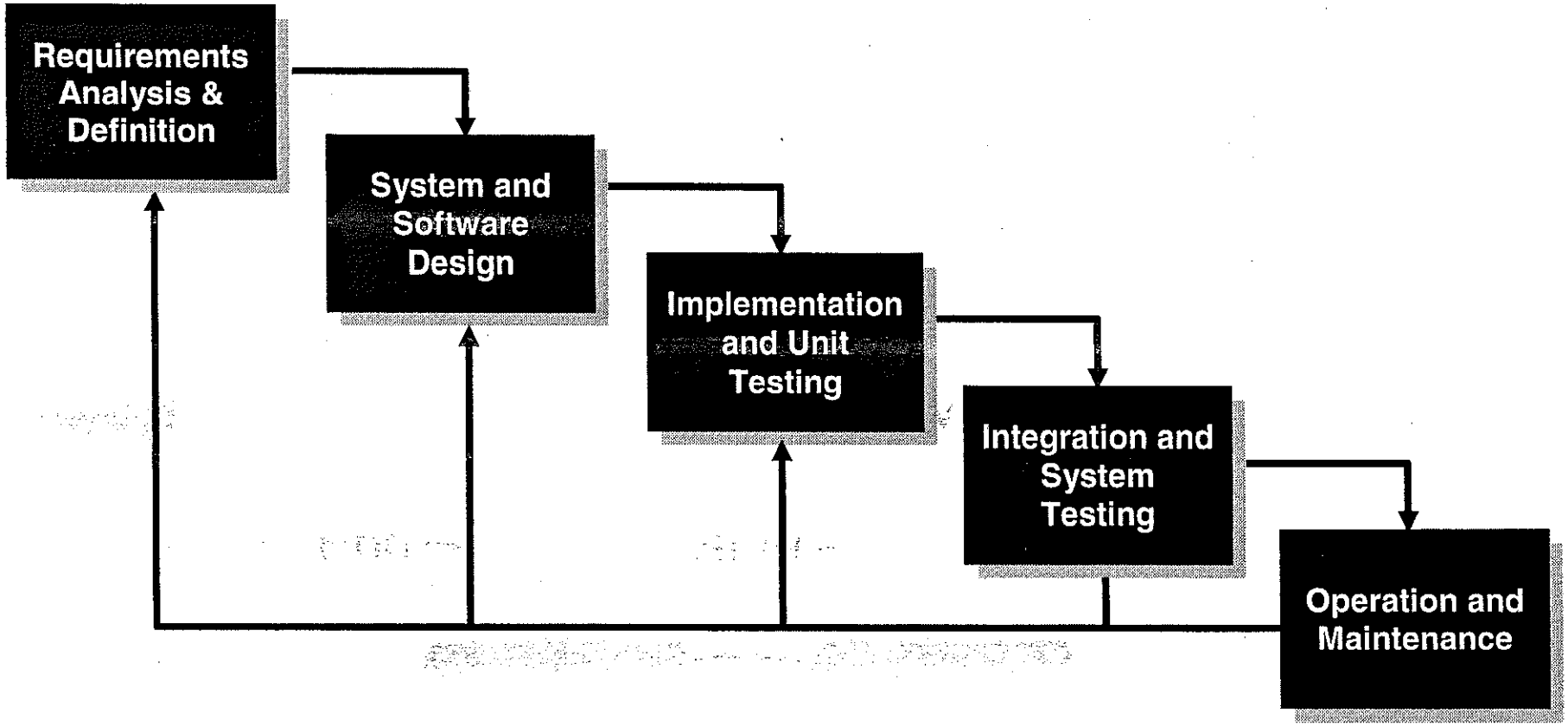


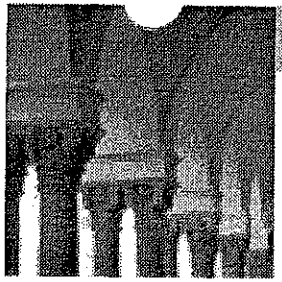
Software Development Life Cycles

- Waterfall
- Prototype
- Incremental
- Spiral
- Other variations

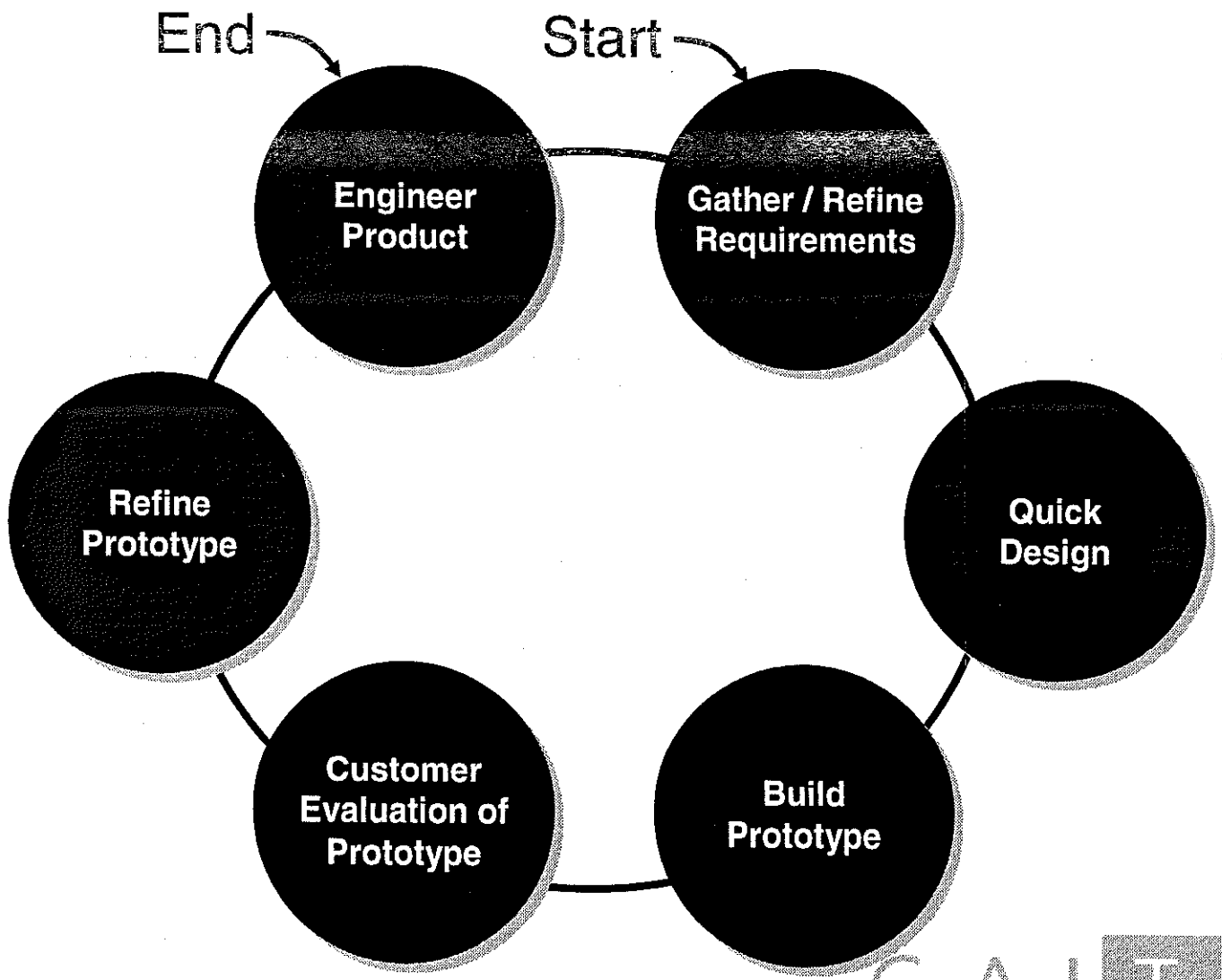


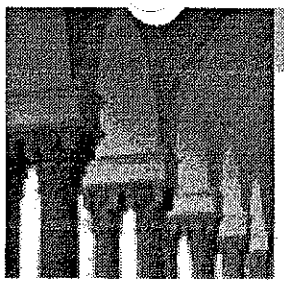
Waterfall


Requirements Analysis & Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

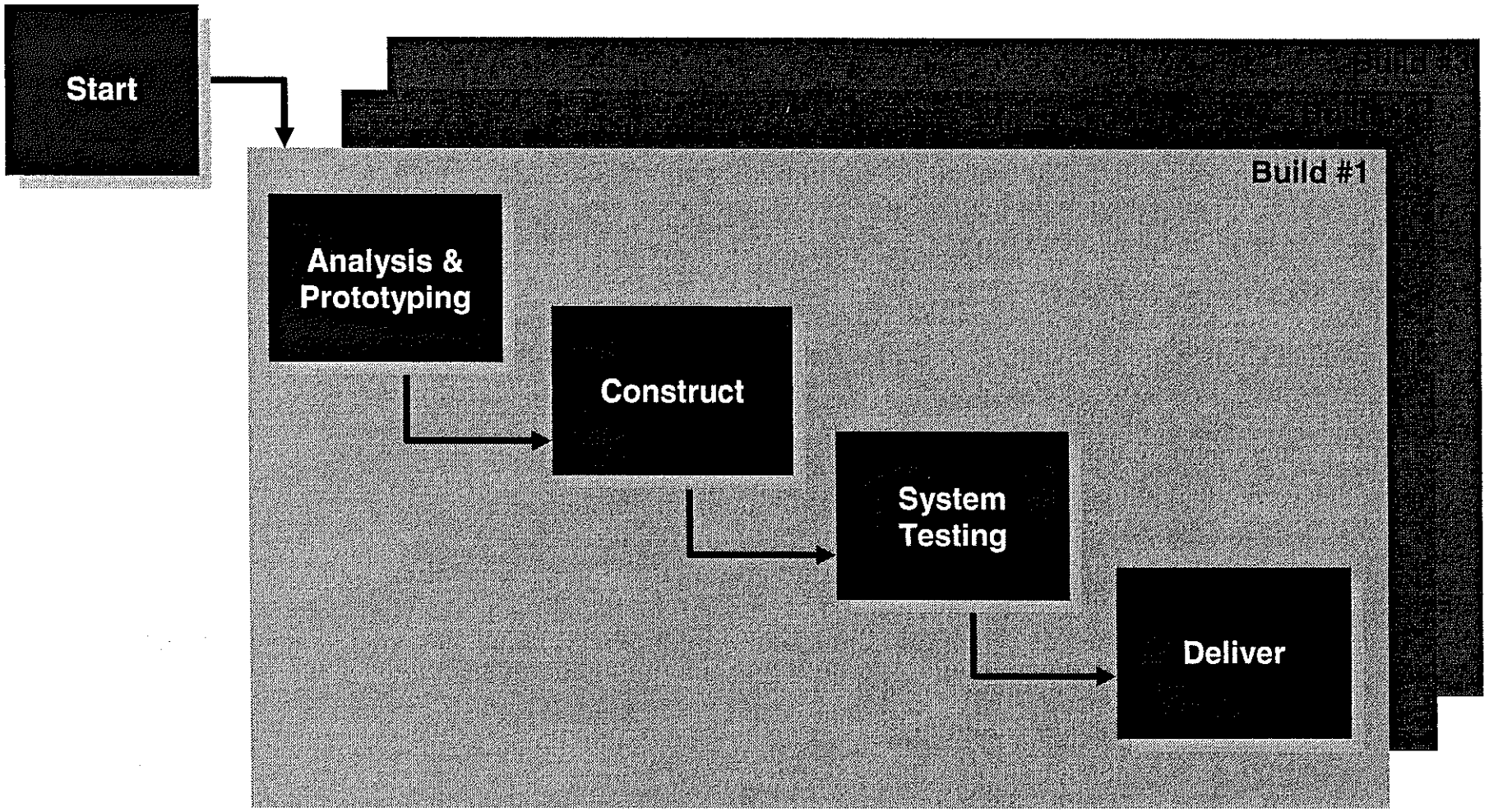


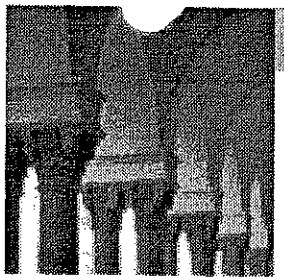
Prototype


End
Start
Engineer Product
Gather / Refine Requirements
Refine Prototype
Quick Design
Customer Evaluation of Prototype
Build Prototype

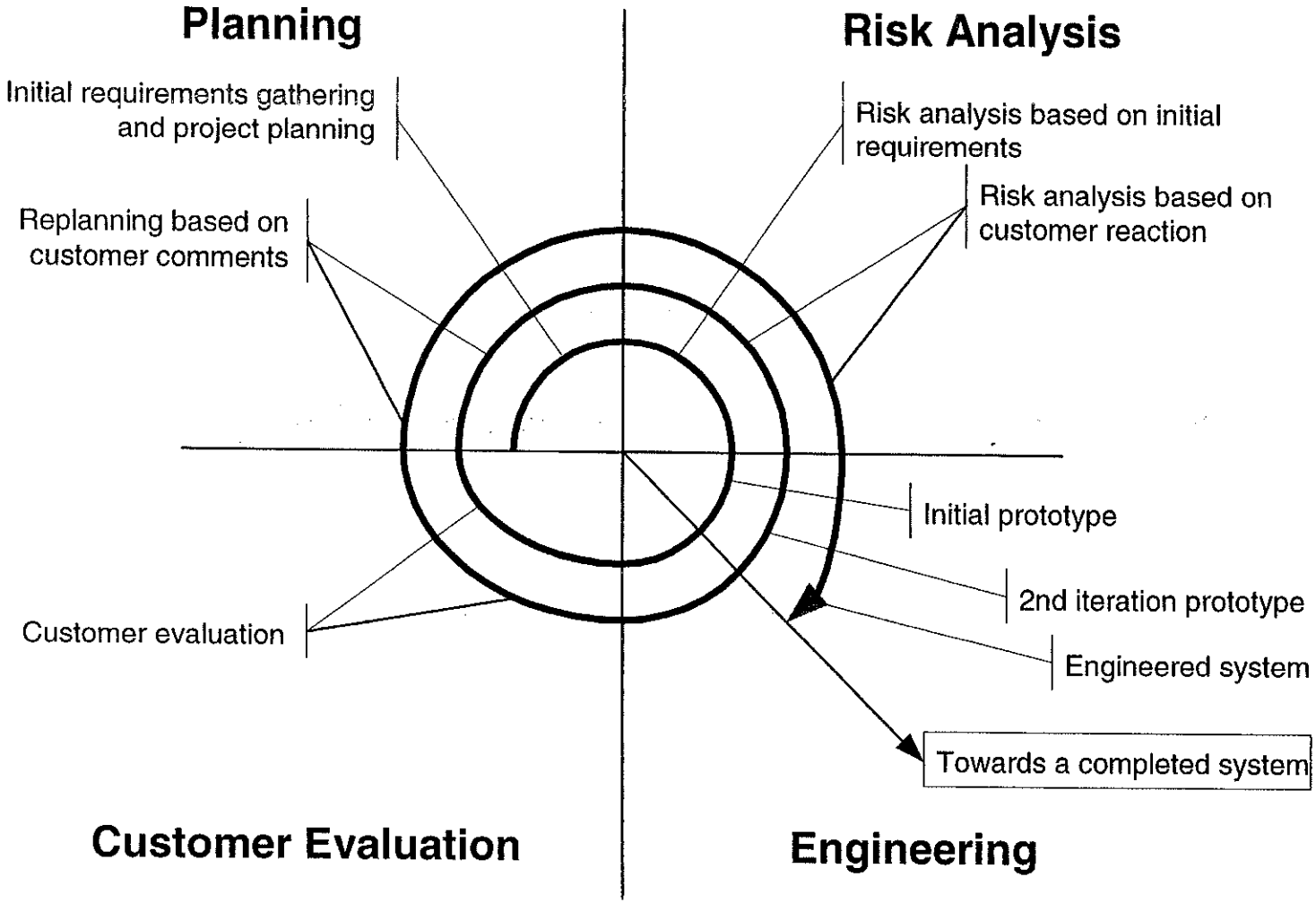


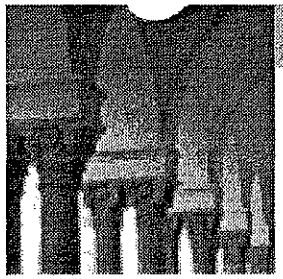
Incremental





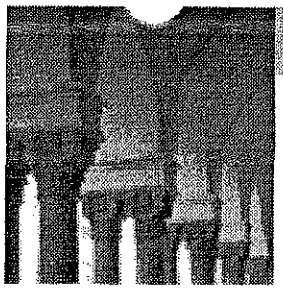
Spiral


Planning
Initial requirements gathering and project planning
Replanning based on customer comments
Risk Analysis
Risk analysis based on initial requirements
Risk analysis based on customer reaction
Initial prototype
2nd iteration prototype
Engineered system
Towards a completed system
Customer evaluation
Customer Evaluation
Engineering



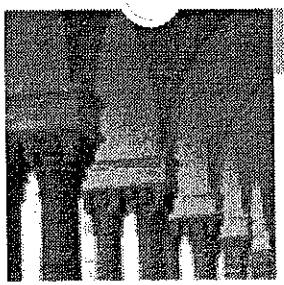
SDLC Selection Matrix

Criteria	Waterfall	Prototype	Incremental	Spiral
Availability of resources	All	Some	Some	Some
Complexity of project	Low	Medium	High	High
Understanding of user requirements	Specific	Vague	Vague	Vague
Product technology	Existing	New	New	New
Requirements volatility	Low	High	Medium	High
Risk management perspective	No	Yes	No	Yes
Schedule constraint	Medium	Low	Medium	Medium
Problem domain knowledge	High	Fair	Poor	Poor



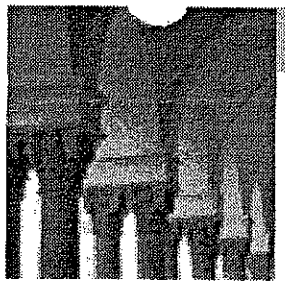
Availability of Resources

- Time, people, money and tools
  - **All** - Resources can be identified and are available
  - **Some** - Resources cannot be estimated or the resources are not all available



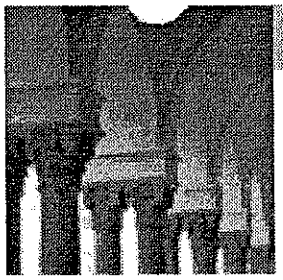
Complexity of Project

- The number of subsystems, the number of disciplines and the novelty of technology
  - **Low** - All of the criteria for complexity are low
  - **Medium** - Criteria for complexity are mixed; some are high and others are low
  - **High** - All of the criteria of complexity are high



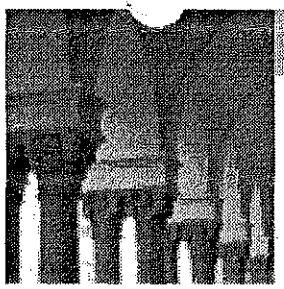
Understanding of User Requirements

- Understanding user requirements is critical to the project
  - **Vague** - Requirements are not well-defined (users may not be able to communicate their requirements, causing developers to make assumptions)
  - **Specific** - Requirements are well-defined, measurable and testable



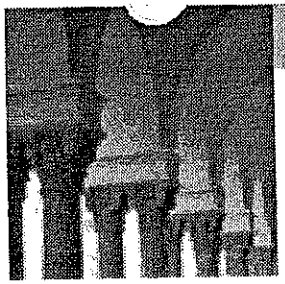
Product Technology

- If the technology is new, the life cycle selection is impacted
- Current or familiar technology allows the developers to function in a well-known and understood environment
  - **Existing** - Up-to-date technology that has been previously used by the developers
  - **New** - Technology that has not been applied by the project team



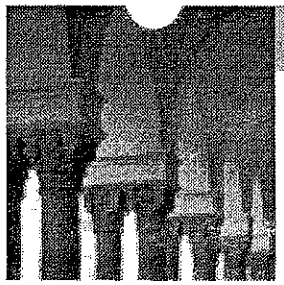
Requirements Volatility

- Degree to which requirements are expected to change during the development lifecycle
  - **Low** - Requirements are well-defined and exist in a stable system
  - **Medium** - Requirements are subject to minimal change
  - **High** - Requirements are highly volatile, the project team is chasing a moving target



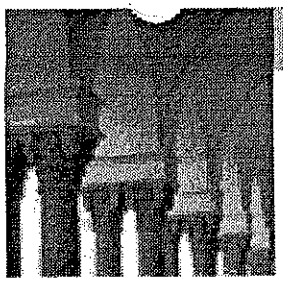
Risk Management Perspective

- Some life cycle models deal with risk as a factor, while others do not
  - **No** - Risk management is not a consideration for life cycle selection
  - **Yes** - Risk management must be included in the life cycle



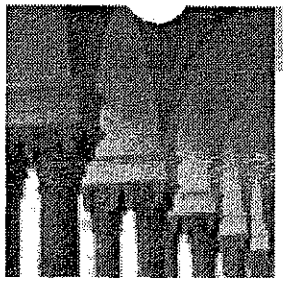
Schedule Constraint

- Project with a drop-dead date or high priority on schedule
  - **Low** - Little or no schedule constraint
  - **Medium** - Moderate schedule constraint that will result in loss of revenue if not met
  - **High** - Extreme schedule constraint that will result in major revenue loss if not met



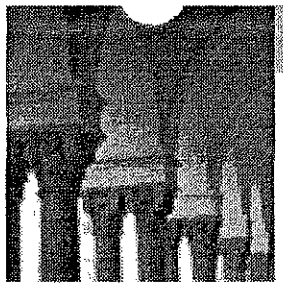
Problem Domain Knowledge

- How well the team understands the problem domain represented by the application
  - **Poor** - Little or no familiarity
  - **Fair** - Average familiarity
  - **High** - Strong understanding



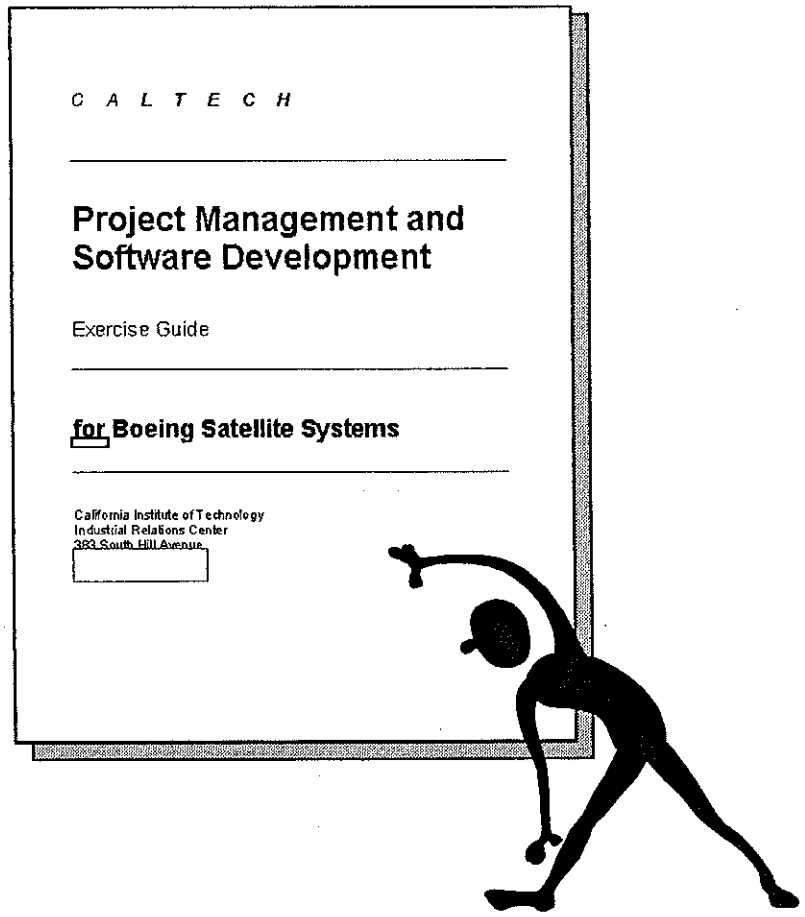
SDLC Selection Matrix

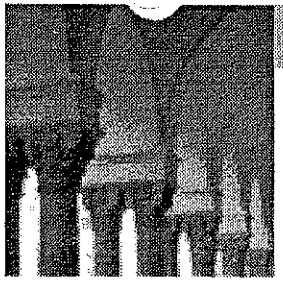
Criteria	Waterfall	Prototype	Incremental	Spiral
Availability of resources	All	Some	Some	Some
Complexity of project	Low	Medium	High	High
Understanding of user requirements	Specific	Vague	Vague	Vague
Product technology	Existing	New	New	New
Requirements volatility	Low	High	Medium	High
Risk management perspective	No	Yes	No	Yes
Schedule constraint	Medium	Low	Medium	Medium
Problem domain knowledge	High	Fair	Poor	Poor



Exercise

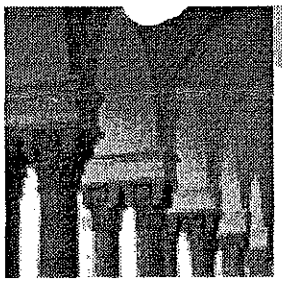
- 5.1 – Selecting a Software Development Model





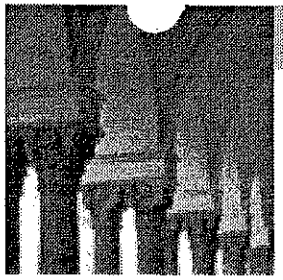
Rational Unified Process

- Software engineering process framework developed and marketed by Rational Software
- RUP is a software development approach that is:
  - iterative
  - architecture-centric
  - use-case driven



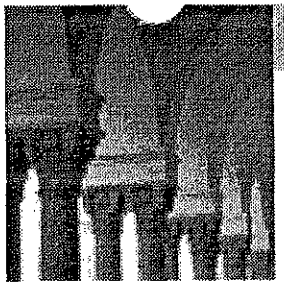
Principles of RUP Approach

- Attack major risks early and continuously
- Deliver Value to your customer
- Stay focused on executable Software
- Accommodate change early
- Baseline and executable architecture early
- Build the system with components
- Work together as one team
- Make quality a way of life



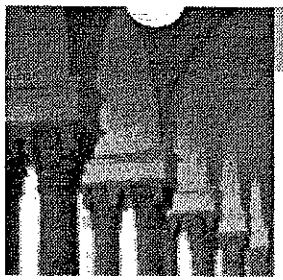
Best Practices

- Express project's plans in the areas of scope, time, cost, quality, process
- What are the risks if the project does not follow these plans
- Monitor progress using objective metrics
- Revise plans if the project goes off course
- Learn from your mistakes so they are not repeated in the next iteration



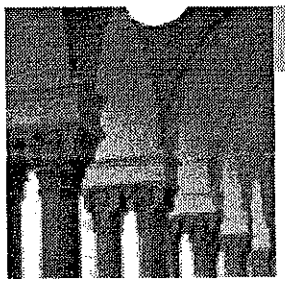
Activities of the PM

- Launch a new project
- Develop the SDP
- Starting and Closing phases and iteration
- Monitoring the project



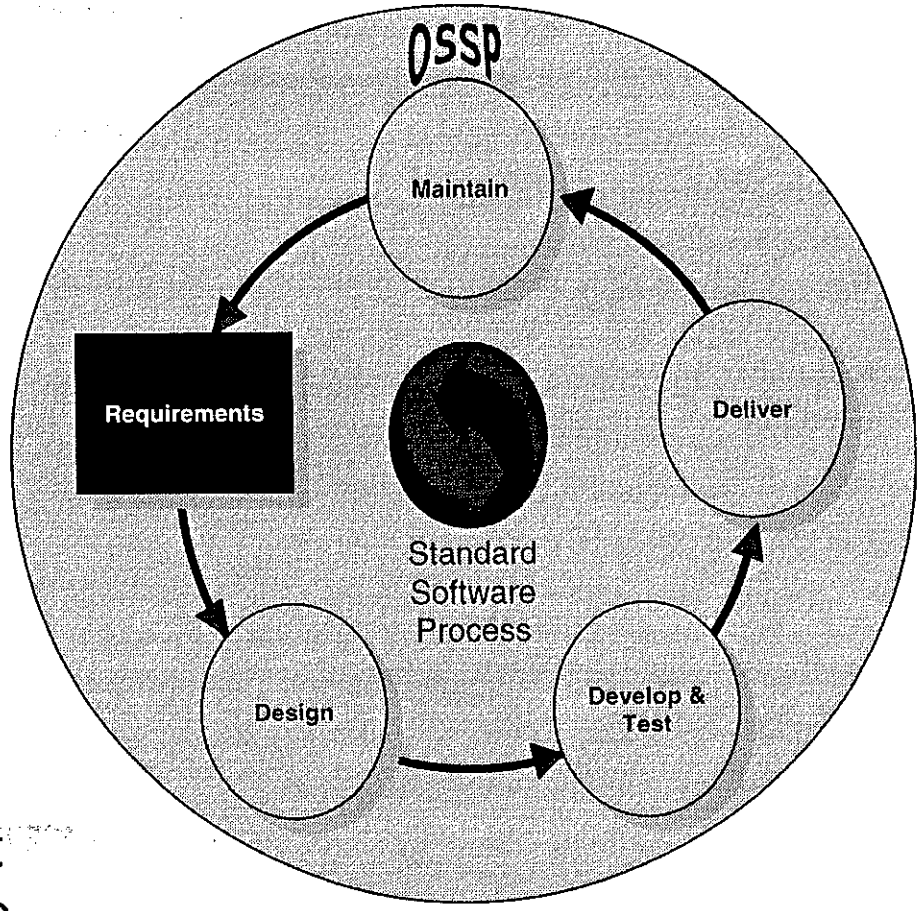
Project Management and RUP

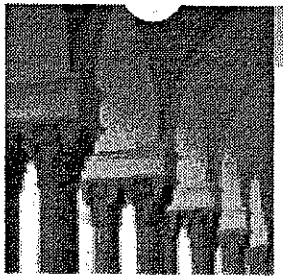
- Stay focused on results
- People, Product, process, project
- Role of PM requires many different skills:
  - Technical skills – a good level of understanding of the technical issues is necessary to achieve the best results
  - Communication Skills – personal communication, status reports, customer reviews and audits



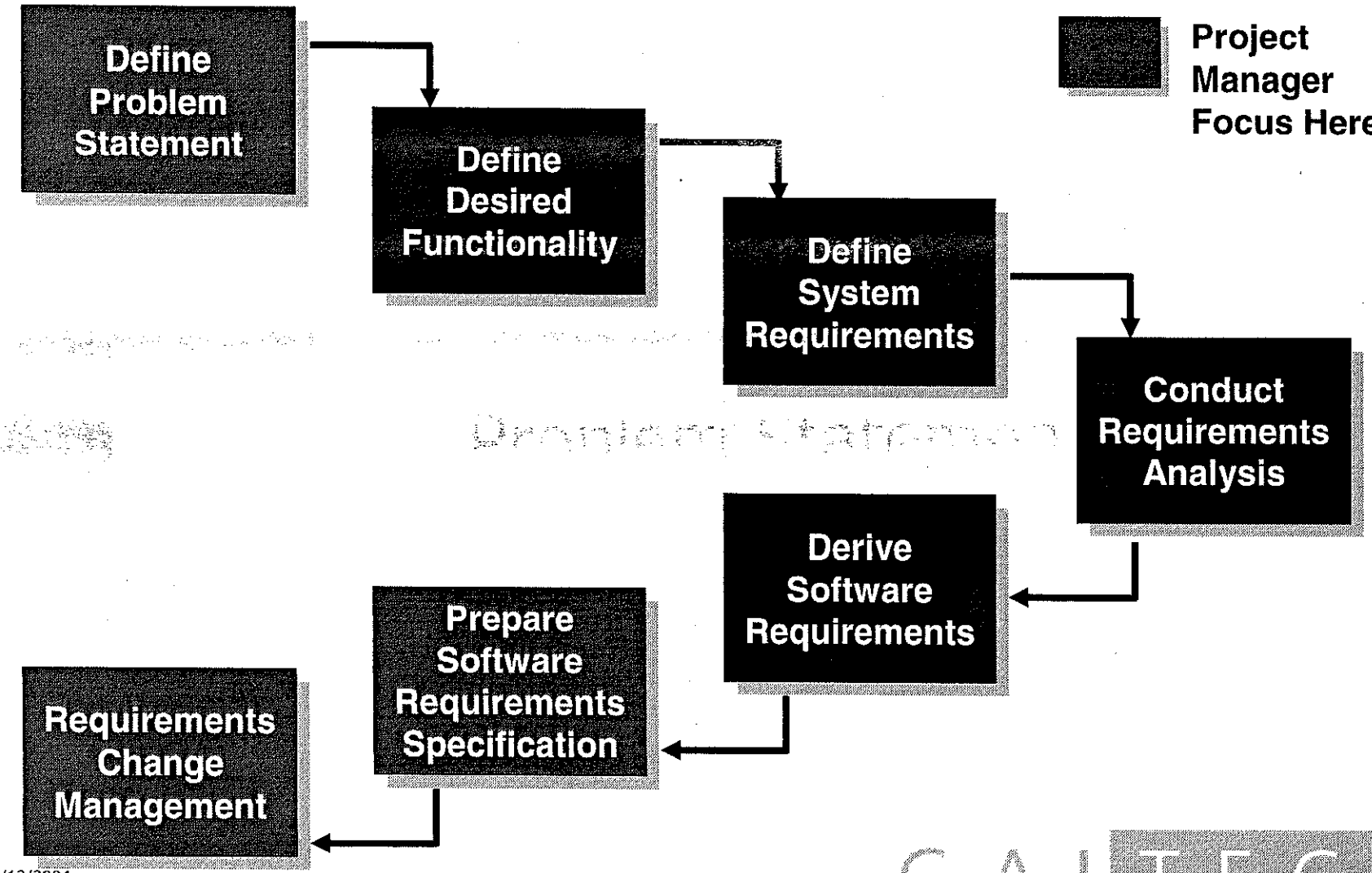
Requirements

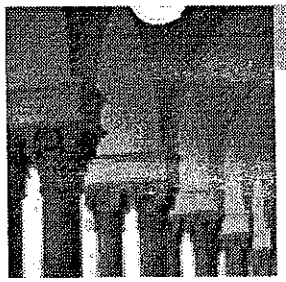
- Objective
  - Work with the customer to define the required features and functions that the software must perform
  - Document requirements to the level of detail required to support system design
  - Determine system requirements allocated to software
- Traps
  - Avoid rushing to judgment - not all requirements are allocated to software





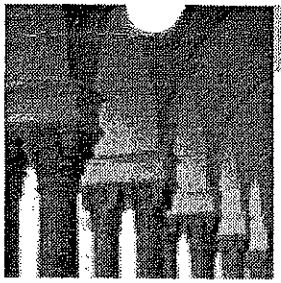
Requirements Activities





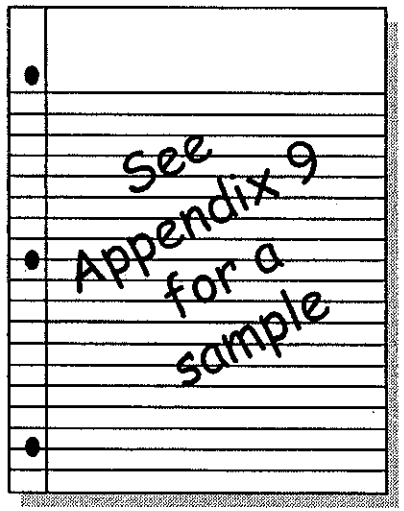
Problem Statement

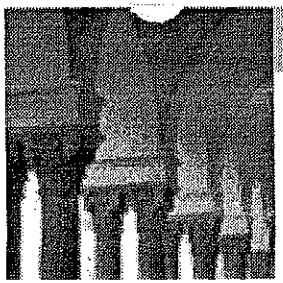
- Without a good problem definition, you often put a great deal of effort into solving the wrong problem and may potentially eliminate valid solutions
- Determine that your people are working with the right client representatives
  - Who is behind the work request?
  - Are they right people, i.e., are their answers official?
  - Will they use the completed product?
  - What are the potential benefits of the solution?
  - Are there valid alternatives to software development?



Software Requirements Specification (SRS)

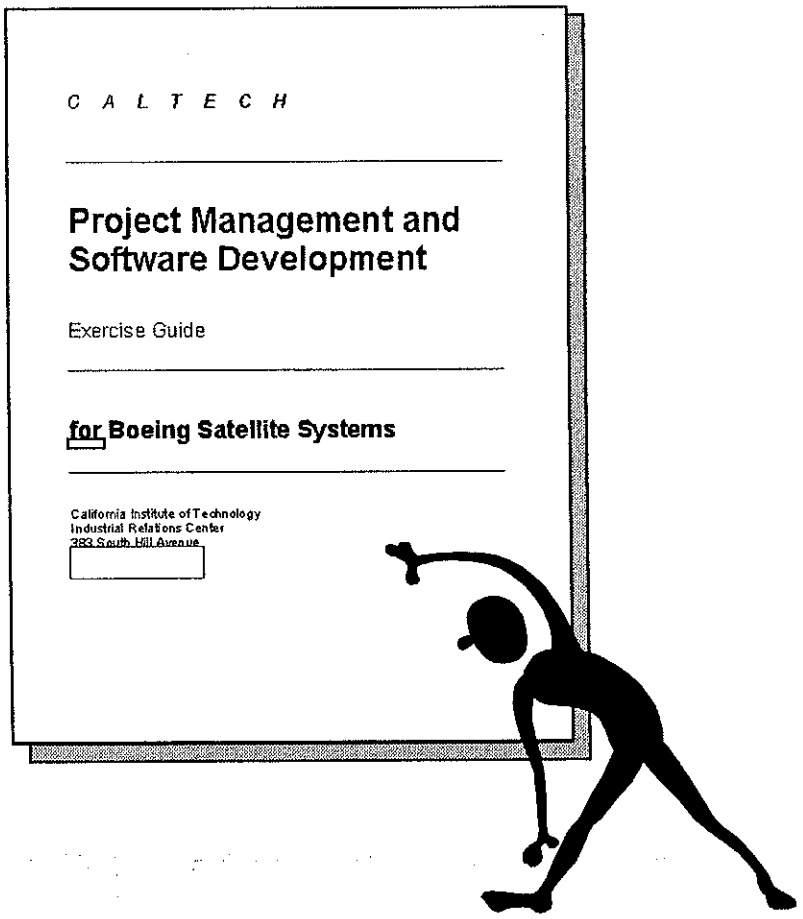
- Technical contract between you and your clients
- Specifies the detailed requirements definition, including inputs, processing and outputs
- May be ANSI or Original
- Approved by
  - Project and Solutions Delivery Manager
  - Software Project Manager
  - Client Manager
  - Other involved groups

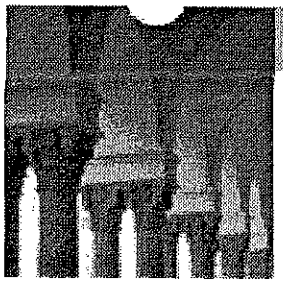




Exercise

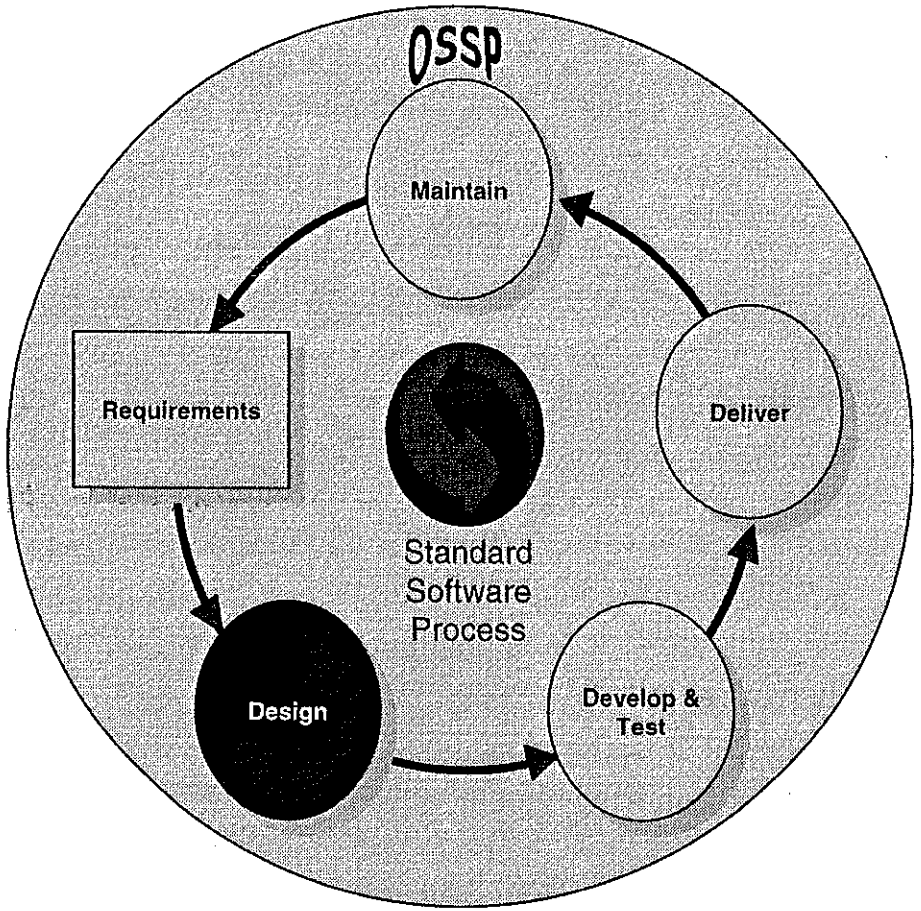
- 5.2 – Writing Requirements

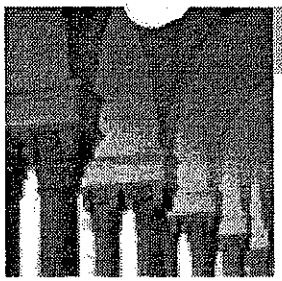




Design

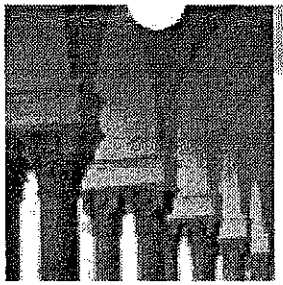
- Objective
  - Produce a detailed specification of the application being developed
  - Translate requirements into concepts that the developer can program
- Traps
  - Project Manager may feel lost
  - Technical staff may get off-track
  - Very programmer/system architect intensive





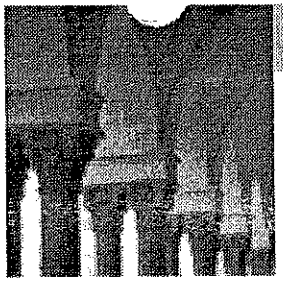
Attributes Of Well-Designed Software

- The software design should:
  - Exhibit a **hierarchical organization** that makes intelligent use of control among software components
  - Be **modular**; logically partitioned into components that perform specific functions and subfunctions
  - Contain separate and **distinct representation** of data and procedure
  - Lead to modules that exhibit **independent** functional characteristics
  - Lead to **interfaces that reduce complexity** of connections between modules and with the external environment
  - Be derived using a **repeatable method** that is driven by information obtained during the requirements phase



Project Manager Aspects

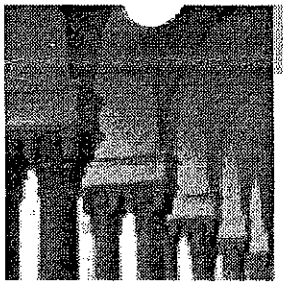
- Design is conducted in two phases
  - Preliminary Design
  - Detail Design
- Preliminary Design
  - Transformation of requirements into data and software architecture
- Detail Design
  - Refinements to the architectural representation of the software into detailed data structures and algorithmic representations



Technical Aspects

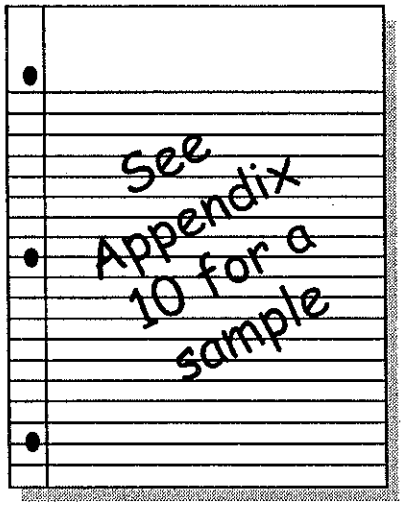
- Procedural design
  - Transforms the structural components into a procedural description of the software (example: source code)

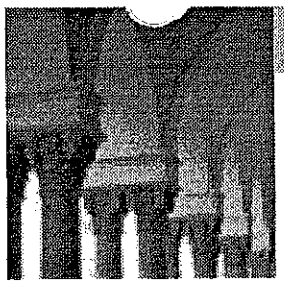
- Interface design
  - Establishes the layout and interaction mechanisms for the human-machine interface (example: GUI)



Software Design Specification (SDS)

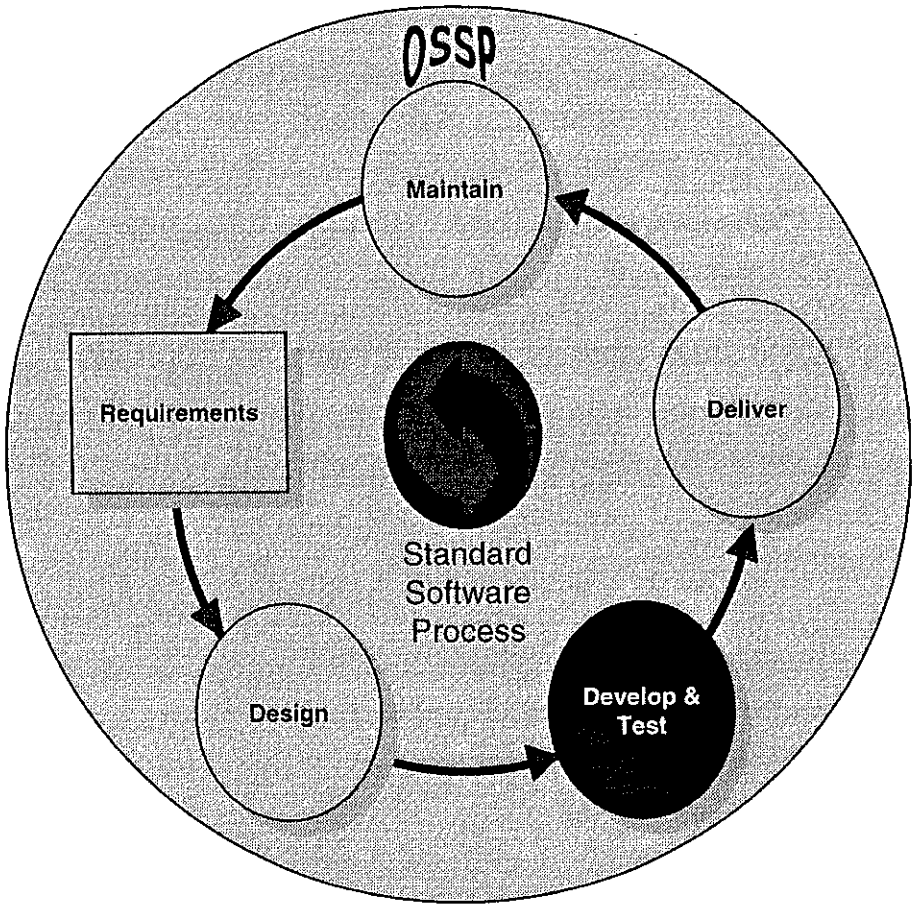
- Technical document which specifies the formal data design, program structure, interfaces, external file structures and global data
- May be ANSI or Original
- Approved by
  - Project manager
  - Program manager
  - Managers of other impacted development groups
  - Client project manager (optional)
  - Client technical lead (optional)

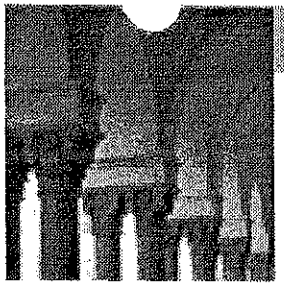




Develop & Test

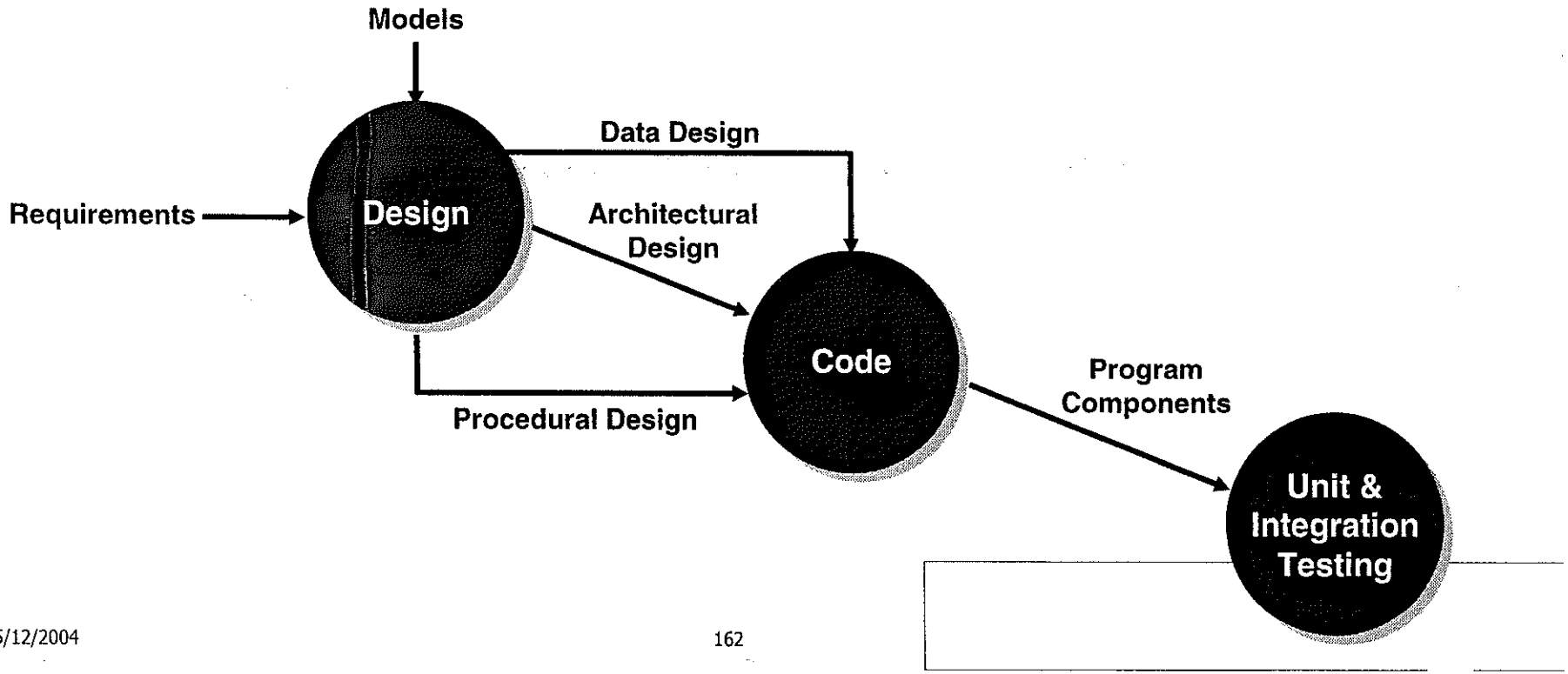
- Objective
  - Develop the code according to the design specification
  - Perform unit testing
  - Perform integration testing
  - Perform system testing
- Traps
  - Avoiding requirements traceability

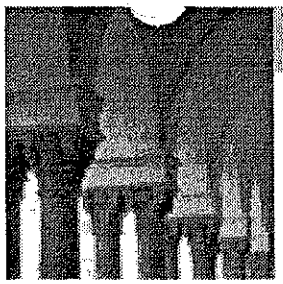




Project Manager Aspects

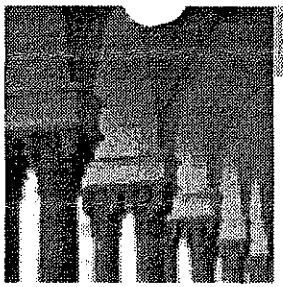
- Project tracking and oversight
- Problem resolution
- Coordination between developers





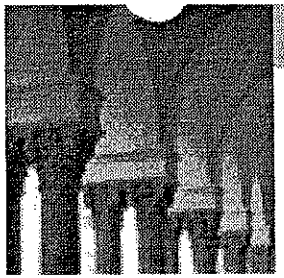
Software Quality Processes

- Documented Requirements
- Change Control
- Technical Reviews (Gates)
- Quality Assurance



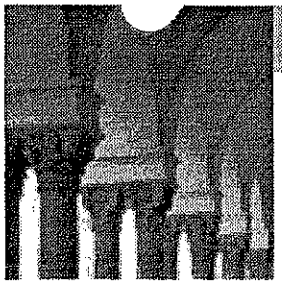
Quality Assurance

- Defect Tracking
  - Should be in place at the beginning of the project
  - Problem Reports and resolution should be managed by version control system
  - Track individually
  - Report collectively
    - Total Open vs. Closed
    - Average time to close (aging)
    - Categories (e.g. system, application, etc.)



Quality Assurance

- System Testing
  - Use independent testers
  - Requirements traceability matrix
  - Beta Testing
- Software Release Criteria
  - Should be spelled out in the Software QA plan
  - Must be measureable

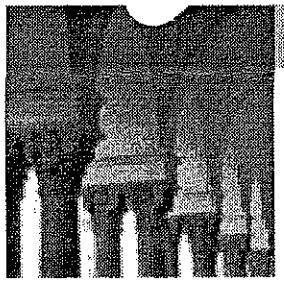


Requirements Traceability Matrix (RTM)

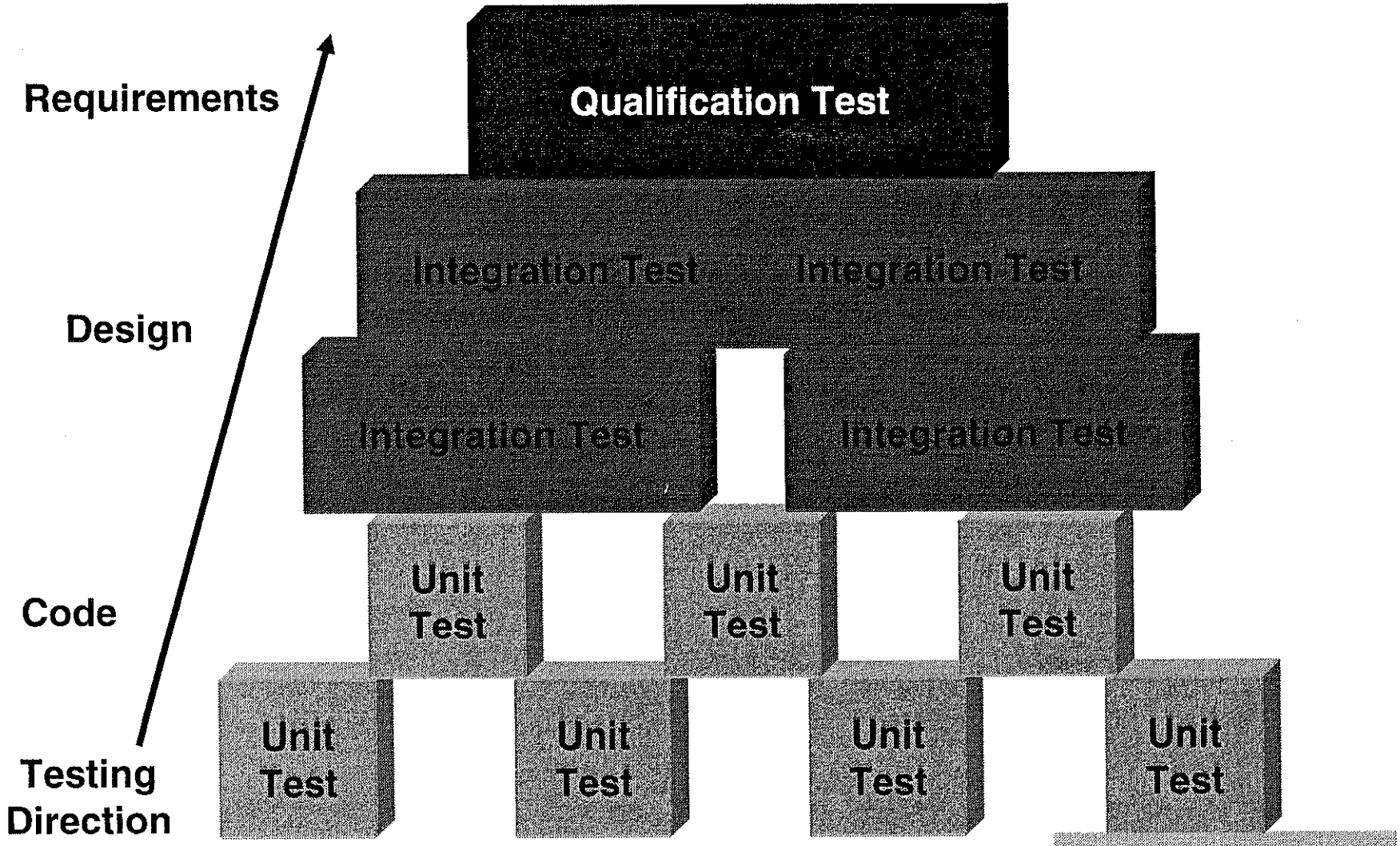
- Uses a tracking number to trace each requirement through implementation

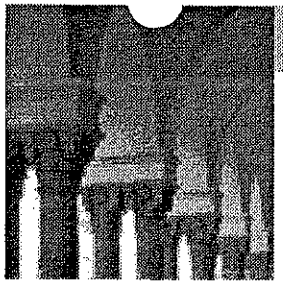
Sample RTM:
Software System Identifier - Generic Security Object

SR Identifier	SR Rev. #	SR Name	SR Description	Release ID/Name	SRS Section	SDS Section	Software Object/ Module	Test Document/ Case	Status	Comments
5010	0	GUI	Provide GUI for security rule definition/maintenance	02.00.00	3.1.1	3.1	GSO20.PBL	Test Plan 2.1.1/5010	Implemented	
5020	0	App Call	Provide a function for an app to call the object for security for a window	02.00.00	3.1.2	3.2	GSO20.PBL	Test Plan 2.1.2/5020	Implemented	
5030	0	Contain er App	Provide a container app to allow for control list database creation	02.00.00	3.1.3	3.3	GSO20.PBL	Test Plan 2.1.3/5030	Implemented	
5040	0	Security Report	Provide a report on security on a per window basis	02.00.00	3.1.4	3.4	GSO20.PBL	Test Plan 2.1.4/5040	Implemented	
5050	0	GSO data backup	Provide a facility to backup GSO tables in SQL format	02.00.00	3.1.5	3.5	GSO20.PBL	Test Plan 2.1.5/5050	Implemented	Open PR on this requirement



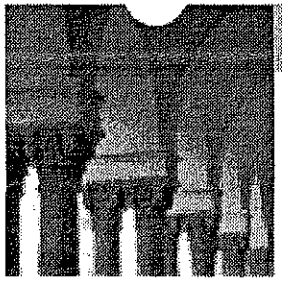
Testing Hierarchy





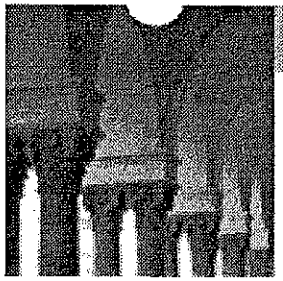
Unit Test

- Tests of individual modules
- Performed by the developer
- Test scenarios are based on the design specifications
- Project Manager actions
  - Ensure unit testing is planned for, conducted and results documented
  - If the modules/libraries don't work, everything else is moot!



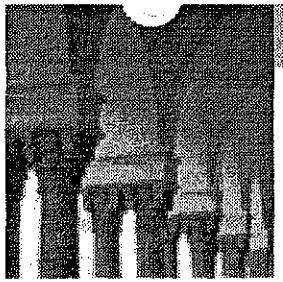
Integration Test

- Demonstrates conformity with the design
- Formal testing of the integration of modules into the program structure
  - Allows for testing in smaller segments
  - Errors easier to detect and isolate
- Test scenarios based on the design specifications
- Project Manager actions
  - Ensure integration testing is planned for, conducted and the results documented
  - Integration testing typically included in the project test plan



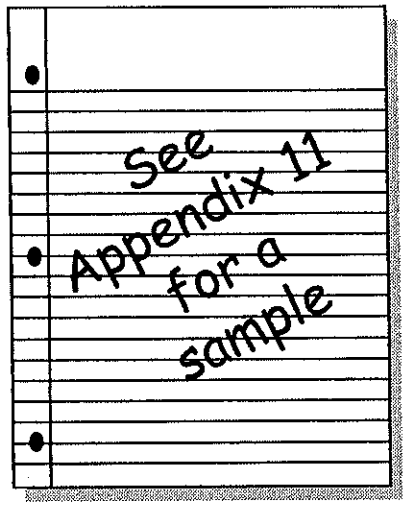
Qualification Test (aka System/Validation Test)

- Demonstrates conformity with requirements
  - Did we implement the requirements?
  - Does the software actually function as designed?
- May include regression testing
  - Re-qualification of existing functionality
  - Typically conducted after major system modifications
  - Scope established in the test plan
- Project Manager responsibilities
  - Make it happen!
  - Don't let schedule pressures compromise qualification testing
  - Ensure that all testing is conducted in a systematic and formal fashion
    - Based on requirements specifications
    - Scope defined by the test plan
  - Manage as part of the overall Verification and Validation

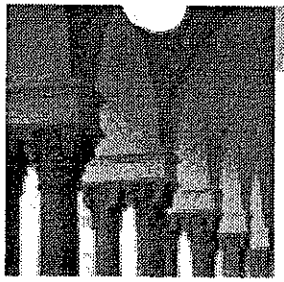


Test Plan

- Specifies test scripts and scenarios
- Governed by V&V Plan
- Project Manager responsibilities
  - Make it happen!
  - Don't let schedule pressures compromise qualification testing
  - Ensure that all testing is conducted in a systematic and formal fashion based on requirements specifications

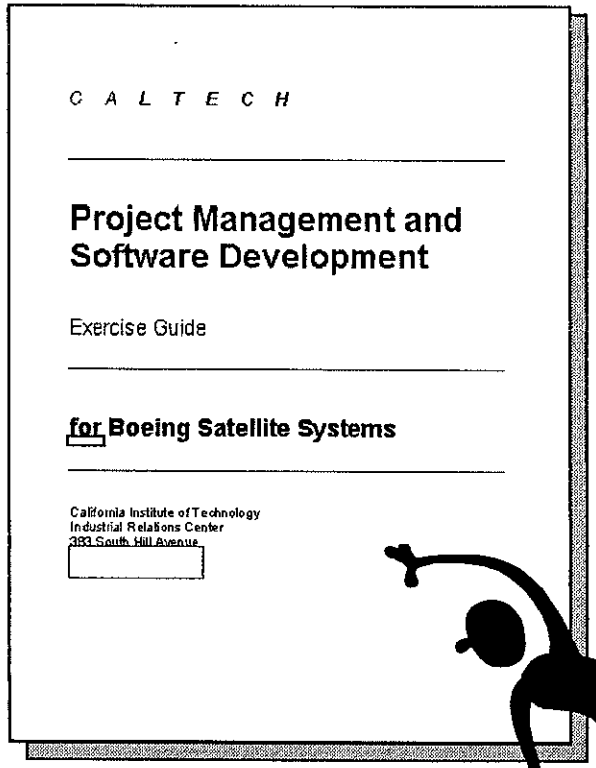


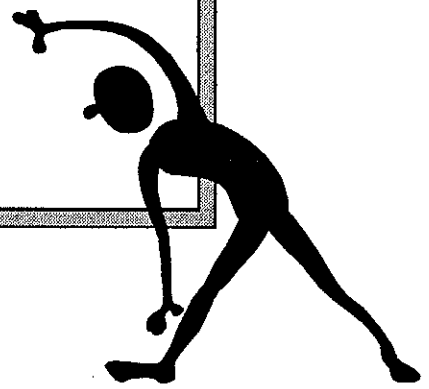
verification = I met requirement
validation = how it was done

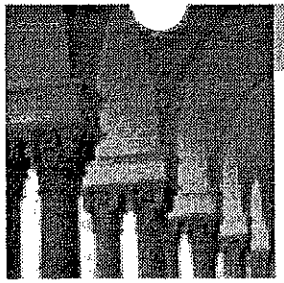


Exercise

- 5.3 – Writing the Test Plan

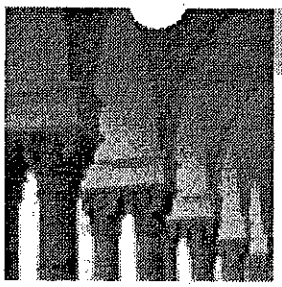






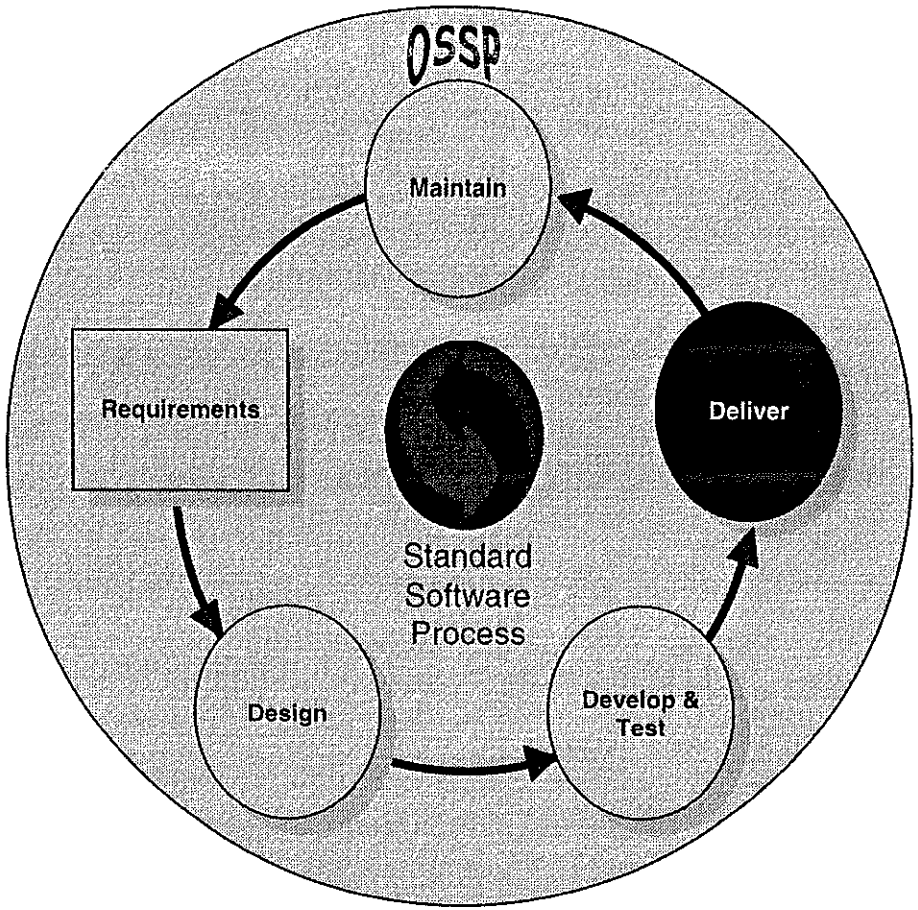
Customer Acceptance Test

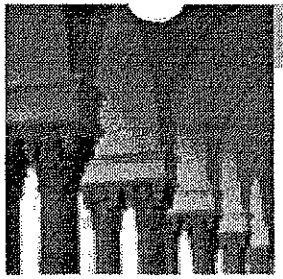
- Customer acceptance testing needs to be managed
  - Challenges
    - Clients are typically not software professionals
    - Usually don't approach testing in a systematic fashion
    - "That's not what I wanted"
  - Assist the client in defining acceptance criteria
    - Document criteria
    - Develop test scenarios
  - Assist the client in conducting their testing
    - Don't do it yourself!
- Formal acceptance is mandatory!



Deliver

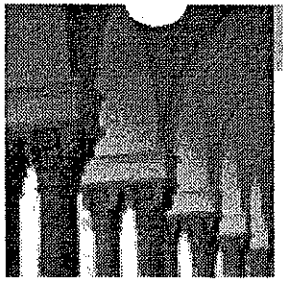
- Objective
  - Deliver user manuals
  - Perform training
  - Install the software
  - Implement the system
  - Provide technical assistance
- Traps
  - Skimping on training and documentation
  - Not planning for support





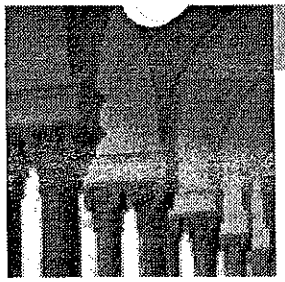
Software Delivery Tasks

- User manuals
  - Should not be a last minute task
  - Typically started during the design phase
  - Can often freeze the design
- Training
  - Usually start training during the final system test or user acceptance testing
  - Actual implementation may be postponed until initial user training is complete



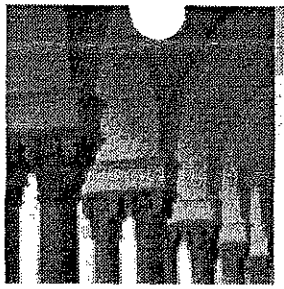
Software Delivery Tasks

- Software installation
  - Deployment mechanisms
    - Autoloaders
    - CD
    - Internet/Intranet
  - Technical assistance



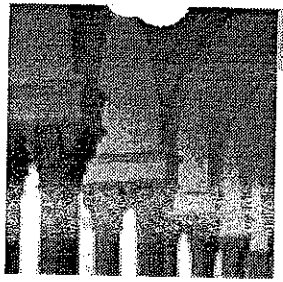
Software Warranty Period

- It is a good practice to provide a certain period of no-cost defect corrections
  - Frequently called a warranty period or warranty run
  - Does not include system enhancements



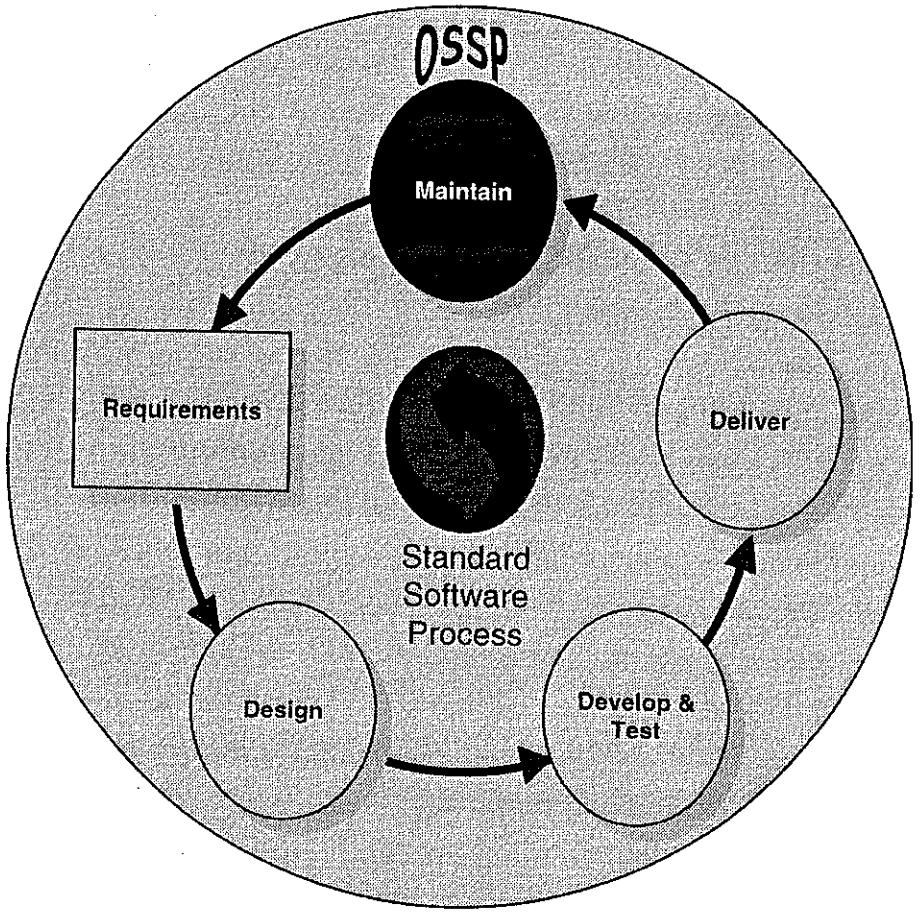
Software Warranty Period

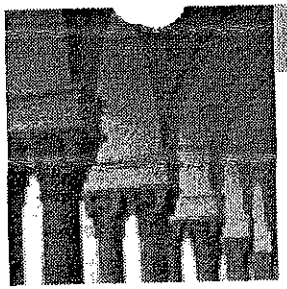
- Tricky legal ground with several issues
  - The warranty protections afforded to purchasers of mass market "shrink-wrap" or "click-wrap" software
  - The practice of marketing software to consumers in the form of a license, rather than the sale of goods
  - The use of shrink-wrap or click-wrap warranties to disclaim substantive implied warranty protections provided by state law
  - The problems inherent in making clear and conspicuous disclosure of material terms in shrink-wrap and click-wrap agreements
  - The unavailability of shrink-wrap or click-wrap license agreements for pre-sale review
  - The role of the Magnuson-Moss Warranty Act in the marketing, sale or licensing of software



Maintain

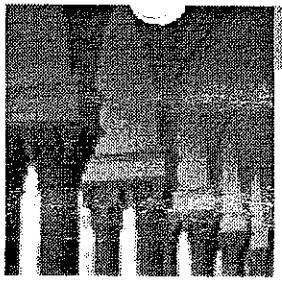
- Objective
  - Provide ongoing support
  - Obtain feedback
  - Evaluate change requests
  - Publish release schedules
- Traps
  - Failing to plan for support resources





In Summary

- Can you explain the differences between the Software Development Lifecycle (SDLC) models?
- List the detailed activities that take place in each project phase?
- Understand your role as a project manager during each of the project phases?



Objectives

- Review the basic methods of tracking project progress
- Understand the types of meetings that are typically held on a software project
- Understand the importance of change management for software projects