SOFTWARE PROCESS AND PROJECT MANAGEMENT

Course code: AIT512

III. B.Tech II semester

Regulation: IARE (R-16)

BY

MR. E SUNIL REDDY

Asst.Professor

DEPARTMENT OF INFORMATION TECHNOLOGY

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

DUNDIGAL, HYDERABAD - 500 043

Course Outcomes

The course should enable the students to:	
CO 1	Describe the concept of Software Development Life Cycle and analyze the concepts of processes, TSP, PSP.
CO 2	Determine the functional requirements, elicitation techniques and Quality Attribute workshop, ACDM, documentation, and specification, change management and traceability of requirements
CO 3	Understand Estimation, Planning, And Tracking
CO 4	Explore the concept of Configuration And Quality Management.
CO 5	Use of Software Process Definition And Management.

UNIT-I

Overview of Software Development Life Cycle, introduction to processes, Personal Software Process(PSP), Team Software Process(TSP), unified processes, agile processes, choosing the right process

Course Learning Outcomes

The course will enable the students to:	
CLO1	Describe the basic concepts of Software Development Life Cycle.
CLO 2	Summarize the concept of processes.
CLO 3	Analyze the concepts of Personal Software Process (PSP), Team Software Process (TSP).
CLO 4	Use the concept of agile processes in real-world problems.

Contents

- Overview of Software Development Life Cycle
- Introduction to processes
- Personal Software Process(PSP)
- Team Software Process(TSP)
- Unified processes
- Agile processes
- Choosing the right process.

Overview of Software Development Life Cycle

Software Development Life Cycle(SDLC) is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

There are following six phases in every Software development life cycle model:

1. Requirement gathering and analysis
2. Design
3. Implementation or coding
4. Testing
5. Deployment
6. Maintenance

Introduction to Processes

A software process (also knows as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

Any software process must include the following four activities:

1. Software specification (or requirements engineering): Define the main functionalities of the software and the constrains around them.
2. Software design and implementation: The software is to be designed and programmed.
3. Software verification and validation: The software must conforms to it's specification and meets the customer needs.
4. Software evolution (software maintenance): The software is being modified to meet customer and market requirements changes.

Introduction to Processes

- In practice, they include sub-activities such as requirements validation, architectural design, unit testing, …etc.
- There are also supporting activities such as configuration and change management, quality assurance, project management, user experience.
- Along with other activities aim to improve the above activities by introducing new techniques, tools, following the best practice, process standardization (so the diversity of software processes is reduced), etc.

Introduction to Processes

A process also includes the process description, which includes:

Products: The outcomes of the an activity. For example, the outcome of architectural design maybe a model for the software architecture.

Roles: The responsibilities of the people involved in the process. For example, the project manager, programmer, etc.

Pre and post conditions: The conditions that must be true before and after an activity. For example, the pre condition of the architectural design is the requirements have been approved by the customer, while the post condition is the diagrams describing the architectural have been reviewed.

Personal Software Process(PSP)

Every developer uses some process to build computer software. The Personal Software Process (PSP) emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product. In addition PSP makes the practitioner responsible for project planning (e.g., estimating and scheduling) and empowers the practitioner to control the quality of all software work products that are developed. The PSP model defines five framework

Planning. This activity isolates requirements and develops both size and resource estimates. In addition, a defect estimate is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.

High-level design. External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.

Personal Software Process(PSP)

High-level design review. Formal verification methods are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.

Development. The component-level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.

Postmortem. Using the measures and metrics collected, the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

PSP emphasizes the need to record and analyze the types of errors you make, so that you can develop strategies to eliminate them.

Team Software Process(TSP)

The Team Software Process (TSP) provides a defined operational Because many industry-grade software projects are addressed by a team of practitioners, The goal of TSP is to build a "self-directed" project team that organizes itself to produce high-quality software. Humphrey defines the following objectives for TSP:

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPTs) of 3 to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.

Team Software Process(TSP)

A self-directed team has a consistent understanding of its overall goals and objectives; defines roles and responsibilities for each team member; tracks quantitative project data identifies a team process that is appropriate for the project and a strategy for implementing the process; defines local standards that are applicable to the team’s software engineering work; continually assesses risk and reacts to it; and tracks, manages, and reports project status.

TSP defines the following framework activities: project launch, high-level design, implementation, integration and test, and postmortem.

Team Software Process(TSP)

TSP makes use of a wide variety of scripts, forms, and standards that serve to guide team members in their work. TSP recognizes that the best software teams are selfdirected. Team members set project objectives, adapt the process to meet their needs, control the project schedule, and through measurement and analysis of the metrics collected, work continually to improve the team’s approach to software engineering.

Team Software Process(TSP)

A self-directed team has a consistent understanding of its overall goals and objectives; defines roles and responsibilities for each team member; tracks quantitative project data identifies a team process that is appropriate for the project and a strategy for implementing the process; defines local standards that are applicable to the team’s software engineering work; continually assesses risk and reacts to it; and tracks, manages, and reports project status.

TSP defines the following framework activities: project launch, high-level design, implementation, integration and test, and postmortem.

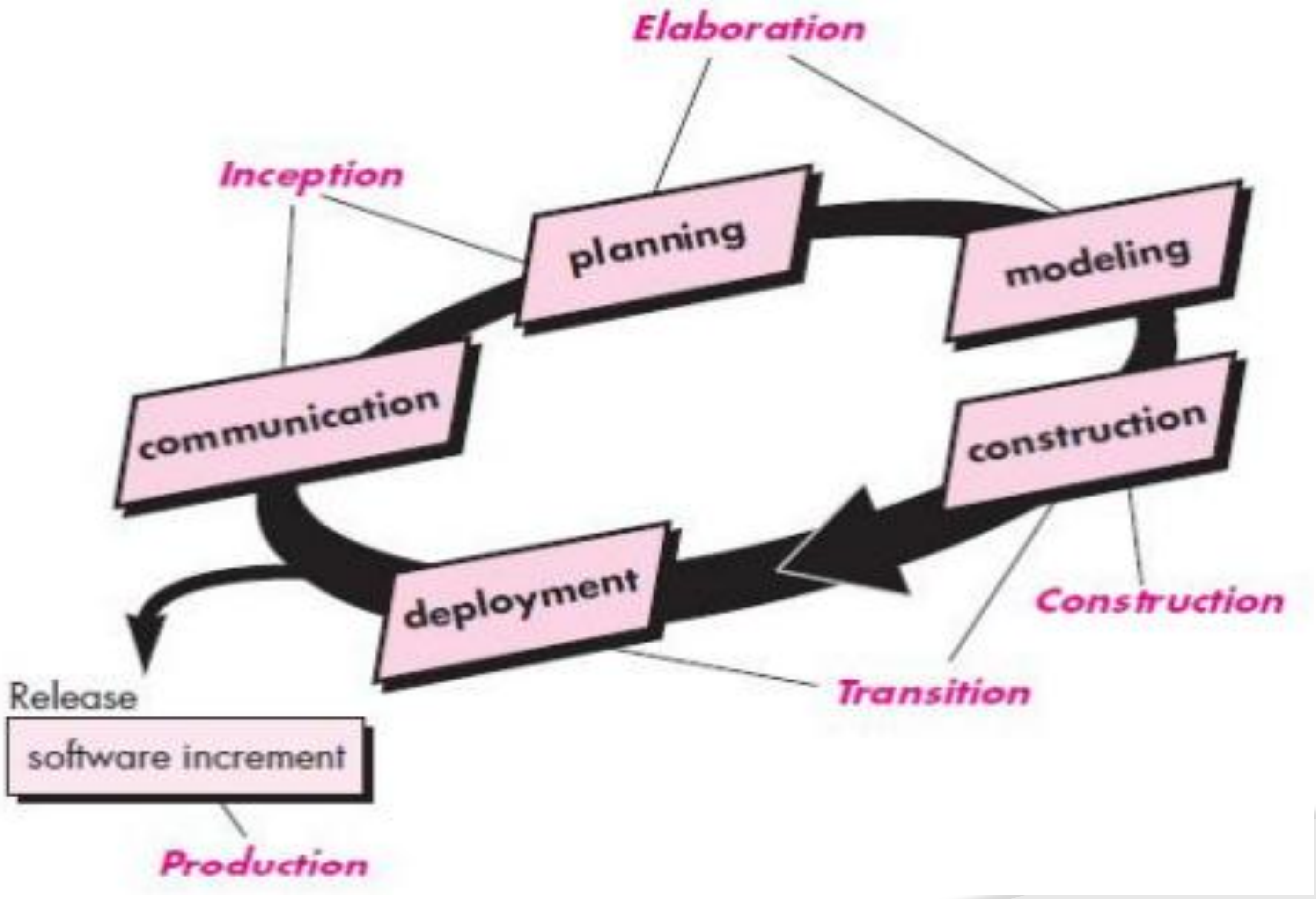
Unified Process

- The Unified Software Development Process or Unified Process is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP). Other examples are Open UP and Agile Unified Process
- The Unified Process divides the project into four phases:
  - Inception: Establish the business case. Identify external entities (actors, systems).[both customer communication and planning activities.]
  - Elaboration: Understand problem domain. Establish architecture, and consider design tradeoffs. Identify project risks. Estimate and schedule project. [communication and modeling activities]

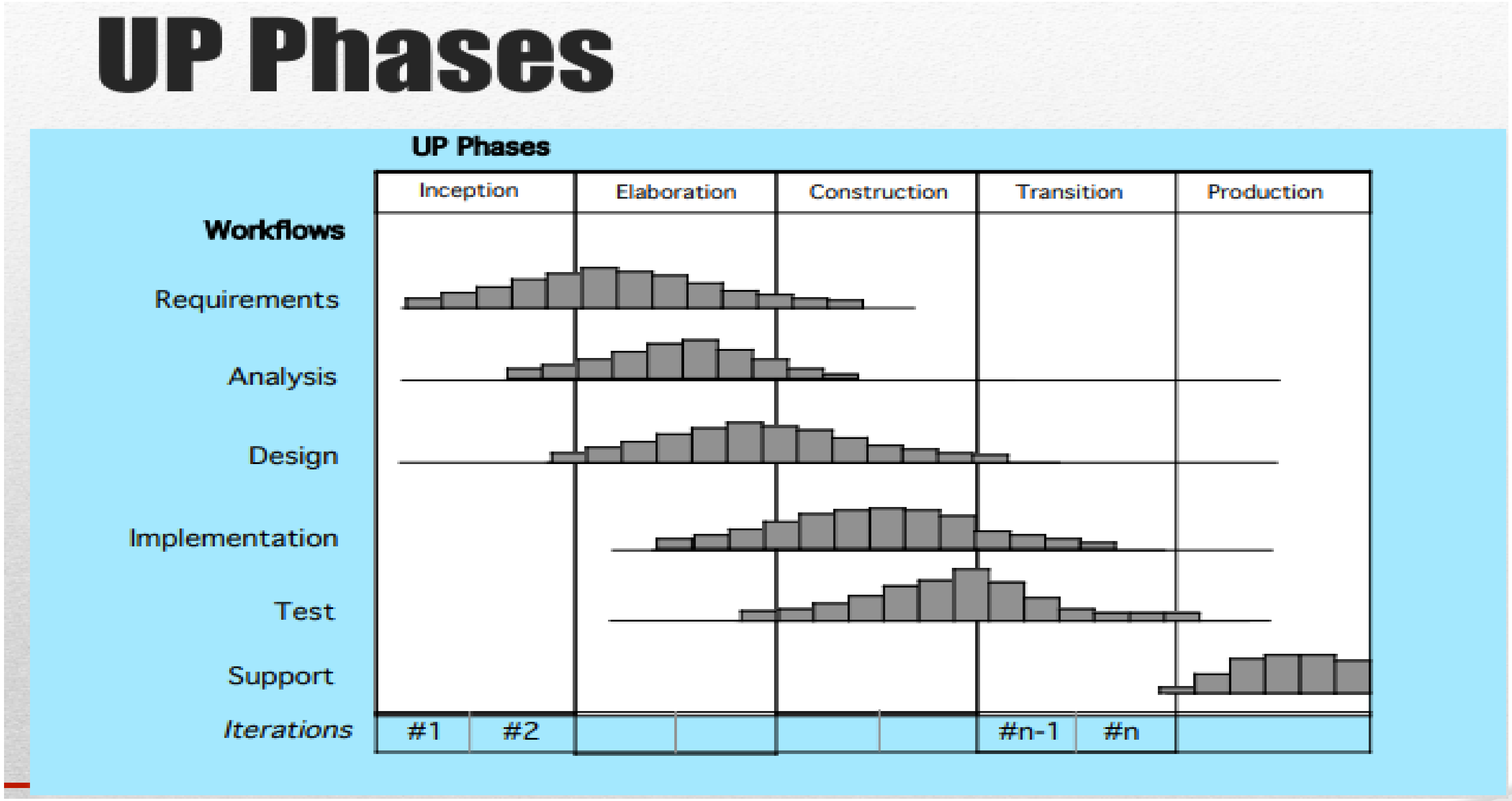
Unified Process

- Construction: Design, program and test. Components are bought and integrated.]construction activity]
- Transition: release a mature version and deploy in real world. [(delivery and feedback) activity]

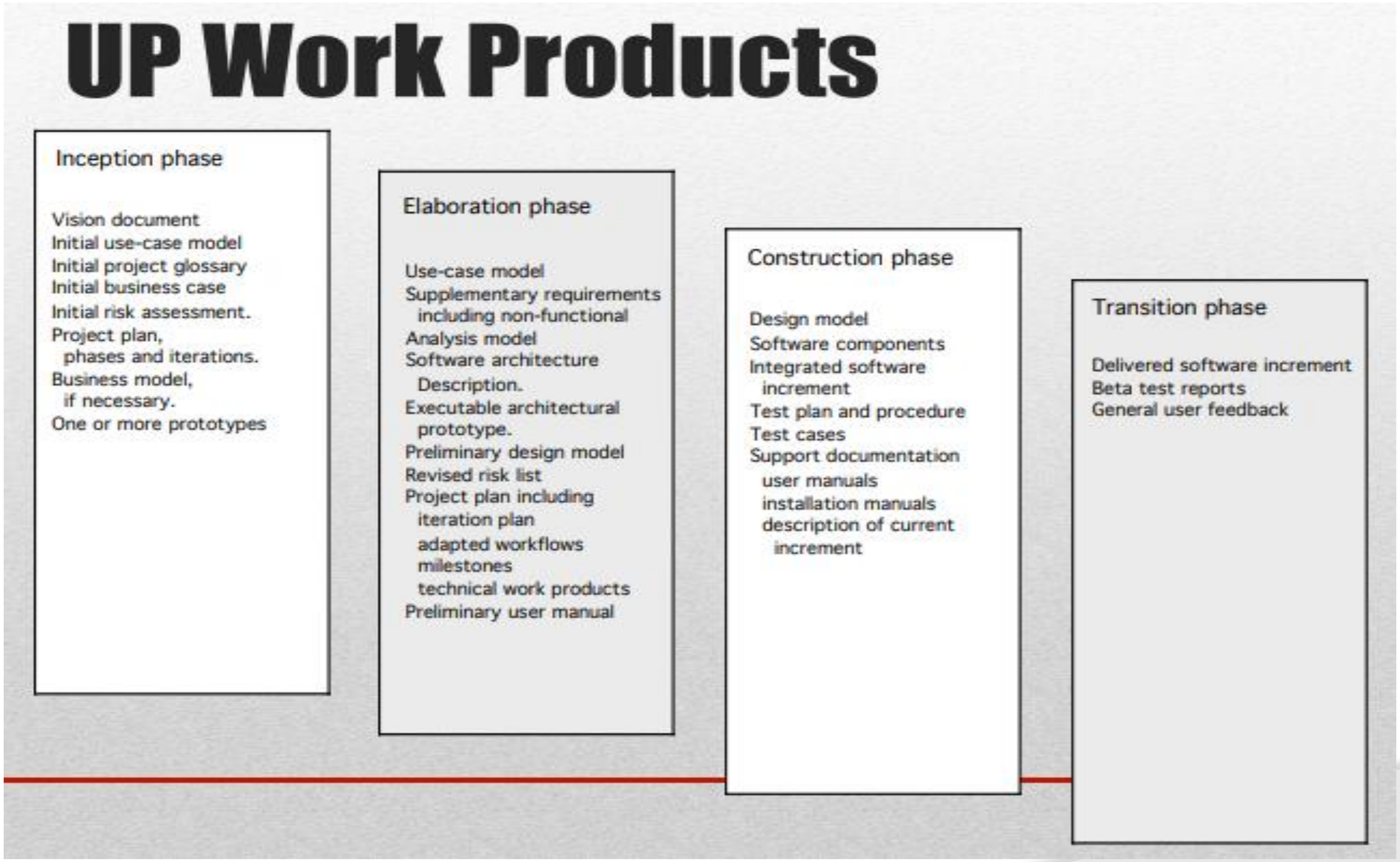
Unified Process



Unified Process



Unified Process



UP Work Products

Inception phase

Vision document
Initial use-case model
Initial project glossary
Initial business case
Initial risk assessment.
Project plan,
phases and iterations.
Business model,
if necessary.
One or more prototypes

Elaboration phase

Use-case model
Supplementary requirements
including non-functional
Analysis model
Software architecture
Description.
Executable architectural
prototype.
Preliminary design model
Revised risk list
Project plan including
iteration plan
adapted workflows
milestones
technical work products
Preliminary user manual

Construction phase

Design model
Software components
Integrated software
increment
Test plan and procedure
Test cases
Support documentation
user manuals
installation manuals
description of current
increment

Transition phase

Delivered software increment
Beta test reports
General user feedback

Agile Process

- Agile software processes is an iterative and incremental based development, where requirements are changeable according to customer needs. It helps in adaptive planning, iterative development and time boxing
- The agile process follows the software development life cycle which includes requirements gathering, analysis, design , coding , testing and delivers partially implemented software and waits for the customer feedback. In the whole process , customer satisfaction is at highest priority with faster development time.

Agile Process

The Agile Alliance defines 12 agility principles for those who want to achieve agility:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months,with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

Agile Process

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self– organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Choosing The Right Process

How to select the right SDLC

Selecting the right SDLC is a process in itself that the organization can implement internally or consult for. There are some steps to get the right selection.

STEP 1: Learn the about SDLC Models

SDLCs are the same in their usage. In order to select the right SDLC, you should have enough experience and be familiar with the SDLCs that will be chosen and understand them correctly.

STEP 2: Assess the needs of Stakeholders

We must study the business domain, stakeholders concerns and requirements, business priorities, our technical capability and ability, and technology constraints to be able to choose the right SDLC against their selection criteria.

Choosing The Right Process

STEP 3: Define the criteria

Some of the selection criteria or arguments that you may use to select an SDLC are:

Is the SDLC suitable for the size of our team and their skills?

Is the SDLC suitable for the selected technology we use for implementing the solution?

Is the SDLC suitable for client and stakeholders concerns and priorities?

Is the SDLC suitable for the geographical situation (distributed team)?

Is the SDLC suitable for the size and complexity of our software?

Is the SDLC suitable for the type of projects we do?

Is the SDLC suitable for our software engineering capability?

Is the SDLC suitable for the project risk and quality insurance?

Choosing The Right Process

STEP 4: Decide

When you define the criteria and the arguments you need to discuss with the team, you will need to have a decision matrix and give each criterion a defined weight and score for each option. After analyzing the results, you should document this decision in the project artifacts and share it with the related stakeholders.

STEP 5: Optimize

You can always optimize the SDLC during the project execution, you may notice upcoming changes do not fit with the selected SDLC, it is okay to align and cope with the changes. You can even make your own SDLC model which optimum for your organization or the type of projects you are involved in.

UNIT-II

Functional requirements and quality attributes, elicitation techniques, Quality Attribute Workshop (QAW), analysis, prioritization, and trade off, Architecture Centric Development Method (ACDM), requirements, documentation, and specification, change management, traceability of requirements.

Course Learning Outcomes

The course will enable the students to:	
CLO1	Determine the Functional requirements and quality attributes,
CLO 2	Understand elicitation techniques, Quality Attribute Workshop (QAW).
CLO 3	Determine the analysis, prioritization, and trade off
CLO 4	Use Architecture Centric Development Method (ACDM).
CLO 5	Illustrate the documentation, and specification.
CLO 6	Describe the change management and traceability of requirements.

Contents

- Functional Requirements and Quality Attributes
- Elicitation Techniques
- Quality Attribute Workshop(QAW)
- Analysis , Prioritization and Trade off
- Architecture Centric Development Method (ACDM)
- Requirements, Documentation and Specification
- Change Management.
- Traceability of Requirements.

Functional Requirements and Quality Attributes

- Requirements of a system can be classified into functional and non functional (i.e. quality attributes). Functional requirements are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- Different types of methods are used to specify functional requirements. Use case driven approaches describe “the ways in which a user uses a system” that is why use case diagram is often used for capturing functional requirements.

Functional Requirements and Quality Attributes

Identify actors and use cases.

For the road pricing system, the actors we identified are:

- Vehicle owner
- Vehicle driver
- Bank
- System clock

The following are the use cases required by the actors listed above:

- Register vehicle
- Pass single toll
- Enter motorway
- Exit motorway
- Pay bill

Functional Requirements and Quality Attributes

Quality attributes define global properties of a system. Usually these are only dealt with in the later stages of a software development process, such as design and implementation

Identify quality attributes.

Quality attributes can be assumptions, constraints or goals of stakeholders. By analysing the initial of set requirements, the potential quality attributes are identified.

For example

fundamental quality attribute are Security issue, response time
Other concerns are identified in a similar fashion: Multiuser System, Compatibility, Legal Issues, Correctness and Availability

Integrate functional requirements with crosscutting quality attributes

Elicitation Techniques

A major goal of Requirements Elicitation is to avoid the confusions between stakeholders and analysts

It is important to distinguish different elicitation methods according to the four methods of communication.

1. Conversational
    - Interviews
    - Questionnaire
    - Brainstorming
2. Observational
3. Analytic
    - Laddering
    - Repertory grid
4. Synthetic

Quality Attribute Workshop(QAW)

The Quality Attribute Workshop (QAW) is a facilitated method that engages system stakeholders early in the life cycle to discover the driving quality attributes of a software-intensive system

The QAW involves the following steps:

1. QAW Presentation and Introductions.
2. Business/Mission Presentation.
3. Architectural Plan Presentation.
4. Identification of Architectural Drivers.
5. Scenario Brainstorming.
6. Scenario Consolidation.
7. Scenario Prioritization.
8. Scenario Refinement.

Analysis , Prioritization and Trade off

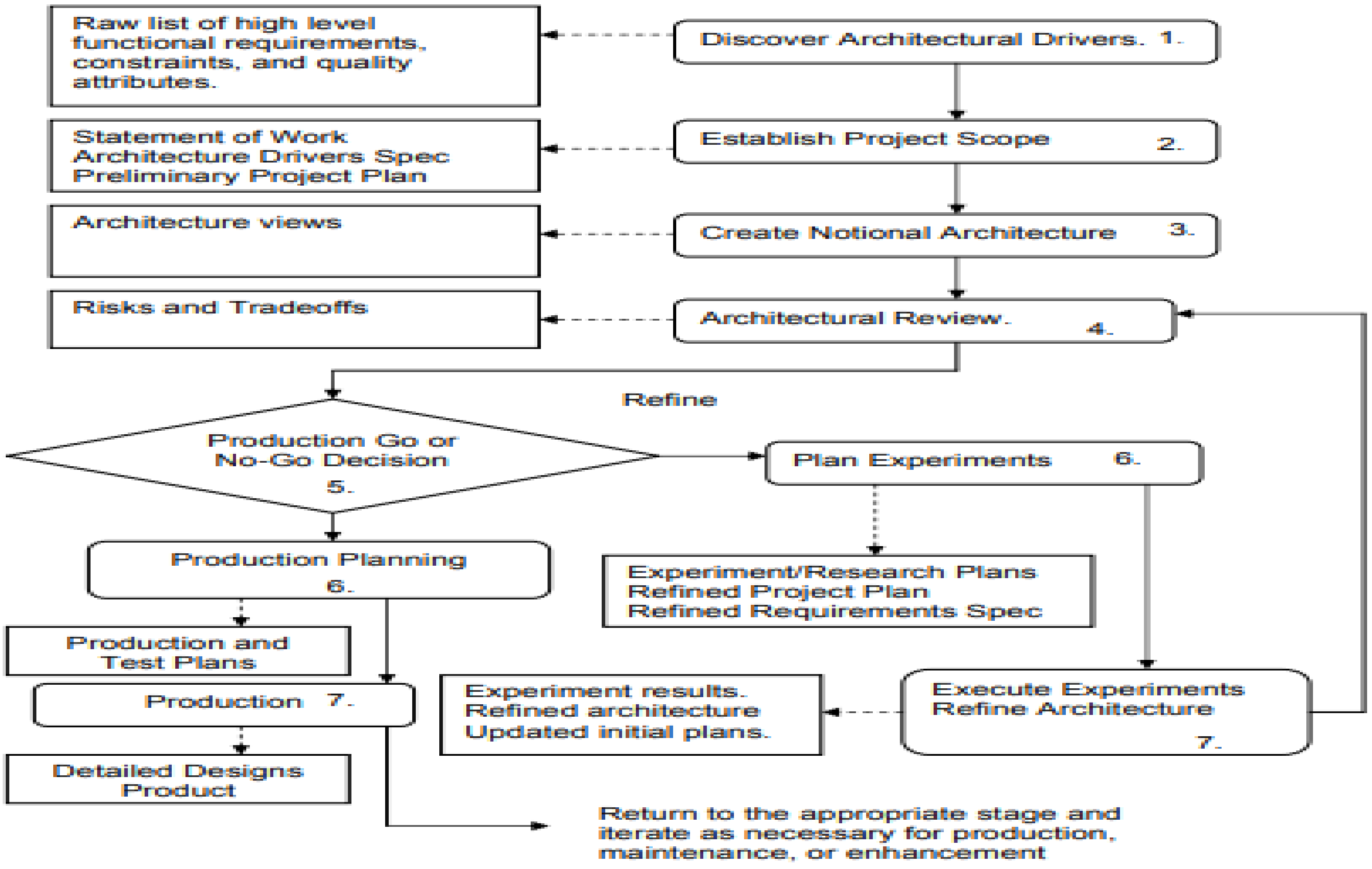
The goal of requirement analysis phase is answer to question:
what software must do (and with what constraints)?
The goal of software analysis phase is answer to question: how system should work?

- Software engineering elements that are used during analysis phase: notations for model record, methods of model preparation tools for easy use of notations and methods.
- Prioritizing requirements helps the project team to understand which requirements are most important and most urgent. Based on this finding a software engineer can decide what to develop/implement in the first release and what on the coming releases. Prioritization is also a useful activity for decision making in other phases of software engineering like development testing, and implementation. There are a number of techniques available to prioritize the requirements with their associated strengths and limitations.

Analysis , Prioritization and Trade off

ATAM(Architecture Tradeoff Analysis Method) STEPS
Step 0 - Planning/Information exchange
Step 1 - Scenario brainstorming
Step 2 - Architecture presentation
Step 3 - Scenario coverage checking
Step 4 - Scenario grouping and prioritization
Step 5 - Map high priority scenarios onto architecture
Step 6 - Perform quality attribute-specific analysis

Architecture Centric Development Method (ACDM)


Raw list of high level functional requirements, constraints, and quality attributes.
Discover Architectural Drivers. 1.
Statement of Work
Architecture Drivers Spec
Preliminary Project Plan
Establish Project Scope 2.
Architecture views
Create Notional Architecture 3.
Risks and Tradeoffs
Architectural Review. 4.
Refine
Production Go or No-Go Decision 5.
Plan Experiments 6.
Production Planning 6.
Experiment/Research Plans
Refined Project Plan
Refined Requirements Spec
Production and Test Plans
Production 7.
Experiment results.
Refined architecture
Updated initial plans.
Execute Experiments
Refine Architecture 7.
Detailed Designs
Product
Return to the appropriate stage and iterate as necessary for production, maintenance, or enhancement

Requirements, Documentation and Specification

- Software requirements specification establishes the basis for agreement between customers and contractors or suppliers (in market-driven projects, these roles may be played by the marketing and development divisions) on what the software product is to do as well as what it is not expected to do. Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules.
- The software requirements specification document enlists enough and necessary requirements that are required for the project development. To derive the requirements we need to have clear and thorough understanding of the products to be developed or being developed

Change Management

Effectively managing organizational change is a four-step process:

1. Recognizing the changes in the broader business environment
2. Developing the necessary adjustments for their company’s needs
3. Training their employees on the appropriate changes
4. Winning the support of the employees with the persuasiveness of the appropriate adjustments

Change Management

Successful change management is more likely to occur if the following are included

1. Benefits management and realization
2. Effective communication
3. Devise an effective education, training and/or skills
4. Counter resistance from the employees of companies and align them to overall strategic direction of the organization
5. Provide personal counselling (if required) to alleviate any change-related fears
6. Monitoring of the implementation and fine-tuning as required

Traceability of Requirements

Traceability is the ability to verify the history, location, or application of an item by means of documented recorded identification

Measurement

Logistics:

traceability refers to the capability for tracing goods along the distribution chain on a batch number or series number basis

Materials:

associate a finished part with results of a test performed on a sample from the same melt identified by the unique lot number of the material

Supply chain:

Environmentally friendly retailers may choose to make information regarding their supply chain freely available to customers

Traceability of Requirements

Software development:

refers to the recording through means of barcodes or RFID tags & other tracking media

Food processing:

refers to the recording through means of barcodes or RFID tags & other tracking media

Forest products:

a new tool to verify claims and assure buyers about the source of their materials

UNIT-III

Identifying and prioritizing risks, risk mitigation plans, estimation techniques, use case points, function points, COCOMO II, top down estimation, bottom up estimation. Work break down structure, macro and micro plans, planning poker, wideband Delphi, documenting the plan, tracking the plan, Earned Value Method (EVM).

Course Learning Outcomes

The course will enable the students to:	
CLO1	Explain software risks.
CLO 2	Understand the concept of function points, COCOMO II, estimations
CLO 3	Understand the Work break down structure, macro and micro plans
CLO 4	Understand the planning poker ,wideband Delphi
CLO 5	Summarize the tracking the plan ,Earned Value Method (EVM)

Contents

- Identifying and Prioritizing risks.
- Risk Mitigation Plans.
- Estimation Techniques.
- Use case Points, Function Points
- COCOMO II.
- Top down Estimation, Bottom up Estimation.
- Work Break Down Structure.
- Macro and Micro plans.
- Planning Poker
- Wideband Delphi.
- Documenting the plan, Tracking the Plan.
- Earned Value Method (EVM).

Identifying and Prioritizing risks

The process of determining which risks may affect the project and documenting their characteristics.

Risk Assessment:

The Risk Assessment and Mitigation tab in the Risk Management workbook has a set of questions that need to be answered that help determine the risk level of the project.

Risk Register:

This is located on the project's SharePoint site where project specific risks can be entered

Risk Analysis:

The process of analyzing and prioritizing risk. The analyzing and prioritizing of risks is done in the Risk Management Workbook on the Risk Assessment-Mitigation tab and in the Risk Register. Risks are prioritized as High, Medium or Low. The prioritization of risks, determines other steps that may need to happen.

Risk Mitigation Plans

Mitigating Actions to Consider

Risk Avoidance –
Actions taken to eliminate the source of risk (e.g. change vendor, lower requirements, change project team member, etc.)

Risk Mitigation –
Actions taken to mitigate the severity and consequences of a risk (e.g. greater training, delayed deployment, etc.)

Risk Transfer –
The transfer of risk from one group to another (e.g. purchasing insurance, etc.)

Risk Monitoring - The monitoring and periodic re-evaluation of a risk for changes to key risk parameters

Risk Acceptance - Acknowledging the risk but not taking preventive measures

Estimation Techniques

There is no simple way to make accurate estimates of the effort required – Initially, not much detail is given – Technologies and people may be unknown

Project cost estimates may be self-fulfilling – Estimate defines budget, project adjusted to meet budget

Estimation of software projects can be done by different techniques.

The important techniques are:

Estimation by Expert Judgement

Estimation by Analogy

Estimation by Available Resources

Estimation by Software Price

Estimation by Algorithmic cost Modelling-Model is built based on historical cost information-Generally based on the size of the software

Estimation Techniques

Estimation by Expert Judgement

Several experts in software development and the application domain are consulted

Process iterates until some consensus is reached

Advantages: Relatively cheap estimation method. Can be accurate if experts have direct experience of similar systems

Disadvantages: Very inaccurate if there are no experts!

Estimation by analogy

The project is compared to a similar project in the same application domain

Advantages: Accurate if project data available

Disadvantages: Impossible if no comparable project has been tackled

Estimation Techniques

Pricing to win

The project costs whatever the customer has to spend on it

Advantages: You get the contract

Disadvantages: The probability that the customer gets the system he or she wants is small. Often, costs do not accurately reflect the work required

Use case Points

Use Case Points are used as an analysis phase technique for estimating software development. Use Case Points are derived from another software development estimating technique called "Function Points." However, Function Points are used by systems analysts as a development phase technique that requires technical detail for estimating.

Use Case Points

Conducted by business analysts during the analysis phase of a project Can be classified as a class IV or III estimate (4)
Is based on information contained in a business requirement document Functional requirements as modelled via system use cases, actors, and scenarios Non functional requirements Assumptions on developmental risks

Use case Points

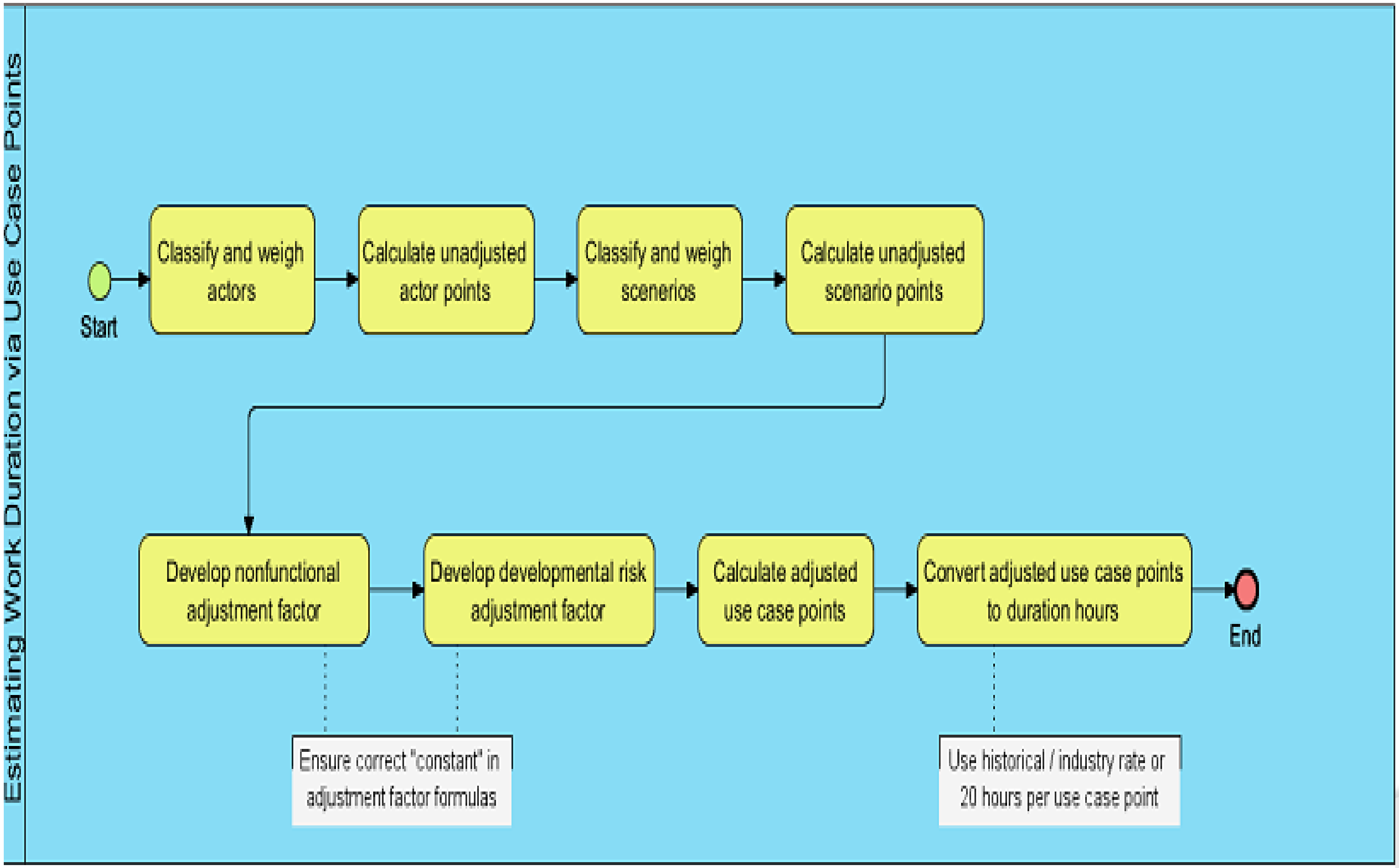

Estimating Work Duration via Use Case Points
Start
Classify and weigh actors
Calculate unadjusted actor points
Classify and weigh scenerios
Calculate unadjusted scenario points
Develop nonfunctional adjustment factor
Develop developmental risk adjustment factor
Calculate adjusted use case points
Convert adjusted use case points to duration hours
End
Ensure correct "constant" in adjustment factor formulas
Use historical / industry rate or 20 hours per use case point

Figure 1. Use Case Points Estimating Process

Function Points

Function Points

•Conducted by systems analysts during the development phase
•Can be classified as a class II or I estimate which should be more detailed (4)
•Is based on information contained in a technical specification
•Data Functions using internal and external logical files
•Transaction Functions using external inputs and external outputs, plus
external inquiries

FPA Uses and Benefits in Project Planning
FPA Uses and Benefits in Project Construction
FPA Uses and Benefits after Software Implementation

COCOMO II

COCOMO 2 levels

■ Early prototyping model–Estimates based on OP and a simple formula

■ Early design model – Estimates based on FP that are translated to LOC

■ Reuse model–Estimates effort to integrate reused and generated code

■ Post-architecture level – Estimates based on lines of source code

Effort = A × (Size)B × M

Effort in terms of person-months

•A: 2.45 in 1998

•Size: Estimated Size in KLOC

•B: combined process factors

•M: combined effort factors

Top down Estimation, Bottom up Estimation

Top-down:

Starts at the system level and assess system functionality and its delivery through subsystems

– Usable without much knowledge

– Factors in integration

– configuration and documentation costs

–Can underestimate low-level problems

Bottom-up:

Start at component level and aggregate to obtain system effort

-Usable when architecture of the system is known

–May underestimate system-level activities such as integration

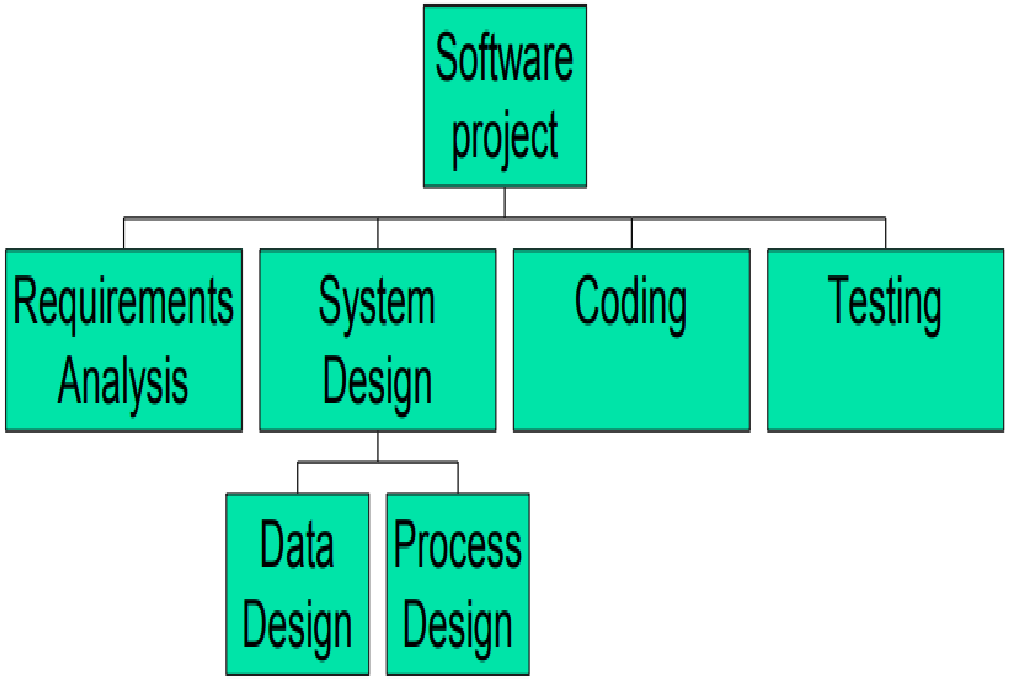
Work Break Down Structure

Overall work has to be decomposed into manageable units
Complex tasks are broken down into subtasks and further refined called Work Breakdown Structures (WBS).
WBS Contains a list of activities, derived from:
– Previous experience
– Expert brainstorming
WBS helps in
– identifying the main activities
– break each main activity down into sub activities which can further be
broken down into lower level sub-activities
Creating WBS
■ Phase based approach
■ Product based approach
■ Hybrid approach

Work Break Down Structure

- Phase based approach

Work Breakdown Structure (an extract)



Advantage

– Activity list likely complete and non overlapping

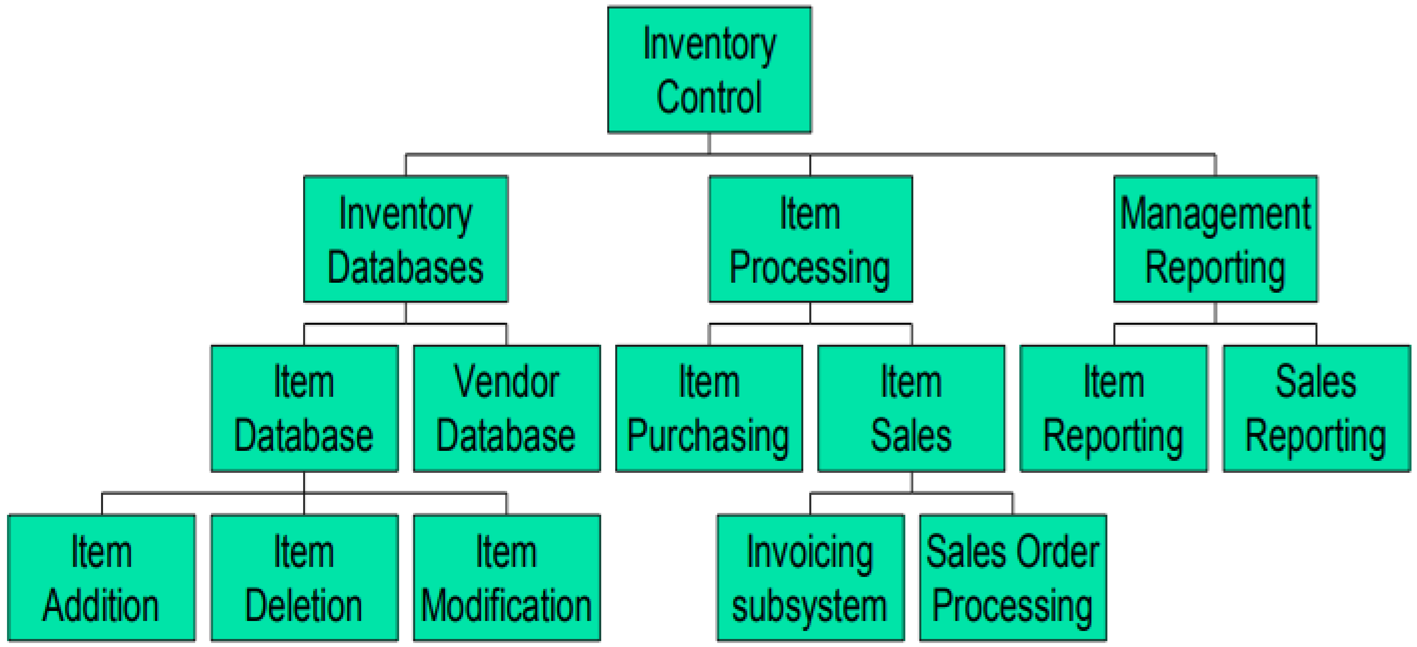
Disadvantage

– May miss some activities related to final product

Work Break Down Structure

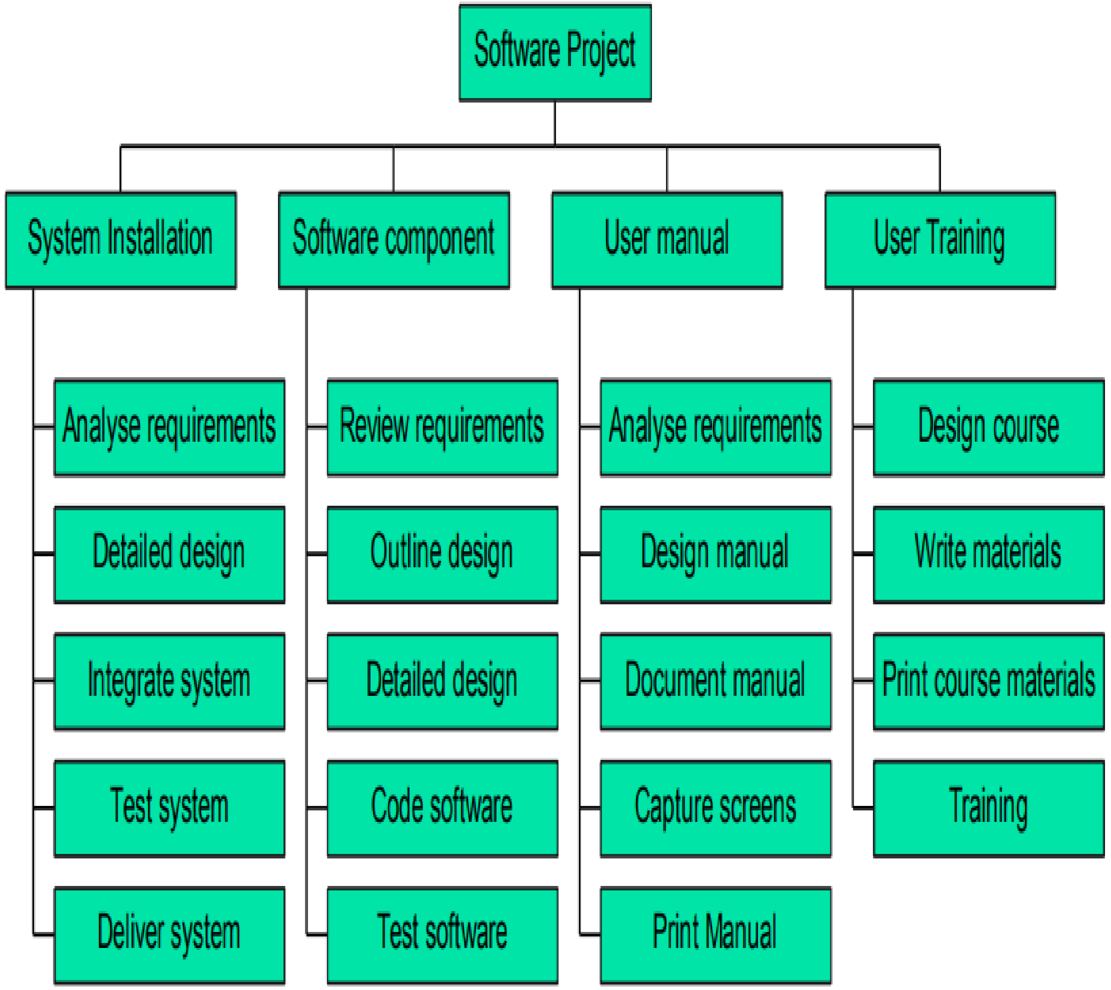
- Product based approach

A Product Breakdown Structure (an extract)



Work Break Down Structure

- Hybrid based approach-
A mix of the phase-based and product based approaches (most commonly used)
- The WBS consists of
– a list of the products of the project
– a list of phases for each product



Macro and Micro plans

The Macro or Top-Down approach can provide a quick but rough estimate

- Done when the time and expense of a detailed estimate are an issue
- Usually occurs during conception stage when a full design and WBS are not available
- Requires experienced personnel to do the estimate
- Can be highly inaccurate

A Micro or Bottom-Up approach can provide a fairly accurate estimate, but is time consuming

- Takes into account the project design and a “roll-up” of WBS elements
- May require multiple personnel and time to complete
- If done properly, a bottom-up estimate can yield accurate cost and time estimates

Planning Poker

- Planning Poker is a consensus-based technique for estimating, mostly used to estimate effort or relative size of user stories in Scrum.
- Planning Poker combines three estimation techniques − Wideband Delphi Technique, Analogous Estimation, and Estimation using WBS.
- Planning Poker was first defined and named by James Grenning in 2002 and later popularized by Mike Cohn in his book "Agile Estimating and Planning”, whose company trade marked the term.

Planning Poker

In Planning Poker Estimation Technique, estimates for the user stories are derived by playing planning poker. The entire Scrum team is involved and it results in quick but reliable estimates.

Planning Poker is played with a deck of cards. As Fibonacci sequence is used, the cards have numbers - 1, 2, 3, 5, 8, 13, 21, 34, etc.

These numbers represent the “Story Points”. Each estimator has a deck of cards. The numbers on the cards should be large enough to be visible to all the team members, when one of the team members holds up a card.

Planning Poker

- One of the team members is selected as the Moderator. The moderator reads the description of the user story for which estimation is being made. If the estimators have any questions, product owner answers them.
- Each estimator privately selects a card representing his or her estimate. Cards are not shown until all the estimators have made a selection. At that time, all cards are simultaneously turned over and held up so that all team members can see each estimate.
- In the first round, it is very likely that the estimations vary. The high and low estimators explain the reason for their estimates. Care should be taken that all the discussions are meant for understanding only and nothing is to be taken personally. The moderator has to ensure the same.

Planning Poker

- The team can discuss the story and their estimates for a few more minutes.

- The moderator can take notes on the discussion that will be helpful when the specific story is developed. After the discussion, each estimator re-estimates by again selecting a card. Cards are once again kept private until everyone has estimated, at which point they are turned over at the same time

Wideband Delphi

Wideband Delphi Technique – Steps

Step 1 – Choose the Estimation team and a moderator.

Step 2 – The moderator conducts the kickoff meeting

Step 3 – Each Estimation team member then individually generates a detailed WBS, estimates each task in the WBS, and documents the assumptions made

Step 4 – The moderator calls the Estimation team for the Estimation meeting.

Step 5 – The entire Estimation team assembles for the estimation meeting.

Step 6 – The Project Manager then assembles the results from the Estimation meeting.

Wideband Delphi

Advantages

Wideband Delphi Technique is a consensus-based estimation technique for estimating effort.

Useful when estimating time to do a task.

Participation of experienced people and they individually estimating would lead to reliable results.

People who would do the work are making estimates thus making valid estimates.

Anonymity maintained throughout makes it possible for everyone to express their results confidently.

A very simple technique.

Assumptions are documented, discussed and agreed.

Disadvantages

Management support is required.

The estimation results may not be what the management wants to hear.

Documenting the plan, Tracking the Plan

seven planning documents.

1.Project management plan

2. High-level project schedule plan

3. Project team planning

4. Scope plan

5. Detailed project work plan

6. Quality assurance planning

7. Risk planning

Earned Value Method (EVM)

Earned value management (EVM), or Earned value project/performance management (EVPM) is a Project Management technique for measuring project performance and progress in an objective manner.

It has the ability to combine measurements of:

•Scope

•Schedule, and

•Costs

Essential features of any EVM implementation include

1.a project plan that identifies work to be accomplished,

2.a valuation of planned work, called Planned Value (PV) or Budgeted cost of Work Scheduled (BCWS), and

3.pre-defined “earning rules” (also called metrics) to quantify the accomplishment of work, called Earned Value (EV) or Budgeted cost of Work Scheduled (BCWP).

UNIT-IV

Identifying articrafts to be configured, naming conventions and version control, configuration control, quality assurance techniques, peer reviews, Fagan inspection, unit, registration, system, and acceptance testing, test data and test cases, bug tracking, casual analysis

Course Learning Outcomes

The course will enable the students to:	
CLO1	Identifying articrafts to be configured, naming conventions
CLO 2	Understand the version control, configuration control, quality assurance techniques.
CLO 3	Summarize the concept of peer reviews, Fagan inspection
CLO 4	Apply testing of unit, registration, system, and acceptance, test data and test cases..
CLO 5	Understand the bug tracking, casual analysis.

Contents

- Identifying Articrafts to be Configured.
- Naming Conventions and Version Control.
- Configuration Control.
- Quality Assurance Techniques
- Peer Reviews.
- Fegan Inspection.
- Unit, Registration, System, and Acceptance Testing
- Test data and Test cases
- Bug Tracking
- Casual Analysis

Identifying Articrafts to be Configured

The artifacts that make up a typical Endeca application definition include the following:

The AppConfig.xml file

The AppConfig.xml file describes each of the application's provisioning information and is stored in the EAC Central Server. The Deployment Template control scripts use AppConfig.xml as the authoritative source for application definition. The Deployment Template stores a copy of the AppConfig.xml file in the [appdir]/config/script directory.

The instance configuration

The instance configuration is a set of files that control the ITL process and the data loaded into the MDEX Engine servers. The instance configuration files are controlled by the Developer Studio, and optionally by the Workbench.

These files include configuration data such as dimension definition, search configuration, the Forge pipeline, and Page Builder landing pages.

Identifying Articrafts to be Configured

Page Builder templates

Page Builder templates are used to drive dynamic landing pages that can be created in Page Builder. They are defined by xml files stored by the Workbench, and accessed through the emgr_update command utility.

Command-line scripts

An application deployment typically includes command-line scripts that perform common operations related to the application's functionality. By convention, these scripts are stored in the Deployment Template's [appdir]/control directory.

Library files

Many parts of an application use library files. For example BeanShell scripts may use application- or Endeca-specific Java classes. By convention, library files are kept under [appdir]/config/lib.

Forge state files

Forge state files reside in the [appdir]/data/state directory.

Naming Conventions and Version Control

- This process describes the deliverable types, naming conventions and version control mechanisms to be applied to deliverables produce by the project.
- The following table describes the configurable item types used within the IFS project

Naming Conventions and Version Control

Business Area: 5 characters (e.g. HR or CeDICT)

Project: 4-6 characters (e.g. Tavern, OHSW)

Deliverable Type :2 or 3 characters

UT = Unit Task Specification

PM = Project Management deliverable

TPR = Test Plan & Result

CR = Change Request

PF = Process Flow

OC = Organisation chart

RR = Resource Request

PR = Presentations

PS = Project Standard

MN = Minutes

AD = Architecture Deliverable

AF = Acceptance Form

DR = Deliverable Review Form

DI = Diagram

ST = Strategy document

Description: Brief description of deliverable

Version : character followed by major and minor numbering,

OR

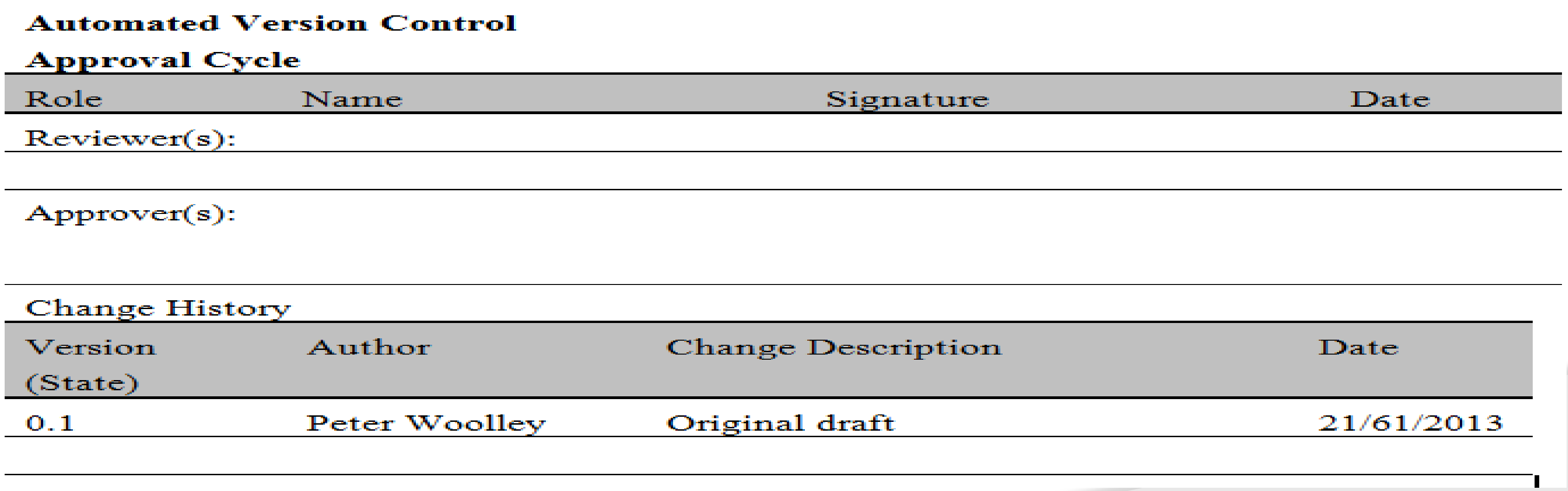
Date in yymmdd format. This version format is used for deliverables that are to be kept at a point in time.

File Type : as per application

Naming Conventions and Version Control

Manual Version Control

If the deliverable version is controlled by the date in the deliverable file name then for each new version of the deliverable the current date is used in the file name. If there are multiple version created for the same deliverable on the same day then an alphabetic character is appended to the date starting at ‘a’.



Automated Version Control

Approval Cycle

Role	Name	Signature	Date
Reviewer(s):			
Approver(s):			

Change History

Version (State)	Author	Change Description	Date
0.1	Peter Woolley	Original draft	21/61/2013

Configuration Control

- Configuration control is an important function of the configuration management discipline. Its purpose is to ensure that all changes to a complex system are performed with the knowledge and consent of management. The scope creep that results from ineffective or nonexistent configuration control is a frequent cause of project failure.
- Configuration control tasks include initiating, preparing, analysing, evaluating and authorising proposals for change to a system (often referred to as " the configuration ").

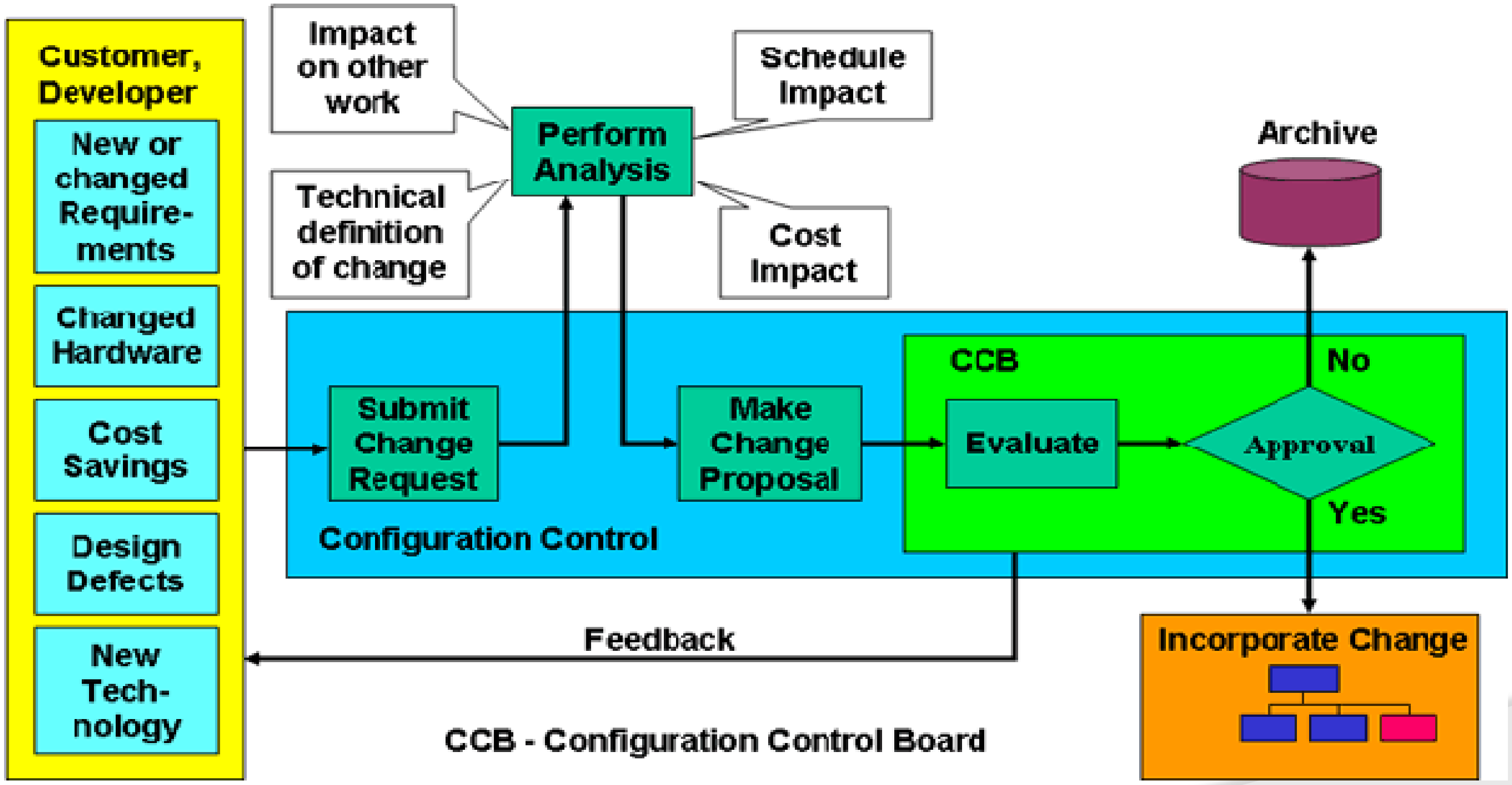
Configuration Control

Configuration control has four main processes:

1. Identification and documentation of the need for a change in a change request
2. Analysis and evaluation of a change request and production of a change proposal
3. Approval or disapproval of a change proposal
4. Verification, implementation and release of a change.

Configuration Control

The Configuration Control Process



Quality Assurance Techniques

Various SQA Techniques include:

Auditing: Auditing involves inspection of the work products and its related information to determine if the set of standard processes were followed or not.

Reviewing: A meeting in which the software product is examined by both the internal and external stakeholders to seek their comments and approval.

Code Inspection: It is the most formal kind of review that does static testing to find bugs and avoid defect growth in the later stages. It is done by a trained mediator/peer and is based on rules, checklist, entry and exit criteria. The reviewer should not be the author of the code.

Design Inspection:

Quality Assurance Techniques

Design inspection is done using a checklist that inspects the below areas of software design:

General requirements and design
Functional and Interface specifications
Conventions
Requirement traceability
Structures and interfaces
Logic
Performance
Error handling and recovery
Testability, extensibility
Coupling and cohesion

Quality Assurance Techniques

Simulation: Simulation is a tool that models the real-life situation in order to virtually examine the behavior of the system under study.

Functional Testing: It is a QA technique which verifies what the system does without considering how it does. This type of black box testing mainly focuses on testing the system specifications or features.

Standardization: Standardization plays a crucial role in quality assurance. It decreases the ambiguity and guesswork, thus ensuring quality.

Static Analysis: It is a software analysis that is done by an automated tool without actually executing the program. This technique is highly used for quality assurance in medical, nuclear and aviation software. Software metrics and reverse engineering are some popular forms of static analysis.

Quality Assurance Techniques

Walkthroughs: Software walkthrough or code walkthrough is a kind of peer review where the developer guides the members of the development team to go through the product and raise queries, suggest alternatives, make comments regarding possible errors, standard violations or any other issues.

Path Testing: It is a white box technique where the complete branch coverage is ensured by executing each independent path at least once.

Stress Testing: This type of testing is done to check how robust a system is by testing it under heavy load i.e. beyond normal conditions.

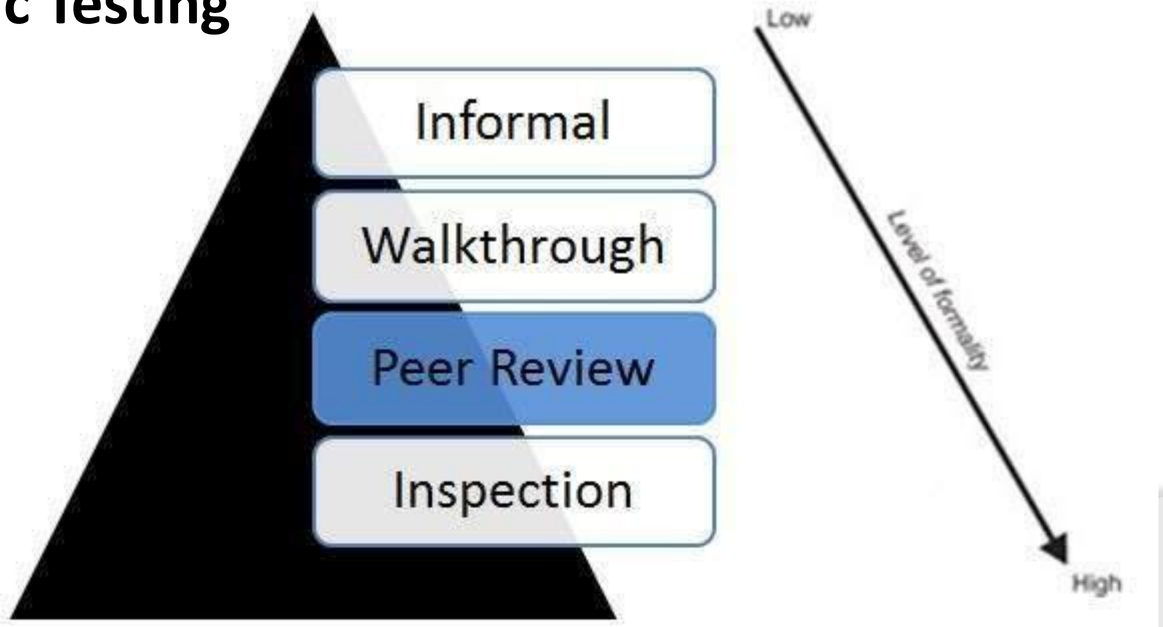
Six Sigma: Six Sigma is a quality assurance approach that aims at nearly perfect products or services. It is widely applied in many fields including software. The main objective of six sigma is process improvement so that the produced software is 99.76 % defect free.

Peer Reviews

What is Peer Review?

A peer review, a review technique, which is a static white-box testing which are conducted to spot the defects early in the life cycle that cannot be detected by black box testing techniques.

Peer Review - Static Testing



Peer Reviews

Peer Review Characteristics:

•Peer Reviews are documented and uses a defect detection process that has peers and technical specialist as part of the review process.

•The Review process doesn't involve management participation.

•It is usually led by trained moderator who is NOT the author.

•The report is prepared with the list of issues that needs to be addressed.

Types of Software Reviews

There are various types of review based on the degree of formality:

Buddy checking: consist of having a person other than the author informally review a document or work. In general this doesn’t involve the use of checklists to guide the process and as such is not repeatable.

Peer Reviews

Walkthrough: involve the author of an artifact presenting it to an audience of their peers, and receiving comments and feedbacks. Usually, such process involves limited documentation of the process and the issues uncovered, which makes defect tracking difficult.

Peer Reviews

Review by circulation: consist of circulating an artifact among peers for comments; operates like a walkthrough but without holding a meeting. Not holding a meeting avoids potential arguments over issues, but also the benefits of discussion.

Inspection: formal and managed peer review process with the following characteristics:

-The process is carried out by a review team with clearly defined roles.

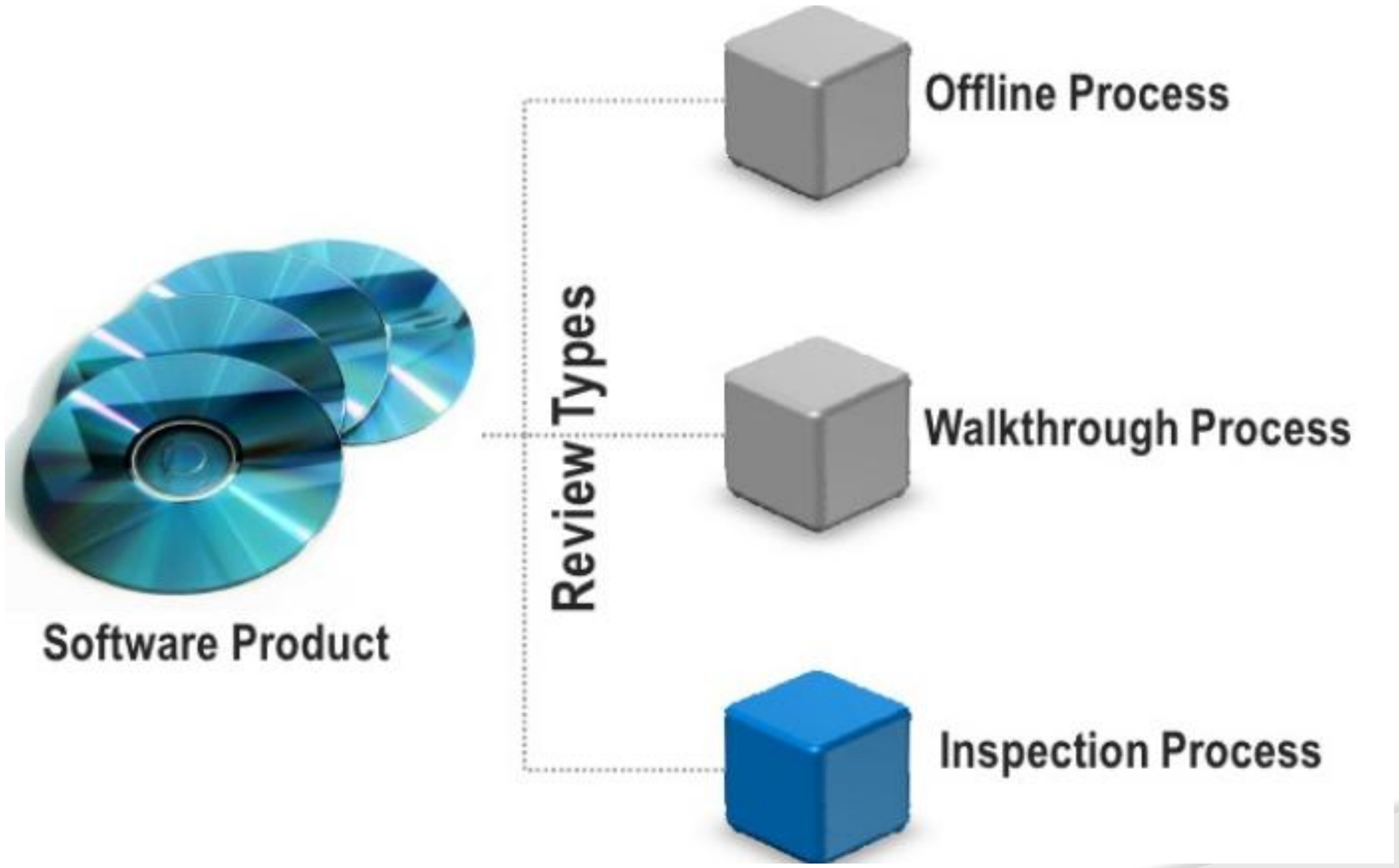
-The process follows an unambiguous set of criteria for each type of artifacts.

-Specific data are collected during the process.

-The process is driven by quantitative goals such as process and quality improvements

Fegan inspection

Review Process Types



Fegan inspection

Why Fagan Inspection?

- Very high defect detection effectiveness – **the single most effective software quality control**
- Widely applicable – to all documents – developed, changed or acquired
- Can be used early in development (to requirements. docs.) and throughout development work
- Well documented track record of reduction in rework costs and defects in released software

Fagan inspection

Fagan's Six Major Steps

1. Planning
2. Overview
3. Preparation
4. Examination
5. Rework
6. Follow-up

1. Planning: Form team, assign roles
2. Overview: Inform team about product (optional)
3. Preparation: Independent review of materials
4. Examination: Inspection meeting
5. Rework: Author verify defects and correct
6. Follow-up: Moderator checks and verifies corrections

Unit , Integration , System , and Acceptance Testing

Unit testing,

It is also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level

Integration testing

It is any type of software testing that seeks to verify theinterfaces between components against a software design.

System testing,

end-to-end testing, tests a completely integratedsystem to verify that it meets its requirements. for example, a system testmight involve testing a logon interface, then creating and editing anentry, plus sending or printing results, followed by summary processingor deletion (or archiving) of entries, then logoff

Acceptance testing

At last the system is delivered to the user for Acceptance testing.

Test data and Test cases

Test plan

A test specification is called a test plan. The developers are well aware what test plans will be executed and this information is made available to management and the developers.

Test case

A test case normally consists of a unique identifier, requirement references from a design specification, preconditions, events, a series of steps (also known as actions) to follow, input, output, expected result, and actual result. Clinically defined a test case is an input and an expected result

Test data and Test cases

Test script

A test script is a procedure, or programming code that replicates user actions. Test Case will be a baseline to create test scripts using a tool or a program.

Test suite

The most common term for a collection of test cases is a test suite. The test suite often also contains more detailed instructions or goals for each collection of test cases

Test data

In most cases, multiple sets of values or data are used to test the same functionality of a particular feature. All the test values and changeable environmental components are collected in separate files and stored as test data. It is also useful to provide this data to the client and with the product or a project.

Bug Tracking

•Bug tracking systems as a part of integrated project management system Bug and issue tracking systems are often implemented as a part of integrated project management systems. This approach allows including bug tracking and fixing in a general product development process, fixing bugs in several product versions, automatic generation of a product knowledge base and release notes.

Bug tracking and test management

While traditional test management tools such as HP Quality Centre and IBM Rational Quality Manager come with their own bug tracking systems, other tools integrate with popular bug tracking systems.

Casual Analysis

Causal analysis and resolution improves quality and productivity by preventing the introduction of defects or problems and by identifying and appropriately incorporating the causes of superior process performance.

The Causal analysis Resolution process area involves the following activities:

Identifying and analyzing causes of selected outcomes. The selected outcomes can represent defects and problems that can be prevented from happening in the future or successes that can be implemented in projects or the organization.

1. Remove causes and prevent the recurrence of those types of defects and problems in the future
2. Proactively analyze data to identify potential problems and prevent them from occurring
3. Incorporate the causes of successes into the process to improve future process performance

UNIT-V

Process elements, process architecture, relationship between elements, process modeling, process definition techniques, ETVX (Entry-Task-Validation-exit), process base lining, process assessment and improvement, CMMI, six sigma

Course Learning Outcomes

The course will enable the students to:	
CLO1	Use Process elements, process architecture.
CLO 2	Usage of Process relationship between elements, process modeling.
CLO 3	Use of the process definition techniques ETVX, CMMI, six sigma.

Contents

- Process elements.
- Process architecture.
- Relationship between elements.
- Process modeling
- Process definition techniques.
- ETVX (Entry-Task-Validation-exit).
- Process baselining
- Process assessment and improvement
- CMMI
- Six sigma

Process Elements, Relationship between elements

- Process Element (PE) is a group of project activities, and/or other process elements related by logical dependencies, which when executed (or enacted) provides value to the project”.
- Process Elements, like software components, have input and output interfaces, defined by pre-conditions and post conditions. Within a process element, the project activities and sub-process elements are classified as variants and invariants.
- Additionally, PE’s incorporate feedback from a knowledge base that has information on past project plans.

Process Elements, Relationship between elements

Figure 1 shows the overall PE interactions.

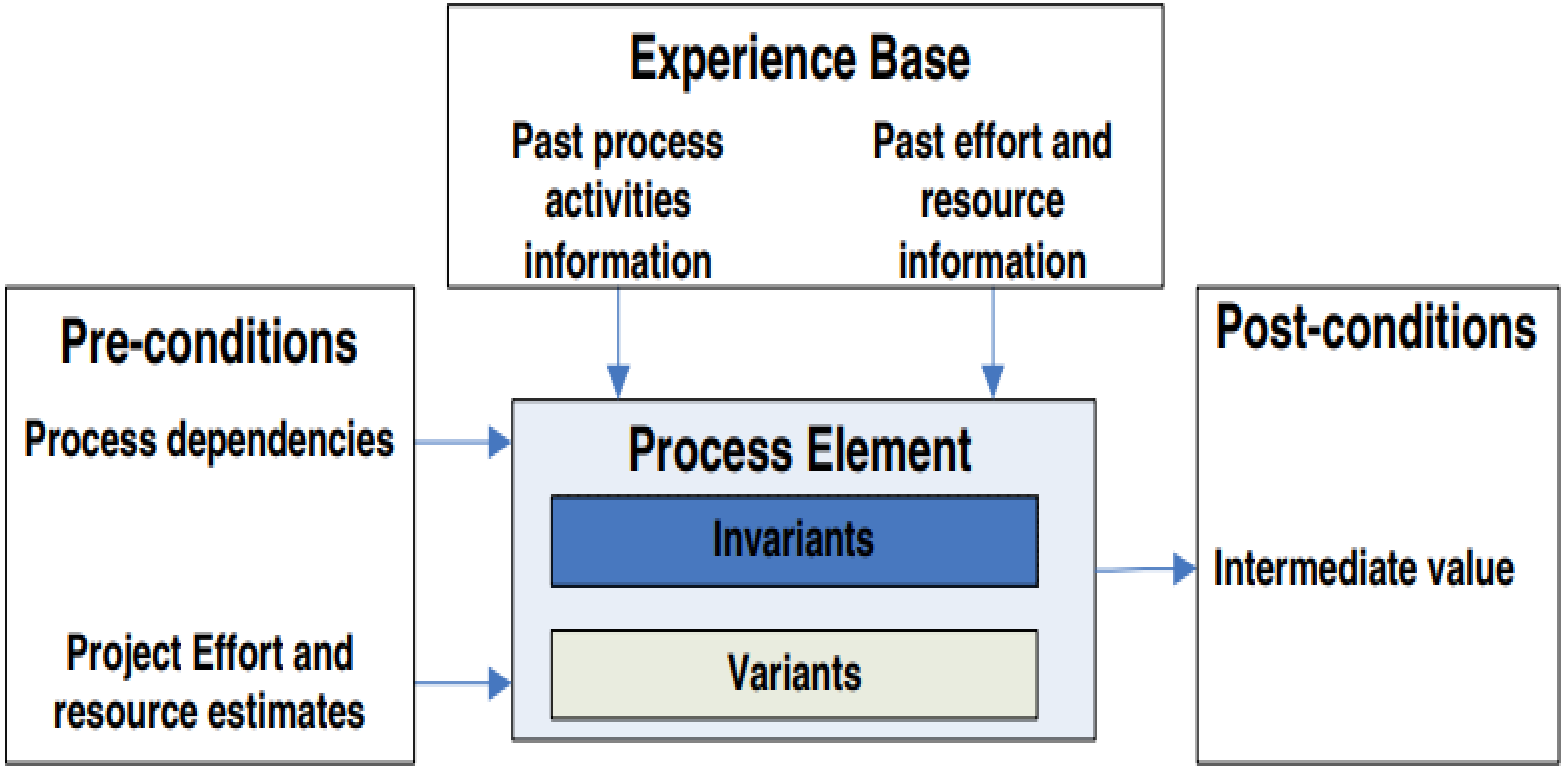


Fig. 1. Process Element Interactions

Process Elements, Relationship between elements

Pre-Conditions The pre-conditions of a PE include its dependencies, i.e. artifacts and information that it will need for execution and the project effort, schedule and/or resource estimates. The dependencies of a PE usually include the results provided by predecessing PEs. The effort, schedule and resource estimates are used to determine the effort and resources of the activities and sub-process elements within the PE.

Post-Conditions The post-conditions define the results obtained from executing a PE. These may be in the form of abstract models, information and analysis, or simply risk reduction. Results from one PE usually form the input or dependencies of another PE.

Process Elements, Relationship between elements

Experience Base Process elements should take into consideration the implications of past projects from both within and outside the organization. Incorporating past project schedule, effort, and relationship information within the process elements will produce empirically proven project plans. Additionally, they should also include organizational and industry best practices. The past effort information for the process element can be used to determine the percent of overall project effort and resources required to implement the process element.

Process Elements, Relationship between elements

- **Process Element** the process element consists of a set of activities and/or sub process elements. Both activities and sub process elements are classified as variant or invariant.
- Unlike software components, which are usually treated as a black box, process elements will most likely undergo some sort of customization to meet the objectives of the project at hand.
- Variant activities are those that can be modified, and if required even removed from the process element, without significantly impacting the results produced by the PE.
- Invariant activities on the other hand, may be tweaked but cannot be extensively modified, since upon such modification the PE can no longer guarantee to produce the initially promised results.

Process Architecture

A conceptual framework for consistently incorporating, relating, and tailoring process elements into enactable processes. A process architecture is often needed when a process must relate to other existing or future processes

Or

Process architecture ensures that processes to be redesigned or newly developed are meeting the organizations objectives and fit within the organization strategy

Process Architecture

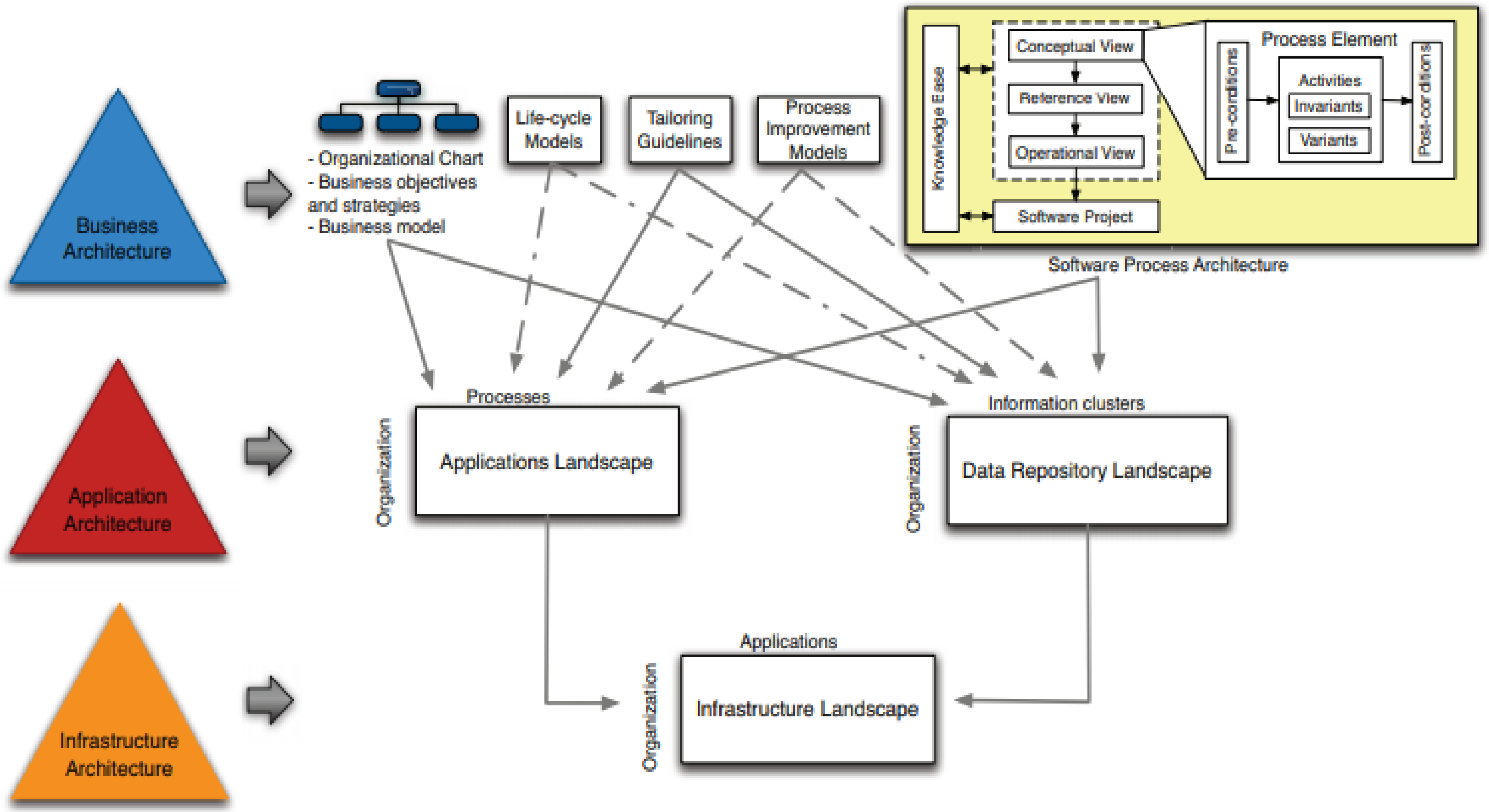


Figure 1. Blueprint to deploy a Software Process Architecture

Process Architecture

Business Architecture is composed of organizational architecture, business model, business objective and strategies, process improvement models, life-cycle models, tailoring guidelines and SPA. This SPA is divided in five views: 1) **Conceptual view** defines the structure of a process element like a set of activities (variants and invariants), pre-conditions (artifacts and information to execute process elements), postconditions (results obtained from executing process elements) 2) **Reference view** determines the process improvement models that will be used and which best practices. 3) **Operational view** defines the standard software processes. 4) **knowledge base** to share information of past projects and others process assets. 5) **Software project** is a set of processes tailored to a specific project. The elements of this architecture are related with the elements of application architecture.

Process Architecture

Application architecture is composed of two parts: 1) Applications landscape is supporting the automation of business processes. 2) Data Repository landscape is the physical storage of all relevant company data. The elements of this architecture are related with the elements of infrastructure architecture.

Infrastructure architecture is referred to as technology architecture, comprises the software, hardware and network infrastructure required for operations of all applications.

Process Modeling

- Process modeling is the act of process model development. With a process model defined as: A process model is an abstract, formal representation of the real- world process.
- The purpose of a model is to reduce the complexity of understanding or interacting with a phenomenon by eliminating the detail that does not influence its relevant behavior". Thus process models are abstractions, removing unwanted complexity. We also require process models to be formal, and process models are therefore built from a limited set of concepts, using certain rules

Process Modeling

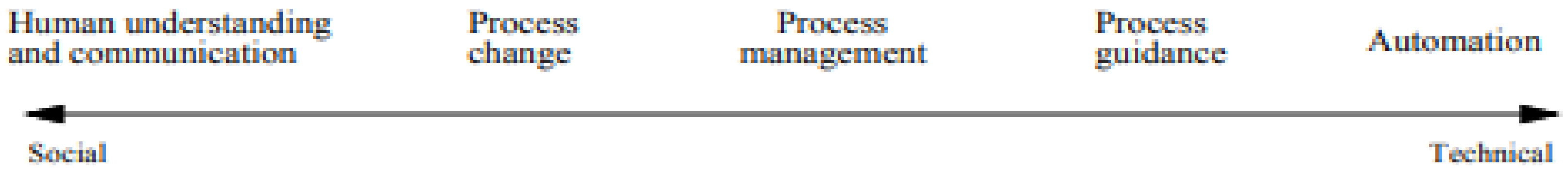


Figure 3.2: Purposes of process modeling.

Process Modeling

Five basic uses of process models

Facilitate human understanding and communication. Process models can be used to increase the knowledge about processes in the organization Support process change. Models can be used to manage change of the process

Support process management: Process models can be used to manage the process. The models are a basis for measurements, and can be used to compare actual performance with plans and ideal model

Automate process guidance: This is an extension of the previous activity. The models are used to guide the process execution, including some sort of computer-aided guidance. This approach includes some prescriptive elements

Automate execution support: This approach relies on process models to automate the menial tasks of the organization. Must be combined with human guidance to be generally useful.

Process Definition Techniques

A software process (also knows as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system. Any software process must include the following four activities:

Software specification (or requirements engineering):

Define the main functionalities of the software and the constrains around them

Software design and implementation:

The software is to be designed and programmed.

Process Definition Techniques

A software process (also knows as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

Any software process must include the following four activities:

Software specification (or requirements engineering):

Define the main functionalities of the software and the constrains around them

Software design and implementation:

The software is to be designed and programmed.

Process Definition Techniques

Software verification and validation: The software must conforms to it’s specification and meets the customer needs. **Software evolution (software maintenance**): The software is being modified to meet customer and market requirements changes.

In practice, they include sub-activities such as requirements validation, architectural design, unit testing, …etc. There are also supporting activities such as configuration and change management, quality assurance, project management, user experience.

Process Definition Techniques

- Along with other activities aim to improve the above activities by introducing new techniques, tools, following the best practice, process standardization (so the diversity of software processes is reduced), etc.
- When we talk about a process, we usually talk about the activities in it. However, a process also includes the process description, which includes:

Process Definition Techniques

Products:
The outcomes of the an activity. For example, the outcome of architectural design maybe a model for the software architecture.
Roles:
The responsibilities of the people involved in the process. For example, the project manager, programmer, etc.
Pre and post conditions:
The conditions that must be true before and after an activity. For example, the pre condition of the architectural design is the requirements have been approved by the customer, while the post condition is the diagrams describing the architectural have been reviewed.

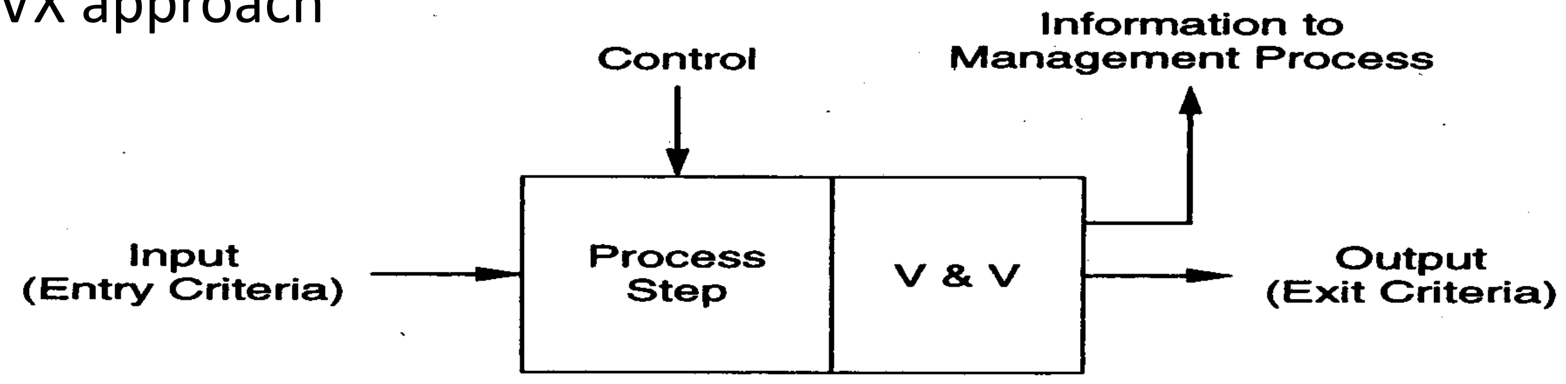
ETVX

ETVX approach to specify a phase

- Entry criteria: what conditions must be satisfied for initiating this phase
- Task: what is to be done in this phase
- Verification: the checks done on the outputs of this phase
- eXit criteria: when can this phase be considered done successfully

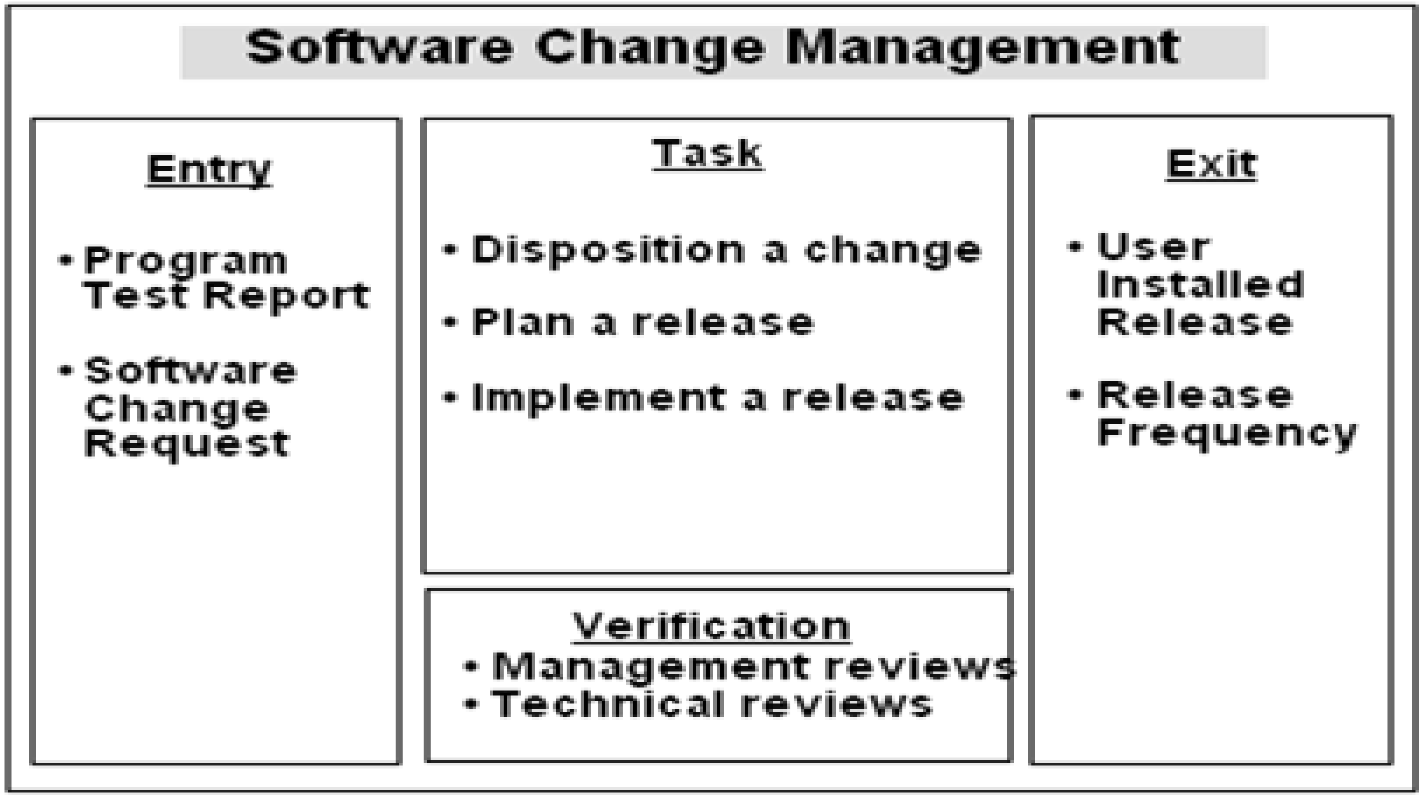
A phase also produces info for mgmt

ETVX approach



ETVX

ETVX Example: Change management



Software Change Management

Entry

- Program Test Report
- Software Change Request

Task

- Disposition a change
- Plan a release
- Implement a release

Verification

- Management reviews
- Technical reviews

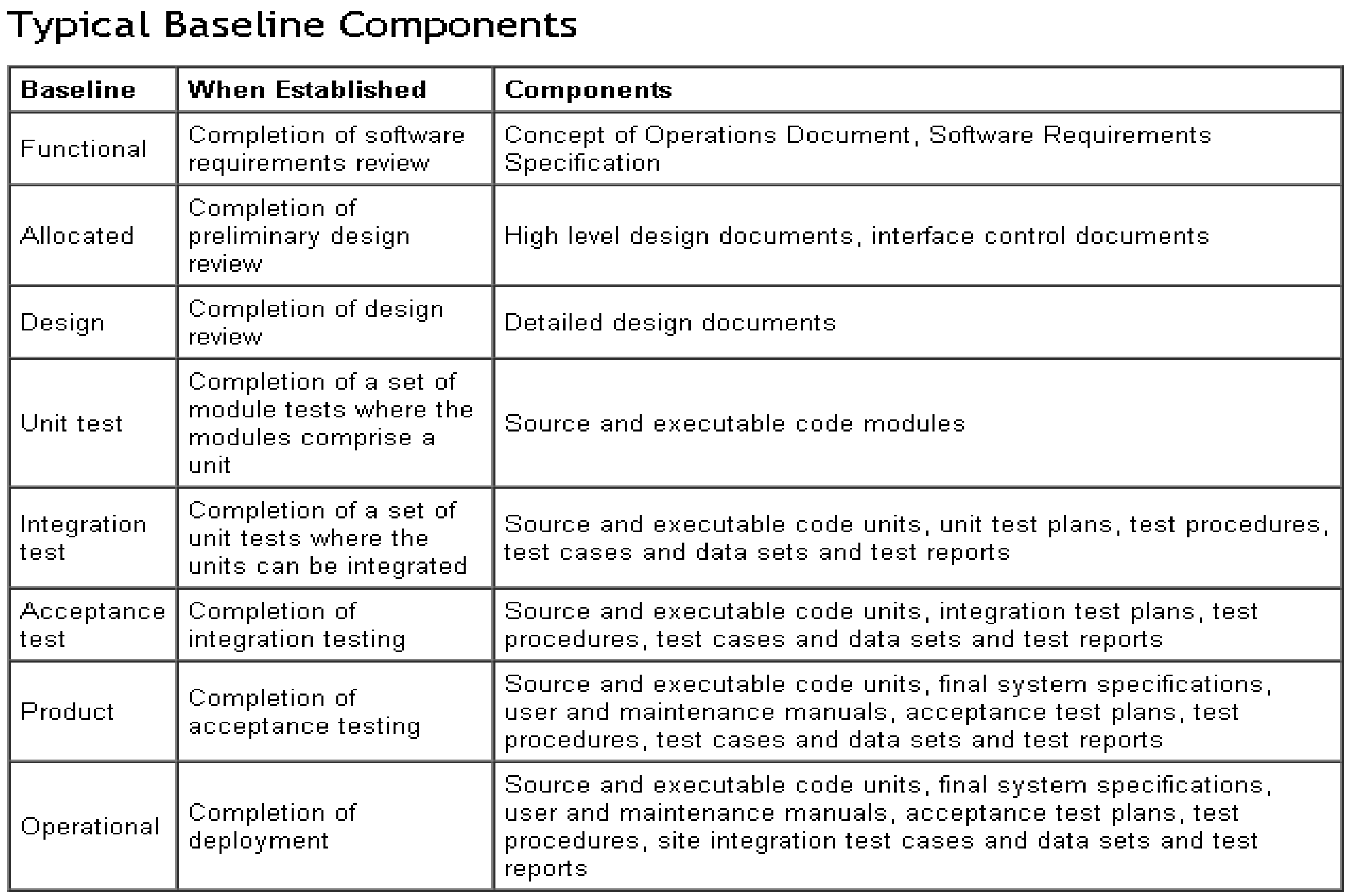
Exit

- User Installed Release
- Release Frequency

Process base lining

- A baseline is a reference point in the software development life cycle marked by the completion and formal approval of a set of predefined work products.
- The objective of a baseline is to reduce a project's vulnerability to uncontrolled change by fixing and formally change controlling various key deliverables (configuration items) at critical points in the development life cycle.
- Baselines are also used to identify the aggregate of software and hardware components that make up a specific release of a system.

Process base lining



Typical Baseline Components

Baseline	When Established	Components
Functional	Completion of software requirements review	Concept of Operations Document, Software Requirements Specification
Allocated	Completion of preliminary design review	High level design documents, interface control documents
Design	Completion of design review	Detailed design documents
Unit test	Completion of a set of module tests where the modules comprise a unit	Source and executable code modules
Integration test	Completion of a set of unit tests where the units can be integrated	Source and executable code units, unit test plans, test procedures, test cases and data sets and test reports
Acceptance test	Completion of integration testing	Source and executable code units, integration test plans, test procedures, test cases and data sets and test reports
Product	Completion of acceptance testing	Source and executable code units, final system specifications, user and maintenance manuals, acceptance test plans, test procedures, test cases and data sets and test reports
Operational	Completion of deployment	Source and executable code units, final system specifications, user and maintenance manuals, acceptance test plans, test procedures, site integration test cases and data sets and test reports

Process assessment and improvement

A number of different approaches to software process assessment and improvement have been proposed.

Standard CMMI Assessment Method for Process Improvement (SCAMPI)

provides a five-step process assessment model that incorporates five phases: initiating, diagnosing, establishing, acting, and learning. The SCAMPI method uses the SEI CMMI as the basis for assessment.

CMM-Based Appraisal for Internal Process Improvement (CBA IPI)

provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment.

SPICE (ISO/IEC15504)

a standard that defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process.

ISO 9001:2000 for Software

a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies.

CMMI

CMMI Staged Representation

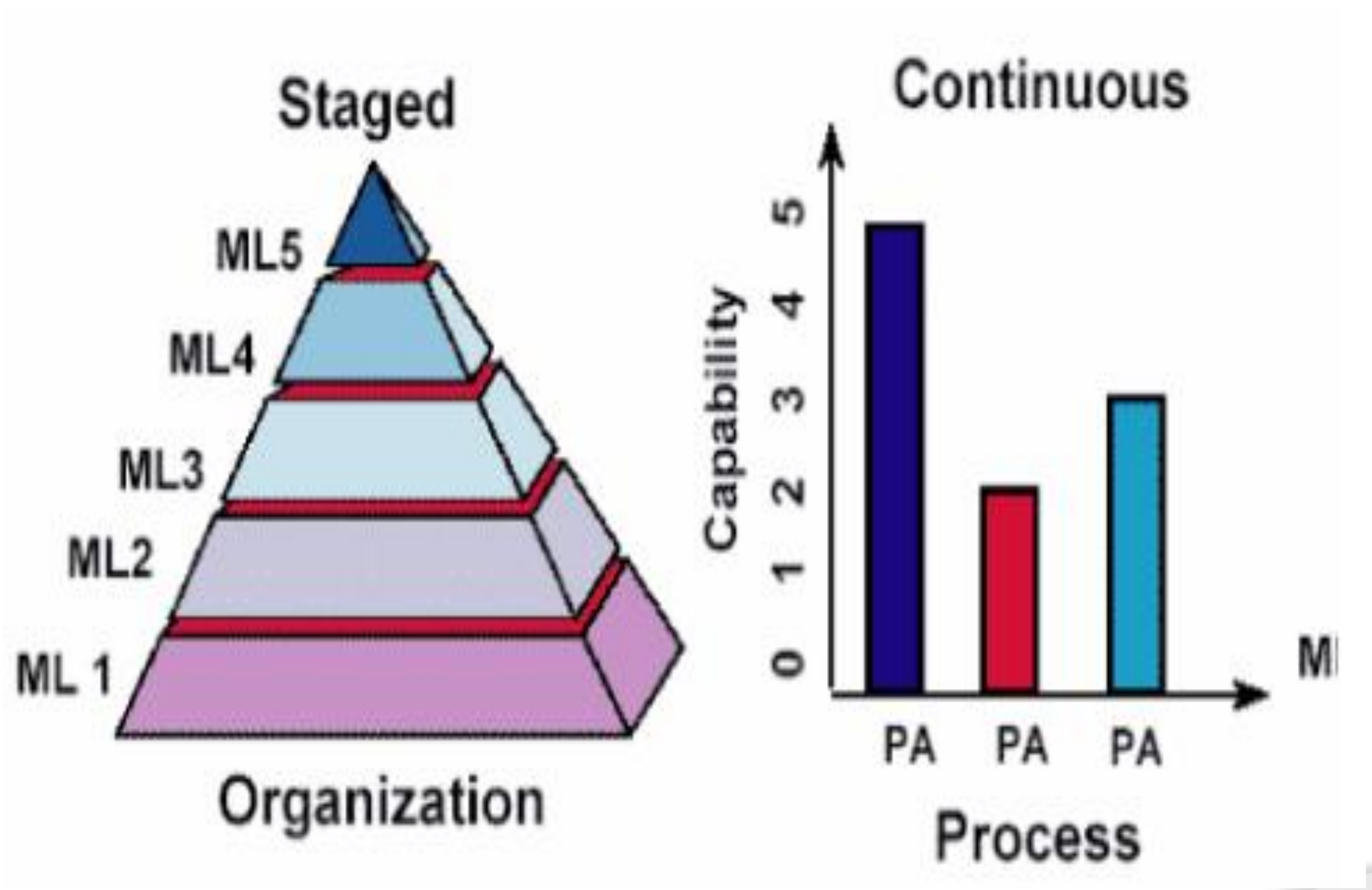
- Process areas are organized by maturity levels.
- Improvement is measured using maturity levels. Maturity levels measure the maturity of a set of processes across an organization: it ranges from 1 through 5.
- There is only one type of specific practice. The concepts of base and advanced practices are not used. All specific practices appear in the staged representation except when a related base-advanced pair of practices appears in the continuous representation, in which case only the advanced practice appears in the staged representation.
- Common features are used to organize generic practices.
- Only the level 2 and level 3 generic practices are included.
- There is no need for an equivalence mechanism to back the continuous representation because each organization can choose what to improve and how much to improve using the staged representation

CMMI

CMMI Continuous Representation

- Process areas are organized by process area categories.
- Improvement is measured using capability levels. Capability levels measure the maturity of a particular process across an organization; it ranges from 0 through 5.
- There are two types of specific practices: base and advanced. All specific practices appear in the continuous representation.
- Capability levels are used to organize the generic practices.
- All generic practices are included in each process area.
- Equivalent staging allows determination of a maturity level from an organization's achievement profile.

CMMI


Staged
ML5
ML4
ML3
ML2
ML 1
Organization
Continuous
Capability
0
1
2
3
4
5
MI
PA
PA
PA
Process

CMMI

In CMMI models with a continuous representation, there are six capability levels designated by the numbers 0 through 5.

0 − Incomplete

1 − Performed

2 − Managed

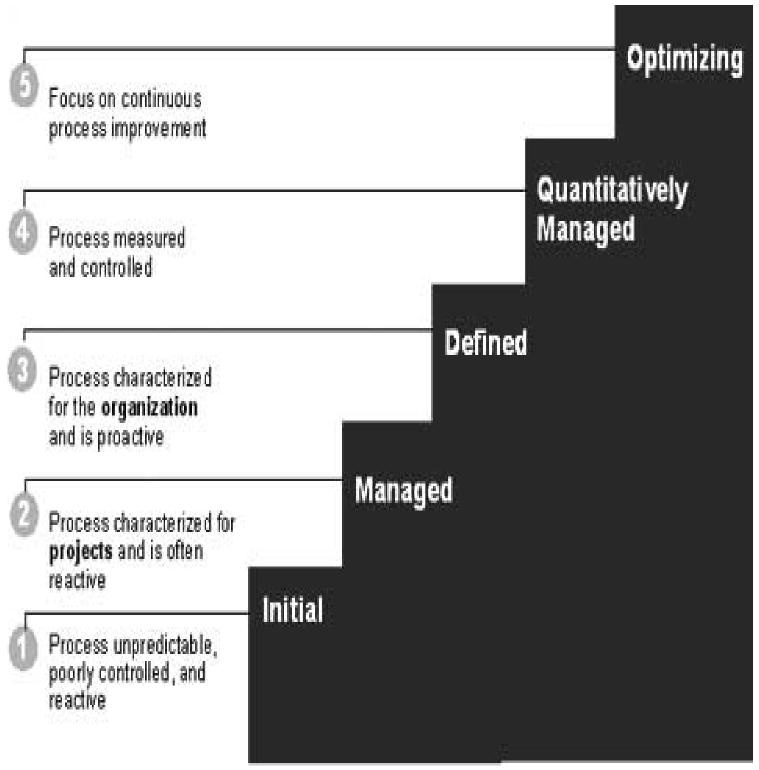
3 − Defined

4 − Quantitatively Managed

5 − Optimizing

CMMI Staged Representation Maturity Levels

The following image shows the maturity levels in a CMMI staged representation


Optimizing
5 Focus on continuous process improvement
Quantitatively Managed
4 Process measured and controlled
Defined
3 Process characterized for the organization and is proactive
Managed
2 Process characterized for projects and is often reactive
Initial
1 Process unpredictable, poorly controlled, and reactive

Six sigma

- •The purpose of Six Sigma is to improve processes to do things better, faster, and at lower cost.
- •It can be used to improve every facet of business, from production, to human resources, to order entry, to technical support.
- •Six Sigma can be used for any activity that is concerned with cost, timeliness, and quality of results.

Sub-methodologies: DMAIC and DMADV.

- • The Six Sigma DMAIC process (defines, measure, analyze, improve, control) is an improvement system for existing processes failing below specification and looking for incremental improvement.
- •The Six Sigma DMADV process (define, measure, analyze, design, verify) is an improvement system used to develop new processes or products at Six Sigma quality levels.

Six sigma

DMADV Methodology:

Define --> Measure --> Analyze --> Design -->Verify

Define : Define the Problem or Project Goals that needs to be addressed.

Measure: Measure and determine customers needs and specifications.

Analyze: Analyze the process for meet the customer needs.

Design: Design a process that will meet customers needs.

Verify: Verify the design performance and ability to meet customer needs

Six sigma

DFSS Methodology

Define --> Identify --> Design --> Optimize -->Verify

Define :

Identify the Customer and project.

Identify:

Define what the customers want, or what they do not want.

Design:

Design a process that will meet customers needs.

Optimize:

Determine process capability & optimize design.

Verify:

Test, verify, & validate design.

THANK YOU