INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad -500 043

INFORMATION TECHNOLOGY

COURSE LECTURE NOTES

Course Name	**SOFTWARE PROCESS AND PROJECT MANAGEMENT**
Course Code	**AIT512**
Programme	B.Tech
Semester	VI
Course Coordinator	Mr. E Sunil Reddy, Assistant Professor
Course Faculty	Mr. E Sunil Reddy, Assistant Professor
Lecture Numbers	1-45
Topic Covered	All

COURSE OBJECTIVES:

The course should enable the students to:	
I	Understand overall software development life cycle and adopt suitable processes.
II	Analyze, prioritize, and manage both functional and quality requirements.
III	Estimate efforts required, plan, and track the plans.
IV	Understand and apply configuration and quality management techniques.

COURSE LEARNING OUTCOMES (CLOs):

Students, who complete the course, will have demonstrated the ability to do the following:

CLO Code	**CLO's**	**At the end of the course, the student will have the ability to:**	**PO's Mapped**	**Strength of Mapping**
AIT512.01	CLO 1	Describe the basic concepts of Software Development Life Cycle.	PO 1	1
AIT512.02	CLO 2	Summarize the concept of processes.	PO 2	2
AIT512.03	CLO 3	Analyze the concepts of Personal Software Process (PSP), Team Software Process (TSP).	PO 11	3
AIT512.04	CLO 4	Use the concept of agile processes in real-world problems.	PO 11	3
AIT512.05	CLO 5	Determine the Functional requirements and quality attributes.	PO 2	2

CLO Code	CLO's	At the end of the course, the student will have the ability to:	PO's Mapped	Strength of Mapping
AIT512.06	CLO 6	Understand elicitation techniques, Quality Attribute Workshop (QAW).	PO 2	2
AIT512.07	CLO 7	Determine the analysis, prioritization, and trade off	PO 2	2
AIT512.08	CLO 8	Use Architecture Centric Development Method (ACDM).	PO 2	2
AIT512.09	CLO 9	Illustrate the documentation, and specification.	PO 9	2
AIT512.10	CLO 10	Describe the change management and traceability of requirements.	PO 9	2
AIT512.11	CLO 11	Explain software risks.	PO 2	2
AIT512.12	CLO 12	Understand the concept of function points, COCOMO II, estimations	PO 2	2
AIT512.13	CLO 13	Understand theWork break down structure, macro and micro plans	PO 9	2
AIT512.14	CLO 14	Understand theplanning poker ,wideband Delphi	PO 1	1
AIT512.15	CLO 15	Summarize the tracking the plan ,Earned Value Method (EVM)	PO 1	1
AIT512.16	CLO 16	Identifying articrafts to be configured, naming conventions	PO 1, PO 2	2
AIT512.17	CLO 17	Understand the version control, configuration control, quality assurance techniques.	PO 1, PO 2	2
AIT512.18	CLO 18	Summarize the concept of peer reviews, Fagan inspection	PO 1, PO 2	2
AIT512.19	CLO 19	Apply testing of unit, registration, system, and acceptance, test data and test cases..	PO 11, PO 9	3
AIT512.20	CLO 20	Understand the bug tracking, casual analysis.	PO 11, PO 9	3
AIT512.21	CLO 21	Use Process elements, process architecture.	PO 11	3
AIT512.22	CLO 22	Usage ofProcess relationship between elements, process modeling.	PO 11	3
AIT512.23	CLO 23	Use of the process definition techniques ETVX, CMMI, six sigma.	PO 11	3

SYLLABUS

Unit-I	DEVELOPMENT LIFE CYCLE PROCESSES
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.	
Unit -II	REQUIREMENTS MANAGEMENT
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.	
Unit -III	ESTIMATION, PLANNING, AND TRACKING
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).	
Unit-IV	CONFIGURATION AND QUALITY MANAGEMENT
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	

Unit-V	**SOFTWARE PROCESS DEFINITION AND MANAGEMENT**
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.	
Text Books:	
1. PankajJalote, "Software Process Management in Practice", Pearson, Illustrated, 2002. 2. Walker Royce, "Software Project Management – A Unified Framework", Pearson Education, 1st Edition, 2002.	
Reference Books:	
1. Watts S.Humphrey, "PSP: A Self Improvement Process for Software Engineers", Addison Wesley, 1st Edition, 2005. 2. Chris F. Kemerer, "Software Project Management- Readings and Cases", McGraw-Hill, Illustrated 2nd Edition, 1997. 3. Watts S. Humphrey, "Introduction to the Team Software Process", Addison-Wesley, Illustrated Reprint, 2000	

Module – I

Development Life Cycle Processes

Overview of software development life cycle

There are various software development approaches defined and designed which are used/employed during development process of software, these approaches are also referred as "Software Development Process Models" (e.g. Waterfall model, incremental model, V-model, iterative model, etc.). Each process model follows a particular life cycle in order to ensure success in process of software development.

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced according to the design which is called development phase. After coding and development the testing verifies the deliverable of the implementation phase against requirements.

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

1. Requirement gathering and analysis: Business requirements are gathered in this phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements like; Who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirements gathering phase. After requirement gathering these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied.
2. Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.
3. Design: In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design

specifications serve as input for the next phase of the model.

4. Implementation / Coding: On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.
5. Testing: After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase unit testing, integration testing, system testing, acceptance testing are done.
6. Deployment: After successful testing the product is delivered / deployed to the customer for their use.
7. Maintenance: Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

Introduction to Process

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.

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.

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.

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)

The PSP aims to provide software engineers with disciplined methods for improving personal software development processes. The PSP helps software engineers to:

- Improve their estimating and planning skills.
- Make commitments they can keep.
- Manage the quality of their projects.
- Reduce the number of defects in their work.

The goal of the PSP is to help developers produce zero-defect, quality products on schedule. Low-defect and zero defect products have become the reality for some developers and TSP teams, such as the Motorola division in Florida that achieved zero defects in over 18 projects through implementing PSP techniques.

PSP structure

PSP training follows an evolutionary improvement approach: an engineer learning to integrate the PSP into his or her process begins at the first level - PSP0 - and progresses in process maturity to thefinal level - PSP2.1. Each Level has detailed scripts, checklists and templates to guide the engineer through required steps and helps the engineer improve his own personal software process. Humphrey encourages proficient engineers to customise these scripts and templates as they gain an understanding of their own strengths and weaknesses.

Process

The input to PSP is the requirements; requirements document is completed and delivered to the engineer.

PSP0, PSP0.1 (Introduces process discipline and measurement)

PSP0 has 3 phases: planning, development (design, coding, test) and a post mortem. A baseline is established of current process measuring: time spent on programming, faults injected/removed, size of a program. In a post mortem, the engineer ensures all data for the projects has been properly recorded and analysed. PSP0.1 advances the process by adding a coding standard, a size measurement and the development of a personal process improvement plan (PIP). In the PIP, the engineer records ideas for improving his own process.

PSP1, PSP1.1 (Introduces estimating and planning)

Based upon the baseline data collected in PSP0 and PSP0.1, the engineer estimates how large a new program will be and prepares a test report (PSP1). Accumulated data from previous projects is used to estimate the total time. Each new project will record the actual time spent. This information is used for task and schedule planning and estimation (PSP1.1).

PSP2, PSP2.1 (Introduces quality management and design)

PSP2 adds two new phases: design review and code review. Defect prevention and removal are the focus at the PSP2. Engineers learn to evaluate and improve their process by measuring how long tasks take and the number of defects they inject and remove in each phase of development. Engineers construct and use checklists for design and code reviews. PSP2.1 introduces design specification and analysis techniques

(PSP3 is a legacy level that has been superseded by TSP.)

One of the core aspects of the PSP is using historical data to analyse and improve process performance. PSP data collection is supported by four main elements:

- Scripts
- Measures
- Standards
- Forms

The PSP scripts provide expert-level guidance to following the process steps and they provide a framework for applying the PSP measures. The PSP has four core measures:

- Size – the size measure for a product part, such as lines of code (LOC).
- Effort – the time required to complete a task, usually recorded in minutes.
- Quality – the number of defects in the product.
- Schedule – a measure of project progression, tracked against planned and actual completion dates.

Applying standards to the process can ensure the data is precise and consistent. Data is logged in forms, normally using a PSP software tool. The SEI has developed a PSP tool and there are also open source options available, such as Process Dashboard.

The key data collected in the PSP tool are time, defect, and size data – the time spent in each phase; when and where defects were injected, found, and fixed; and the size of the product parts. Software developers use many other measures that are derived from these three basic measures to understand and improve their performance. Derived measures include:

- estimation accuracy (size/time)
- prediction intervals (size/time)
- time in phase distribution
- defect injection distribution
- defect removal distribution
- productivity
- reuse percentage
- cost performance index

- planned value
- earned value
- predicted earned value
- defect density
- defect density by phase
- defect removal rate by phase
- defect removal leverage
- review rates
- process yield
- phase yield
- failure cost of quality (COQ)
- appraisal COQ
- appraisal/failure COQ ratio

Planning and tracking

Logging time, defect, and size data is an essential part of planning and tracking PSP projects, as historical data is used to improve estimating accuracy.

The PSP uses the PROxy-Based Estimation (PROBE) method to improve a developer's estimating skills for more accurate project planning. For project tracking, the PSP uses the earned value method.

The PSP also uses statistical techniques, such as correlation, linear regression, and standard deviation, to translate data into useful information for improving estimating, planning and quality. These statistical formulas are calculated by the PSP tool.

Using the PSP

The PSP is intended to help a developer improve their personal process; therefore PSP developers are expected to continue adapting the process to ensure it meets their personal needs.

Team software process(TSP)

In combination with the Personal Software Process (PSP), the Team Software Process (TSP) provides a defined operational process framework that is designed to help teams of managers and engineers organize projects and produce software products that range in size from small projects of several thousand lines of code (KLOC) to very large projects greater than half a million lines of code.The TSP is intended to improve the levels of quality and productivity of a team's software development project, in order to help them better meet the cost and schedule commitments of developing a software system.

How TSP Works

Before engineers can participate in the TSP, it is required that they have already learned about the PSP, so that the TSP can work effectively. Training is also required for other team members, the team lead, and management.

The TSP software development cycle begins with a planning process called the launch, led by a coach who has been specially trained, and is either certified or provisional. The launch is designed to begin the team building process, and during this time teams and managers establish goals, define team roles, assess risks, estimate effort, allocate tasks, and produce a team plan. During an execution phase, developers track planned and actual effort, schedule, and defects, meeting regularly (usually weekly) to report status and revise plans. A development cycle ends with a Post Mortem to assess performance, revise planning parameters, and capture lessons learned for process improvement.

The coach role focuses on supporting the team and the individuals on the team as the process expert while being independent of direct project management responsibility. The team leader role is different from the coach role in that, team leaders are responsible to management for products and project outcomes while the coach is responsible for developing individual and team performance.

Unifiedprocess

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 OpenUP and Agile Unified Process.

Overview

The Unified Process is not simply a process, but rather an extensible framework which should be customized for specific organizations or projects. The Rational Unified Process is, similarly, a customizable framework. As a result it is often impossible to say whether a refinement of the process was derived from UP or from RUP, and so the names tend to be used interchangeably.

Unified Process Characteristics

Iterative and Incremental

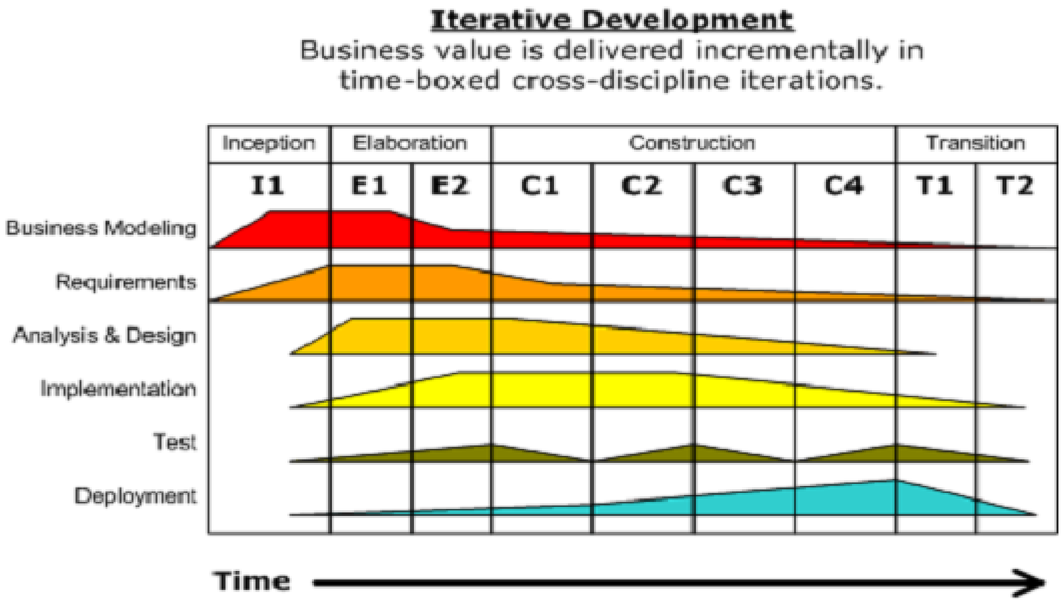


Diagram illustrating how the relative emphasis of different disciplines changes over the course of the projectThe Unified Process is an iterative and incremental development process. The Elaboration, Construction and Transition phases are divided into a series of timeboxed iterations. (The Inception phase may also be divided into iterations for a large project.) Each iteration results in an increment, which is a release of the system that contains added or improved functionality compared with the previous release.Although most iterations will include work in most of the process disciplines (e.g.Requirements, Design, Implementation, Testing) the relative effort and emphasis will change over the course of the project.

Use Case Driven

In the Unified Process, use cases are used to capture the functional requirements and to define the contents of the iterations. Each iteration takes a set of use cases or scenarios from requirements all the way through implementation, test and deployment

Architecture Centric

The Unified Process insists that architecture sit at the heart of the project team's efforts to shape the system. Since no single model is sufficient to cover all aspects of a system, the Unified Process supports multiple architectural models and views.

One of the most important deliverables of the process is the executable architecture baseline which is created during the Elaboration phase. This partial implementation of the system serves to validate the architecture and act as a foundation for remaining development.

Risk Focused

The Unified Process requires the project team to focus on addressing the most critical risks early in the project life cycle. The deliverables of each iteration, especially in the Elaboration phase, must be selected in order to ensure that the greatest risks are addressed first.

Project Lifecycle

The Unified Process divides the project into four phases:

Inception

- Elaboration
- Construction
- Transition

Inception Phase

Inception is the smallest phase in the project, and ideally it should be quite short. If the Inception Phase is long then it may be an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process.

The following are typical goals for the Inception phase.

- Establish a justification or business case for the project
- Establish the project scope and boundary conditions
- Outline the use cases and key requirements that will drive the design tradeoffs
- Outline one or more candidate architectures
- Identify risks
- Prepare a preliminary project schedule and cost estimate

The Lifecycle Objective Milestone marks the end of the Inception phase.

Develop an approximate vision of the system, make the business case, define the scope, and produce rough estimate for cost and schedule.

Elaboration Phase

During the Elaboration phase the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture. Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams).

The architecture is validated primarily through the implementation of an Executable Architecture Baseline. This is a partial implementation of the system which includes the core, most architecturally significant, components. It is built in a series of small, time boxed iterations. By the end of the Elaboration phase the system architecture must have stabilized and the executable architecture baseline must demonstrate that the architecture will support the key system functionality and exhibit the right behavior in terms of performance, scalability and cost.

The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the Construction phase. At this point the plan should be accurate and credible, since it should be based on the Elaboration phase experience and since significant risk factors should have been addressed during the Elaboration phase.

Construction Phase

Construction is the largest phase in the project. In this phase the remainder of the system is built on the foundation laid in Elaboration. System features are implemented in a series of short, timeboxed iterations. Each iteration results in an executable release of the software. It is customary to write full text use cases during the construction phase and each one becomes the start of a new iteration. Common UML (Unified Modelling Language) diagrams used during this phase include Activity, Sequence, Collaboration, State (Transition) and Interaction.

Transition Phase

The final project phase is Transition. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training.

Agile Processes

In software development life cycle, there are two main considerations, one is to emphasize on process and the other is the quality of the software and process itself. Agile software processes is an iterative and incremental baseddevelopment, where requirements are changeable according to customer needs. It helps in adaptive planning, iterative development and time boxing. It is a theoretical framework that promotes foreseen interactions throughout the development cycle. There are several SDLC models like spiral, waterfall, RAD which has their own advantages. SDLC is a framework that describes the activities performed at each stage of a software development life cycle.The software development activities suchas planning, analysis, design, coding, testing and maintenance which need to be performed according to the demand of the customer. It depends on the various applications to choose the specific model. In this paper, however, we will study the agile processes and its methodologies. Agile process is itself a software development process.Agile process is an iterative approach in which customer satisfaction is at highest priority as the customer has direct involvement in evaluating the software.

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.

Characteristics of agile projects

Agile process requires less planning and it divides the tasks into small increments. Agile process is meant for short term projects with an effort of team work that follows the software development life cycle. Software development life cycle includes the following phases 1.Requirements gathering, 2.Analysis, 3.Design, 4.Coding , 5.Testing, 6.Maintenance. The involvement of software team management with customers reduces the risks associated with the software. This agile process is an iterative process in which changes can be made according to the customer satisfaction. In agile process new features can be added easily by using multiple iterations.

Iterative

- The main objective of agile software processes is satisfaction of customers, so it focuses on single requirement with multiple iterations.

Modularity

- Agile process decomposes the complete system into manageable pieces called modules.Modularity plays a major role in software development processes.

Time Boxing

- As agile process is iterative in nature, it requires the time limits on each module with respective cycle.

Parsimony

- In agile processes parsimony is required to mitigate risks and achieve the goals by minimal number of modules.

Incremental

- As the agile process is iterative in nature, it requires the system to be developed in increments, each increment is independent of others, and at last all increments are integrated into complete system.

Adaptive

- Due to the iterative nature of agile process new risks may occurs. The adaptive characteristic of agile process allows adapting the processes to attack the new risks and allows changes in the real time requirements.

Convergent

- All the risks associated with each increment are convergent in agile process by using iterative and incremental approach.

Collaborative

- As agile process is modular in nature, it needs a good communication among software development team.Different modules need to be integrated at the end of the software development process.

PeopleOriented

- In the agile processes customer satisfaction is the first priority over the technology and process. A good software development team increases the performance and productivity of the software.

ADVANTAGES

1. Adaptive to the changing environment: In agile software development method, software is developed over several iterations. Each iteration is characterized by analysis, design, implementation and testing. After each iteration the mini project is delivered to the customer for their use and feedback. Any changes that upgrade the software are welcome from the customer at any stage of development and that changes are implemented.
2. Ensures customer satisfaction: This methodology requires active customer involvement throughout thedevelopment. The deliverables developed after each iteration is given to the user for use and improvement is done based on the customer feedback only. So at the end what we get as the final product is of high quality and it ensures the customer satisfaction as the entire softwareis developed based on the requirements taken from customer.
3. Least documentation: The documentation in agile methodology is shortand to the point though it depends on the agile team. Generally they don’t makedocumentation on internal design of the software. The main things which should be on the documentation are product features list, duration for each iteration and date. This brief documentation saves time of

development and deliver the project in least possible time.

4. Reduces risks of development: As the incremented mini software is delivered to the customers after every short development cycle and feedbacks are taken from the customers, it warns developers about the upcoming problems which may occur at the later stages of development. It also helps to discover errors quickly and they are fixed immediately.

DISADVANTAGES

Customer interaction is the key factor of developing successful software: Agile methodology is based on customer involvement because the entire project is developed according to the requirements given by the customers. So if the customer representative is not clear about the product features, the development process will go out of the track.

Lack of documentation: Though the least documentation saves development time as an advantage of agile method, on the other hand it is a big disadvantage for developer. Here the internal design is getting changed again and again depending on user requirements after every iteration, so it is not possible to maintain the detail documentation of design and implementation because of project deadline. So because of less available information, it is very difficult for the new developers who join the development team at thelater stage to understand the actual method followed to develop the software.

Time consuming and wastage of resources because of constant change of requirements: If the

customers are not satisfied by the partial software developed by certain iteration and they change their requirements then that incremented part is of no use. So it is the total wastage of time, effort and resources required to develop that increment.

More helpful for management than developer: The agile methodology helpsmanagement to take

decisions about the software development, set goals for developers and fix the deadline for them. But it is very difficult for the baseline developers to cope up with the ever changing environment and every time changing the design, code based on just in time requirements.

COMPARISON OF AGILE PROCESS WITH OTHER SDLC MODELS

TABLE I: PRCOESS MODELS

Different Process Models

Features	Agile Process	Spiral Model	RAD Model
Definition	Agile process is the ability to both create and respond tochanging requirements of software.	Spiral model is the software development model which focuses on managing risks.	RAD model is “high speed adaptation of linear sequential model, in which component based construction is used.
Adaptability	y	y	n
Testing Phase	Unit, Integration , System testing	Unit, Integration and System testing	Unit
Quality Factors	y	y	n
Risk Analysis	n	y	n
Off-the- Tools	n	n	y
Failure normally due to	Code	Code	Architecture and design
Knowledge Required	Product and domain	Product and domain	Domain
Entry & exit Criteria	n	n	y
Mock up	y	y	n
Extendability	y	y	n
Project management involvement	y	n	y
Higher Reliability	y	y	n
Time Boxing	y	n	y

Choosing the right process

Software process consists of four fundamental activities:

1. Software specification where engineers or/and customers define what the product should do and how should it operate.
2. Software development is designing and actual coding.
3. Software validation is generally testing. It is important to check if the system is designed and implemented correctly.
4. Software evolution is modifying the system according to new needs of customer (s).

Different types of software need different development process.

Software process model is a simplified description of a software process that presents one view of a process. And again, choice of a view depends on the system developing, sometimes it is useful to apply a workflow model, sometimes, for example – a role/action model.

Most software process models are based on one of three general models or paradigms of software development.

1. The waterfall approach. In this case the development process and all activities are divided into phases such as requirement specification, software design, implementation, testing etc. Development goes phase-by-phase.
2. Iterative development. An initial system is rapidly developed from very abstract specifications. Of course, it can be reimplemented according to new, probably more detailed specifications.
3. Component-based software engineering (CBSE). The development process is done assuming some parts of the system is already exist, so the process focuses on integrating parts together rather than developing everything from scratch.

Four principal dimensions to system dependability are: Availability, Reliability, Safety and Security.

All of these may be decomposed into another, for example security includes integrity (ensuring that data is not damaged) and confidentiality. Reliability includes correctness, precision and timeliness. All of them are interrelated.

Module-II

REQUIREMENTS MANAGEMENT

Functional requirements and Quality attributes

Quality attributes, such as response time, accuracy, security, reliability, are properties that affect the systems a whole. Most approaches deal with quality attributes separately from the functional requirements of a system. This means that the integration is difficult to achieve and usually is accomplished only at the later stages of the software development process. Furthermore, current approaches fail in dealing with the crosscutting nature of some of those attributes, i.e. it is difficult to represent clearly how these attributes can affect several requirements simultaneously. Since this integration is not supported from requirements to the implementation, some of the software engineering principles, such as abstraction, localization, modularisation, uniformity and reusability, can be compromised. What we propose is a model to identify and specify quality attributes that crosscut requirements including their systematic integration into the functional description at an early stage of the software development process, i.e. at requirements.

A model for early quality attributes

The process model we propose is UML compliant and is composed of three main activities: identification, specification and integration of requirements. The first activity consists of identifying all the requirements of system and select from those the quality attributes relevant to the application domain and stakeholders. The second activity is divided into two main parts: (1)specifying functional requirements using a use case based approach; (2) describe quality attributes using special templates and identify those that cut across (i.e. crosscutting) functional requirements. The third activity proposes a set of models to represent the integration of crosscutting quality attributes and functional requirements. Figure 1 depicts this model.

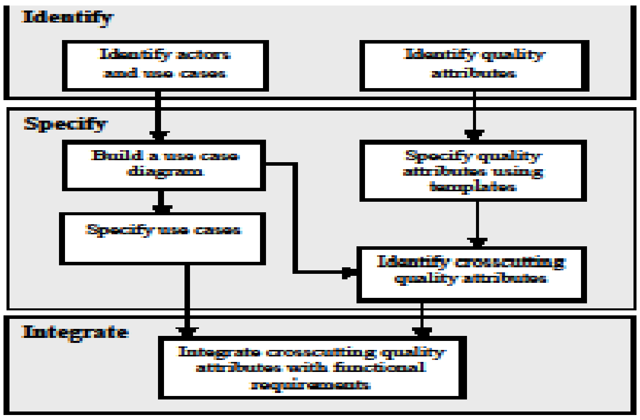
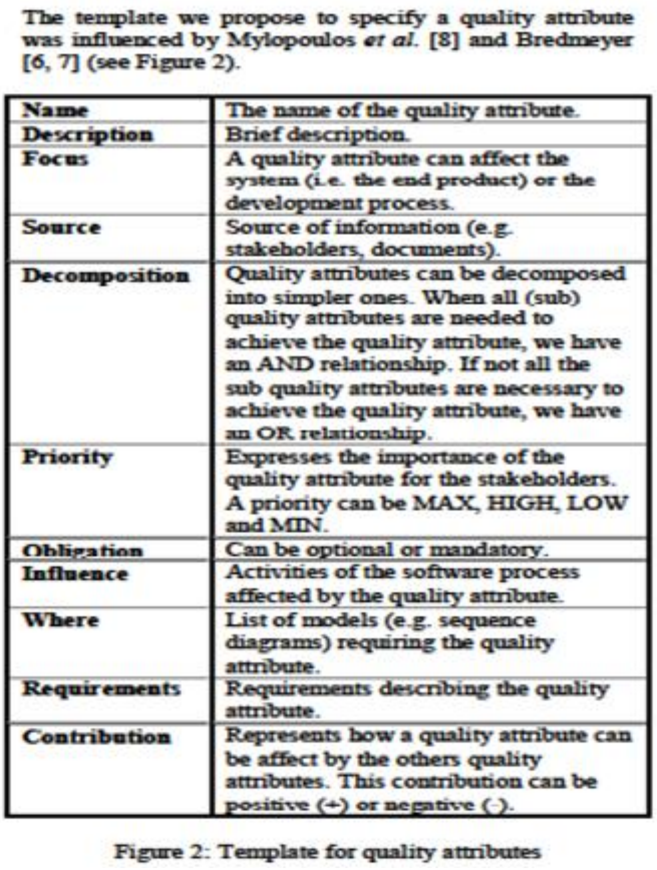

Identify
Identify actors and use cases
Identify quality attributes
Specify
Build a use case diagram
Specify quality attributes using templates
Specify use cases
Identify crosscutting quality attributes
Integrate
Integrate crosscutting quality attributes with functional requirements

Figure 1: A requirements model for quality attributes


The template we propose to specify a quality attribute was influenced by Mylopoulos *et al.* [8] and Bredmeyer [6, 7] (see Figure 2).

Name	The name of the quality attribute.
Description	Brief description.
Focus	A quality attribute can affect the system (i.e. the end product) or the development process.
Source	Source of information (e.g. stakeholders, documents).
Decomposition	Quality attributes can be decomposed into simpler ones. When all (sub) quality attributes are needed to achieve the quality attribute, we have an AND relationship. If not all the sub quality attributes are necessary to achieve the quality attribute, we have an OR relationship.
Priority	Expresses the importance of the quality attribute for the stakeholders. A priority can be MAX, HIGH, LOW and MIN.
Obligation	Can be optional or mandatory.
Influence	Activities of the software process affected by the quality attribute.
Where	List of models (e.g. sequence diagrams) requiring the quality attribute.
Requirements	Requirements describing the quality attribute.
Contribution	Represents how a quality attribute can be affect by the others quality attributes. This contribution can be positive (+) or negative (-).

Figure 2: Template for quality attributes

To identify the crosscutting nature of some of the quality attributes we need to take into account the information contained in rows Where and Requirements. If a quality attribute cuts across (i.e. is required by) several requirements and models, then it is crosscutting.

The integration is accomplished by “weaving” the quality attributes with the functional requirements in three different ways :
Overlap: the quality attribute adds new behaviour to the functional requirements it transverses. In this case, the quality attribute may be required before those requirements, or, it may be required after them.
Override: the quality attribute superposes the functional requirements it transverses. In this case, its behaviour substitutes the functional requirements behaviour.

Wrap: the quality attribute “encapsulates” the requirements it transverses. In this case the behaviour of the requirements is wrapped by the behaviour of the quality attribute. We weave quality attributes with functional requirements by using both standard diagrammatic representations (e.g. use case diagram, interaction diagrams) and by new diagrams.

Identify requirements

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. 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 actors and use cases

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

- Vehicle owner: is responsible for registering a vehicle;
- Vehicle driver: comprehends the vehicle, the driver and the gizmo installed on it;
- Bank: represents the entity that holds the vehicle owner’s account;
- System clock: represents the internal clock of the system that monthly triggers the calculation of debits.

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

- Register vehicle: is responsible for registering a vehicle and its owner, and communicate with the bank to guarantee a good account;
- Pass single toll: is responsible for dealing with tolls where vehicles pay a fixed amount. It reads the vehicle gizmo and checks on whether it is a good one. If the gizmo is ok the light is turned green, and the amount to be paid is calculated and displayed. If the gizmo is not ok, the light is turned yellow and a photo is taken.
- Enter motorway: checks the gizmo, turns on the light and registers an entrance. If the gizmo is invalid a photo is taken and registered in the system.
- Exit motorway: checks the gizmo and if the vehicle has an entrance, turns on the light accordingly, calculates the amount to be paid (as a function of the distance travelled), displays it and records this passage. If the gizmo is not ok, or if the vehicle did not enter in a green lane, the light is turned yellow and a photo is taken.
- Pay bill: sums up all passages for each vehicle, issues a debit to be sent to the bank and a copy to the vehicle owner.

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, if the owner of a vehicle has to indicate, during registration, his/her bank details so that automatic transfers can be performed automatically, then security is an issue that the system needs to address. Another fundamental quality attribute is response time that is a issue when a vehicle passes a toll gate, or when a customer activates his/her own gizmo in an ATM: the toll gate components have to react in time so that the driver can see the light and the amount being displayed. Other concerns are identified in a similar fashion: Multiuser System, Compatibility, Legal Issues, Correctness and Availability.

Specify functional requirements and quality attributes

The functional requirements are specified using the UML models, such as use cases, sequence and class diagrams. The quality attributes are described in templates of the form presented in Figure 2.

Build the use case diagram

The set of all use cases can be represented in a use case diagram, where we can see the existing relationships between use cases and the ones between use cases and actors. Figure 3 shows the use case diagram of the road traffic system.

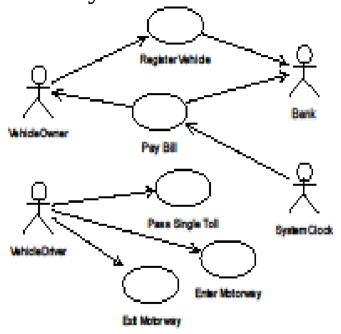
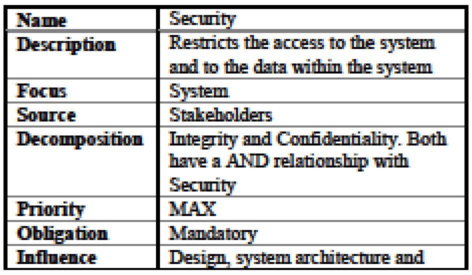


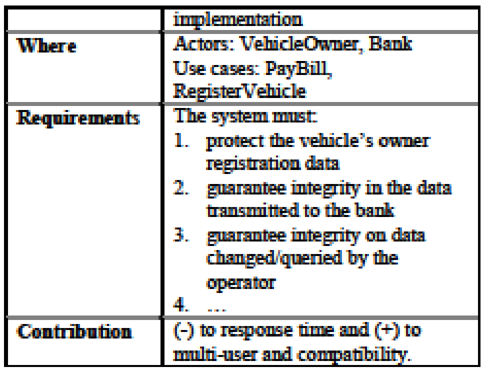
Figure 3. The use case diagram of the Road Traffic Pricing System

Integrate functional requirements with crosscutting quality attributes

Integration composes the quality attributes with the functional requirements, to obtain the whole system. We use UML diagrams to show the integration. The two examples given above (for response time and security) fall into two of the categories already described: overlap and wrapper. We could extend the UML diagrams to represent some quality attributes. For example, the sequence diagram shown in Figure 4 can be extended to show how response time affects a scenario



Name	Security
Description	Restricts the access to the system and to the data within the system
Focus	System
Source	Stakeholders
Decomposition	Integrity and Confidentiality. Both have a AND relationship with Security
Priority	MAX
Obligation	Mandatory
Influence	Design, system architecture and



	implementation
Where	Actors: VehicleOwner, Bank Use cases: PayBill, RegisterVehicle
Requirements	The system must: 1. protect the vehicle's owner registration data 2. guarantee integrity in the data transmitted to the bank 3. guarantee integrity on data changed/queried by the operator 4. …
Contribution	(-) to response time and (+) to multi-user and compatibility.

Figure 6. Template for Security

Elicitation techniques

A major goal of Requirements Elicitation is to avoid the confusions between stakeholders and analysts. This will often involve putting significant sort into requirements elicitation. Unfortunately, Requirements Engineering is an immature discipline, perhaps not entirely unfairly characterized as a battlefield occupied by competing commercial methods, firing competing claims at each other, and leaving the consumers weary and confused.

The goal of this paper is to analyze and compare of the different methods of the requirements elicitation process, which will be useful to compare the different characteristics and the performance of the different elicitation methods. Hence, all the requirement elicitation techniques are very handy for extracting the requirements and different organizations, which can use different requirement elicitation techniques according to organizational culture and needs.

As requirements elicitation is a process in which intensive interaction between stakeholders and the analysts, so for finding the interaction between stakeholders and analysts will be easy for improving the quality of extracted requirements. It is important to distinguish different elicitation methods according to the four methods of communication.

1. Conversational
2. Observational
3. Analytic
4. Synthetic

Each category presents a specific interaction model between analysts and stakeholders. Understanding the method category helps engineers understand different elicitation methods and guides them to select appropriate method for requirements elicitation.

Four Methods of Communication

I)Conversational Methods

The conversational method provides a means of verbal communication between stakeholders and Analysts. As conversation is a natural way of communication and an effective mean of expressing needs and ideas, and the conversational methods are used massively to understand the problems and to elicit generic product requirements. The Conversational Methods are also known as verbal methods, such as Interviews, Questionnaire, and Brainstorming.

a. Interviews: A typical conversational method is interviews. It is most commonly used method in requirements elicitation. An Interview is generally conducted by an experienced analyst, who has some generic knowledge about the application domain as well. In an interview, Analyst discusses the desired product with different stakeholders and develops an understanding of their requirements. Generally Interviews are divided in two groups.

1. Closed Interview: In this interview the requirements, we have to prepare some predefined questions and try to get the answers for these questions for the stakeholder.
2. Open-ended Interview: In this interview, we do not need to prepare any predefined questions, and the information from the stakeholders in open discussions.

b.Questionnaire: Questionnaires are one of the methods of gathering requirements in less cost. Questionnaires reach a large number of people, not only in less time but also in a lesser cost. The general factors which affect the usage of the questionnaire are

The available resources to gather the requirements mainly depends on the available resource

Type of Requirements that has to be gathering depends on the level of the respondent's knowledge and background.

Anonymity provided to the respondent

c.Brainstorming: Brainstorming is another conversation method. It has some similarities with workshops and focus groups as in Brainstorming stakeholders are gather together for a short time period but in this short time period they develop a large and broad list of ideas. In this meeting "out -of-the-box" thinking approach is encouraged. The brainstorming involves both idea generation and idea reduction.

Conversation is one of the most prevalent yet invisible forms of social interaction. People are usually happy to describe their work and difficulties they face. The verbally expressive demands, needs and constraints are often called non-tacit requirements. Conversational methods are very commonly used in requirements development. However, they are laborintensive : meeting setup and transcript producing and analyzing from records of a live interaction take time.

II)Observational Methods**:**

The observational method provides means to develop a better understanding about domain of Application. Observation methods work by observing human activities at environment where systemis expected to be deployed. In addition to state able requirements, some requirements are apparent to stakeholders, but stakeholders find it very hard to verbalize.

The observation methods come into play where Verbal communication becomes helpless for collecting tacit requirements. Therefore, observing how people carry out their routine work forms a means of acquisition of information which are hard to verbalize. The observational methods appear to be well suited when stakeholders find it difficult to state their needs and when analysts are looking for a better understanding of the context in which the desired product is expected to be used. Observational methods is including, Social analysis, Observation, Ethnographic study, and protocol analysis.

Social analysis, Observation, Ethnographic study: An observer spends some time in a society or culture for making detailed observation of all their practices. This practice gives the initial understanding of system, work flow and organizational culture.

Protocol analysis: In protocol analysis a stakeholder is observed when he is engaged in some task, and concurrently speaks out loud and explains his thought. With the protocol analysis it is easy to identify Interaction problems in existing systems and it gives better and closer understanding of Work context and work flow.

For Observational methods, the observer must be accepted by the people being studied and the people being studied should carry on with their normal activities as if the observer is not there.

In both Conversational and Observation methods, requirement elicitation is done by studying some individuals but a variety of documentation may prove out to be handy for extracting the requirements of the desired product. The documentation may include problem analysis, organizational charts, standards, user manuals of existing systems, survey report of competitive systems in market, and so on. By studying these documents, engineers capture the information about the application domain, the workflow, the product features, and map it to the requirements specification.

III)Analytic Methods:

Conversational or Observational methods are used to directly extracted requirements from people's behaviour and their verbalized thought. But still there is a lot of knowledge that is not directly expressed, for example expert's knowledge, information about regulation and legacy products are some examples of such sources. All the stated sources provide engineers rich information in relation to the product. Analytic methods provide ways to explore the existing documentation or knowledge and acquire requirements from a series of deductions.it will include Requirement reuse, documentation studies, laddering, and repertory grid

Requirement reuse: In this technique, glossaries and specification of legacy systems or systems within the same product family is used to identify requirements of the desired system.

It has been observed that many requirements in a new system are more or less same as they were in a legacy system's requirement. So it is not a bad idea to reuse the details of requirements of an earlier system in a new system.

Documentation studies: In this technique different available documents (e.g. Organizational policies, standards, legislation, Market information, Specification of legacy systems) are read and studied to find the content that can prove out to be relevant useful for the requirements elicitation tasks.

Laddering: This technique can be divided in 3 parts: creation, reviewing and modification. Laddering method is a form of structured interview that is widely used in the field of knowledge elicitation activities to elicit stakeholder's goals, aims and values Analyst used laddering method to create, review and modify the hierarchical contents of expert's knowledge in the form of tree diagram. It was first introduced by the clinical psychologists in 1960 to understand the people "score values and beliefs . Its success in the fields of psychology allows other researchers in the industries to adapt it in their fields. Specifically software developers have adapted the laddering techniques for gather the complex user tacit requirements.

Repertory grid: Stakeholder is asked for attributes applicable to a set of entities and values for cells in entity -attribute matrix.

In general, the analytic methods are not vital to requirements elicitation, since requirements are captured indirectly from other sources, rather than end users and customers. However, they form complementary ones to improve the efficiency and effectiveness of requirements elicitation, especially when the information from legacy or related products is reusable.

IV)Synthetic Methods:

So far, we have discussed Conversational, Observational and Analytic methods. It is apparent that No single method is sufficient enough to develop all the requirement of a system. All these methods are good and very handy in some certain context and circumstances. It is often a good idea to combine different elicitation methods for developing requirement. The combination helps the engineer uncover the basic aspects and gain a generic knowledge of the application domain. Instead of combining different of individual methods, the synthetic method forms a coherent whole by systematically combining conversation, observation, and analysis into single methods. Analysts and stakeholder representatives communicate and coordinate in different ways to reach a common understanding of the desired product. Synthetic methods are known as collaborative methods as they are collaboration of multiple requirement elicitation methods. Requirement elicitation techniques of Synthetic methods are including scenarios, passive storyboards, prototyping, interactive storyboards, JAD/RAD sessions, and Contextual inquiry.

Scenarios, passive storyboards: It is an interaction session. In this session a sequence of actions and events described for executing some generic task which the system is intended to accomplish. With the help of this technique, clear requirement related to procedure and data flow can be achieved. With this technique initial set of requirement can be prepared in lesser cost.

Prototyping, Interactive storyboards: In this technique, a concrete but partial system is discussed with stakeholders. This concrete but partial system is expected to be delivered at the end of project. The purpose of showing this system to stakeholders is to elicit and validate functional requirement.

JAD/RAD session: It stands for Joint Application Development/Rapid Application Development and emphasizes user involvement through group sessions with unbiased facilitator. JAD is conducted in the same manner as brainstorming, except that the stakeholders and the users are also allowed to participate and discuss on the design of the proposed system. The discussion with the stakeholders and the users continues until the final requirements are gathered.

Contextual inquiry: this technique is a combination of open-ended interview, workplace observation, and prototyping. This method used for interactive systems design where user interface design is critical.

All four requirement elicitation methods are commonly used but the selection of requirement elicitation method entirely depends on the needs and organizational structure. No matter what development project is, requirements development nearly always takes place in the context of a human activity system, and problem owners are people .. It is essential for requirements engineers to study how people perceive, understand, and express the problem domain, how they interact with the desired product, and how the physical and cultural environments affect their actions.

The conversational methods provide a direct contact channel between engineers and stakeholders, and the requirements are mainly no tacit. The observational methods provide an indirect channel by observing user’s interaction with his work setting and context, and the requirements fall into tacit knowledge. The analytic methods form one complementary indirect contact channel to extract requirements proactively. The synthetic methods focus more on collective effort on clarifying the features of desired products, and the communication channel is therefore a mix of direct contact and indirect contact. Each type of techniques has trade-offs. In reality, of course, the boundary between different types of method is blurred.

Advantage and Disadvantage of Requirement Elicitation

After the discussion the different of the four group of requirement elicitation method. In order to understand the each Requirement elicitation Methods and effective use them in the real case ,we have to focus on the advantages and disadvantages of different requirement elicitation methods: Conversational, Observational, Analytic and Synthetic one by one.

1) As conversation is a natural and effective way of communication, that’s why the conversational methods are used massively. Conversational methods include techniques such as: interviews, Questionnaire and Brainstorming.

Advantages of Conversational Method: Conversational techniques are really helpful for collection rich information about the requirements. Along with the requirements, conversational methods uncover opinions, feelings and goals of different individuals. With the help of conversational methods it is easy to dig into the details with the help of follow up questions to what the person has told you.

Disadvantages of Conversational Method: Along with the number of advantages there are certain disadvantages of conversational methods as this skill is very hard to master.

1)Conversational Methods for requirement elicitation depend a lot on the behavior and attitude of conductor . A Conductor is supposed to be neutral. As a result of conversational method, a collection of information can be obtained and getting meaningful information from gathered information will be difficult. In Conversational Methods the contexts of conversation plays a very important role as well.

2) Observational methods are helpful in understanding the application domain by observing human activities Observational methods are inefficient when the project have very tight schedule at requirement stages. Method like ethnography and protocol analysis methods falls under this category . The Observational method involves: Social analysis, Observation, Ethnographic study and Protocol Analysis.

Advantages of Observational Methods: The observational methods are good choice for uncovering basic aspects of routine order. Moreover they provide vital information for designing solution. Observational Methods are very handy when the development team has lack of experience about product domain.

Disadvantages of Observational Methods: Along with the advantages of observational methods there are certain disadvantages as well. The Biggest disadvantage is that observation methods need a lot of time and these techniques are not good choice when schedule is tight. Just like conversational techniques, observational techniques are also hard to master . Moreover observational techniques require sensitivity and responsiveness to physical environment.

3)Conversational or Observational methods are used to directly extracted requirements from people's behavior and their verbalized thought. But still there is a lot of knowledge that is not directly expressed. For extracting this kind of knowledge and information analytical skills are used. Analytical Skills include Requirement Reuse, Documentation Studies, Laddering and Repertory Girds.

Advantages of Analytical Methods: Analytic Methods have numerous advantages as "People" are not the only source of information in terms of requirements. Experts Knowledge and Opinion plays an important role in requirement maturity. Moreover, reuse of already available information saves time and cost. Analytical methods have hierarchical flow of information as well.

Disadvantages of Analytical Methods: Along advantages, Analytical methods have certain disadvantages as well. The biggest disadvantage is that an analytical method requires some empirical data, documentation or expert's opinions without these it is difficult to elicit proper requirements. Similarly analytical methods can narrow the vision of product. As analytical methods deal with some earlier knowledge so possibility of error replication is a serious and constant threat. Analytical methods are never a good choice when you are going to develop an altogether new system. [12]

Quality Attribute Workshops(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 was developed to complement the Architecture Tradeoffs Analysis Methodism (ATAMSM) and provides a way to identify important quality attributes and clarify system requirements before the software architecture has been created.

This is the third edition of a technical report describing the QAW. We have narrowed the scope of a QAW to the creation of prioritized and refined scenarios. This report describes the newly revised QAW and describes potential uses of the refined scenarios generated during it.

The Quality Attribute Workshop (QAW) is a facilitated method that engages system stakeholders early in the system development life cycle to discover the driving quality attributes ofa software-intensive system. The QAW is system-centric and stakeholder focused; it is used before the software architecture has been created. The QAW provides an opportunity to gather stakeholders together to provide input about their needs and expectations with respect to key quality attributes that are of particular concern to them

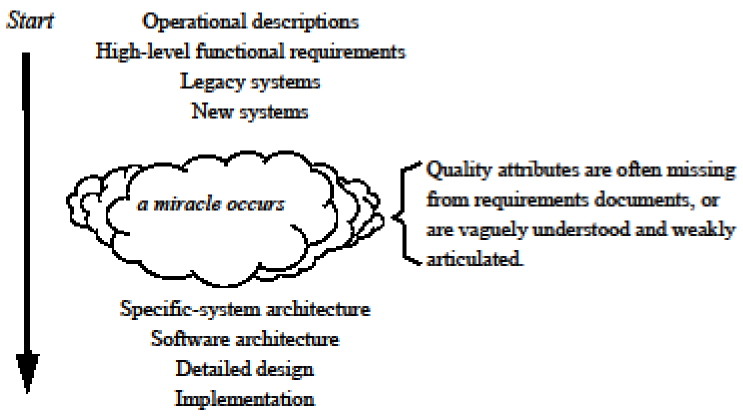

Start
Operational descriptions
High-level functional requirements
Legacy systems
New systems
a miracle occurs
Quality attributes are often missing from requirements documents, or are vaguely understood and weakly articulated.
Specific-system architecture
Software architecture
Detailed design
Implementation

Figure 1: Traditional System Development

Both the system and software architectures are key to realizing quality attribute requirements in the implementation. Although an architecture cannot guarantee that an implementation will meet its quality attribute goals, the wrong architecture will surely spell disaster. As an example, consider security. It is difficult, maybe even impossible, to add effective security to a systems an afterthought. Components as well as communication mechanisms and paths must be designed or selected early in the life cycle to satisfy security requirements. The critical quality attributes must be well understood and articulated early in the development of a system, so the architect can design an architecture that will satisfy them. The QAW is one way to discover, document, and prioritize a system's quality attributes early in its life cycle.
It is important to point out that we do not aim at an absolute measure of quality; rather our purposes to identify scenarios from the point of view of a diverse group of stakeholders (e.g., architects, developers, users, sponsors). These scenarios can then be used by the system engineers toanalyze the system's architecture and identify concerns (e.g., inadequate performance, successful denial-of-service attacks) and possible mitigation strategies (e.g., prototyping, modeling, simulation).

QAW Method
The QAW is a facilitated, early intervention method used to generate, prioritize, and refine quality attribute scenarios before the software architecture is completed. The QAW is focused on system-level concerns and specifically the role that software will play in the system. TheQAW is dependent on the participation of system stakeholders—individuals on whom the systemhas significant impact, such as end users, installers, administrators (of database management systems [DBMS], networks, help desks, etc.), trainers, architects, acquirers, system andsoftware engineers, and others. The group of stakeholders present during any one QAW shouldnumber at least 5 and no more than 30 for a

single workshop. In preparation for the workshop,stakeholders receive a “participants handbook” providing example quality attribute taxonomies,questions, and scenarios. If time allows, the handbook should be customized to the
domain of the system and contain the quality attributes, questions, and scenarios that are appropriate to the domain and the level of architectural detail available.

The contribution of each stakeholder is essential during a QAW; all participants are expectedto be fully engaged and present throughout the workshop. Participants are encouraged to comment and ask questions at any time during the workshop. However, it is important to recognizethat facilitators may occasionally have to cut discussions short in the interest of time or when itis clear that the discussion is not focused on the required QAW outcomes. The QAW is an intense and demanding activity. It is very important that all participants stay focused, are ontime, and limit side discussions throughout the day.

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

The following sections describe each step of the QAW in detail.
Step 1: QAW Presentation and Introductions
In this step, QAW facilitators describe the motivation for the QAW and explain each step ofthe method. We recommend using a standard slide presentation that can be customizeddepending on the needs of the sponsor.
Next, the facilitators introduce themselves and the stakeholders do likewise, briefly statingtheir background, their role in the organization, and their relationship to the system being built.

Step 2: Business/Mission Presentation
After Step 1, a representative of the stakeholder community presents the business and/or missiondrivers for the system. The term “business and/or mission drivers” is used carefully here.Some organizations are clearly motivated by business concerns such as profitability, whileothers, such as governmental organizations, are motivated by mission concerns and find profitabilitymeaningless.

The stakeholder representing the business and/or mission concerns (typicallya manager or management representative) spends about one hour presenting
• the system’s business/mission context
• high-level functional requirements, constraints, and quality attribute requirements
During the presentation, the facilitators listen carefully and capture any relevant informationthat may shed light on the quality attribute drivers. The quality attributes that will be refined inlater steps will be derived largely from the business/mission needs presented in this step.

Step 3: Architectural Plan Presentation
While a detailed system architecture might not exist, it is possible that high-level systemdescriptions, context drawings, or other artifacts have been created that describe some of thesystem’s technical details. At this point in the workshop, a technical stakeholder will presentthe system architectural plans as they stand with respect to these early documents. Informationin this presentation may include

- plans and strategies for how key business/mission requirements will be satisfied
- key technical requirements and constraints—such as mandated operating systems, hardware,
- middleware, and standards—that will drive architectural decisions
- presentation of existing context diagrams, high-level system diagrams, and other writtendescriptions

Step 4: Identification of Architectural Drivers
During steps 2 and 3, the facilitators capture information regarding architectural drivers thatare key to realizing quality attribute goals in the system. These drivers often include high-levelrequirements, business/mission concerns, goals and objectives, and various quality attributes.Before undertaking this step, the facilitators should excuse the group for a 15-minute break,during which they will caucus to compare and consolidate notes taken during steps 2 and 3.

When the stakeholders reconvene, the facilitators will share their list of key architectural driversand ask the stakeholders for clarifications, additions, deletions, and corrections. The idea isto reach a consensus on a distilled list of architectural drivers that include high-level requirements,business drivers, constraints, and quality attributes. The final list of architectural driverswill help focus the stakeholders during scenario brainstorming to ensure that theseconcerns are represented by the scenarios collected.

Step 5: Scenario Brainstorming
After the architectural drivers have been identified, the facilitators initiate the brainstormingprocess in which stakeholders generate scenarios. The facilitators review the parts of a goodscenario (stimulus, environment, and response) and ensure that each scenario is well formedduring the workshop.

Each stakeholder expresses a scenario representing his or her concerns with respect to the systemin round-robin fashion. During a nominal QAW, at least two round-robin passes are madeso that each stakeholder can contribute at least two scenarios. The facilitators ensure that atleast onerepresentative scenario exists for each architectural driver listed in Step 4.Scenario generation is a key step in the QAW method and must be carried out with care.

Wesuggest the following guidance to help QAW facilitators during this step:
Facilitators should help stakeholders create well-formed scenarios. It is tempting forstakeholders to recite requirements such as "The system shall produce reports for users."While this is an important requirement, facilitators need to ensure that the quality attributeaspects of this requirement are explored further. For example, the following scenario shedsmore light on the performance aspect of this requirement: "A remote user requests a databasereport via the Web during peak usage and receives the report within five seconds."Note that the initial requirement hasn't been lost, but the scenario further explores the performanceaspect of this requirement. Facilitators should note that quality attribute namesby themselves are not enough. Rather than say "the system shall be modifiable," the scenarioshould describe what it means to be modifiable by providing a specific example of amodification to the system vis-à-vis a scenario.
The vocabulary used to describe quality attributes varies widely. Heated debates oftenrevolve around to which quality attribute a particular system property belongs. It doesn'tmatter what we call a particular quality attribute, as long as there's a scenario thatdescribes what it means.
Facilitators need to remember that there are three general types of scenarios and to ensurethat each type is covered during the QAW:

- use case scenarios - involving anticipated uses of the system
- growth scenarios - involving anticipated changes to the system
- exploratory scenarios - involving unanticipated stresses to the system that can includeuses and/or changes
- Facilitators should refer to the list of architectural drivers generated in Step 4 from time totime during scenario brainstorming to ensure that representative scenarios exist for eachone.

Step 6: Scenario Consolidation
After the scenario brainstorming, similar scenarios are consolidated when reasonable. To do that, facilitators ask stakeholders to identify those scenarios that are very similar in content. Scenarios that are similar are merged, as long as the people who proposed them agree andfeels that their scenarios will not be diluted in the process. Consolidation is an important step because it helps to prevent a "dilution" of votes during the prioritization of scenarios (Step 7).Such a dilution occurs when stakeholders split their votes between two very similar scenarios. As a result, neither scenario rises to importance and is therefore never refined (Step 8). However, if the two scenarios are similar enough to be merged into one, the votes might be concentrated, and the merged scenario may then rise to the

appropriate level of importance and be refinedfurther. Facilitators should make every attempt to reach a majority consensus with the stakeholders before merging scenarios. Though stakeholders may be tempted to merge scenarios with abandon, they should not do so. In actuality, very few scenarios are merged.

Step 7: Scenario Prioritization

Prioritization of the scenarios is accomplished by allocating each stakeholder a number of votes equal to 30% of the total number of scenarios generated after consolidation. The actual number of votes allocated to stakeholders is rounded to an even number of votes at the discretion of the facilitators. For example, if 30 scenarios were generated, each stakeholder gets 30 x0.3, or 9, votes rounded up to 10. Voting is done in round-robin fashion, in two passes. . Stakeholders can allocate any number oftheir votes to any scenario or combination of scenarios. The votes are counted, and the scenarios are prioritized accordingly.

Step 8: Scenario Refinement

After the prioritization, depending on the amount of time remaining, the top four or five scenarios are refined in more detail. Facilitators further elaborate each one, documenting the following:

Further clarify the scenario by clearly describing the following six things:

1. stimulus - the condition that affects the system
2. response - the activity that results from the stimulus
3. source of stimulus - the entity that generated the stimulus
4. environment - the condition under which the stimulus occurred
5. artefact stimulated - the artefact that was stimulated
6. response measure - the measure by which the system’s response will be evaluated

Describe the business/mission goals that are affected by the scenario.

• Describe the relevant quality attributes associated with the scenario.

• Allow the stakeholders to pose questions and raise any issues regarding the scenario. Such questions should concentrate on the quality attribute aspects of the scenario and any concerns that the stakeholders might have in achieving the response called for in the scenario. See the example template for scenario refinement in Appendix A. This step continues until time runs out or the highest priority scenarios have been refined. Typically, time runs out first.

QAW Benefits

The QAW provides a forum for a wide variety of stakeholders to gather in one room at onetime very early in the development process. It is often the first time such a meeting takes place and generally leads to the identification of conflicting assumptions about system requirements. In addition to clarifying quality attribute requirements, the QAW provides increased stakeholder communication, an informed basis for architectural decisions, improved architectural documentation, and support for analysis and testing throughout the life of the system.

The results of a QAW include

- a list of architectural drivers
- the raw scenarios
- the prioritized list of raw scenarios
- the refined scenarios

This information can be used to

- update the organization’s architectural vision
- refine system and software requirements
- guide the development of prototypes
- exercise simulations
- understand and clarify the system’s architectural drivers
- influence the order in which the architecture is developed
- describe the operation of a system

In short, the architect can use this information to design the architecture. In addition, after the architecture is created, the scenarios can be used as part of a software architecture evaluation. If the Architecture Tradeoffs Analysis Methodism (ATAMSM)4 is selected as the software architecture evaluation method, the scenarios generated during the QAW can be incorporated as seed scenarios in that evaluation .

The QAW lends itself well to the capture of many architecturally relevant materials. Software architectural documentation is a collection of view packets plus any documentation that applies to more than one view [Clements 02b]. Each view packet contains a primary presentation, a catalog of the view’s elements (including element behaviour), a context diagram, a variability guide, architecture background (rationale, analysis results, and assumptions about the environment), and other information including mapping to requirements.

Several pieces of this information will be gleaned directly from the QAW. For example, scenario generation can lead to the creation of use case diagrams, context diagrams, or their equivalent. Refined scenarios can be documented as sequence diagrams or collaboration diagrams. Stakeholders’ concerns and any other rationale information that is captured should be recorded individually in a form that can be included in the appropriate view packet or overview documentation. Details that explain how to transition these artefacts into architectural documentation is the subject of ongoingresearch. In addition to the more immediate benefits cited above, the scenarios continue to provide benefits during later phases of development. They provide input for analysis throughout the life of the system and can be used to drive test case development during implementation testing.

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: I notations for model record, I methods of model preparation, I 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.

Just as blueprints in the building construction industry guides the construction of a building, the software architecture serves a blueprint that addresses technical concerns and programmatic issues of a project. An architectural focus will:
help refine the functional requirements, quality attribute requirements, and constraints
help set and maintain expectations in stakeholders
define the team structure
aid in creating more accurate project estimates
establish the team vocabulary
help identify technical risk early
guide the creation of a more realistic and accurate production schedule and assist in project tracking and oversight
provide an early vision of the solution/system

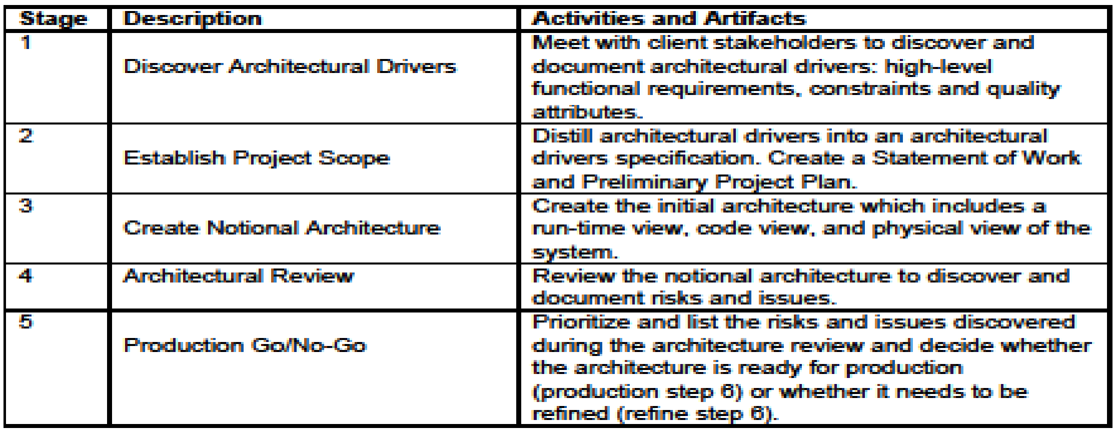
A number of methods have been created by the Software Engineering Institute to help practitioners create better architectures. Some of these methods include: Quality Attribute Workshop (QAW) ,ArchitectureTradeoff Analysis Method (ATAM)], Attribute Driven Design (ADD). These methods have provided great value to practitioners trying to build better architectures. However, these methods have two main problems. First, they are intervention oriented. These methods were not designed with a particular development philosophy (lifecycle or process) in mind. As such, they do not fit neatly into existing development models or processes without significant tailoring

Organizations are constantly bombarded with emerging methods, tools, and techniques and they must:
• figure out if they are useful
• how to use them
• how to make them fit together
• estimate the costs for adoption
• show return on investment

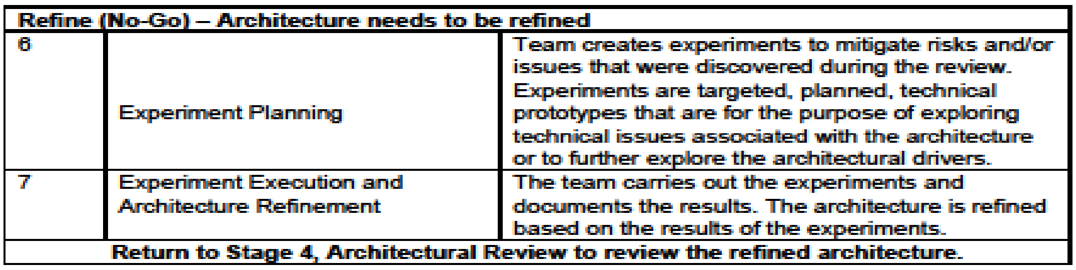
Architecture Centric Development Method (ACDM)

Software development teams need specific guidance about how to create software architecture in the context of a product development lifecycle. ACDM brings together some of the best practices into a lifecycle development model. The key goals of ACDM are to help software development teams:

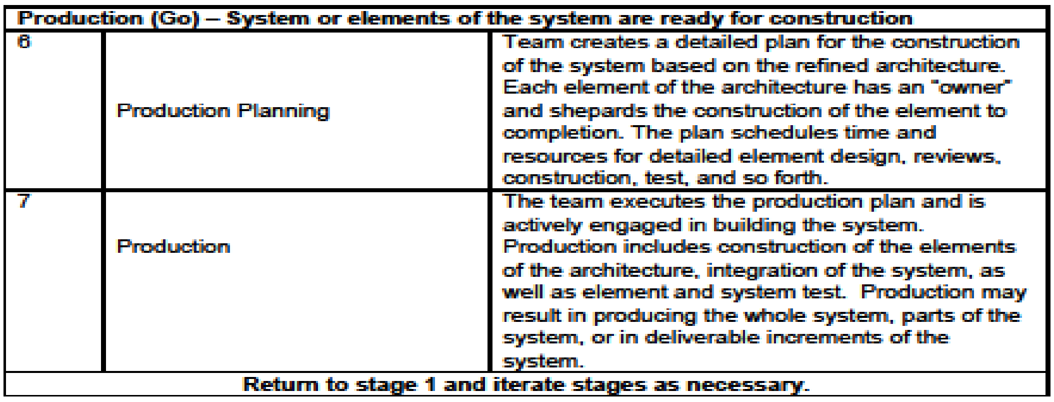
- Get the information from stakeholders needed to define the architecture as early as possible.
- Create, refine, and update the architecture in an iterative way throughout the lifecycle whether the lifecycle is waterfall or iterative.
- Validate that the architecture will meet the expectations once implemented.
- Define meaningful roles for team members to guide their efforts.
- Create better estimates and schedules based on the architectural blueprint.
- Provide insight into project performance.
- Establish a lightweight, scalable, tailor able, repeatable process framework.
- The ACDM is geared toward organizations and teams building software intensive systems and puts the software architecture “front-and-center” during all phases of the project. The method prescribes creating a notional architecture as soon as the most preliminary requirements work has been completed. The architecture is developed early and iteratively refined as a central focus of the project. The architecture is refined until the development team is confident that a system can be implemented and it will meet the needs of the stakeholder community. In ACDM, the architecture is the locus for defining all subsequent processes, planning, activities, and artifacts. Preconditions for beginning ACDM are defining roles for all of the team members. The method describes several roles and their responsibilities. The ACDM essentially follows seven prescribed stages briefly described below.



Stage	Description	Activities and Artifacts
1	Discover Architectural Drivers	Meet with client stakeholders to discover and document architectural drivers: high-level functional requirements, constraints and quality attributes.
2	Establish Project Scope	Distill architectural drivers into an architectural drivers specification. Create a Statement of Work and Preliminary Project Plan.
3	Create Notional Architecture	Create the initial architecture which includes a run-time view, code view, and physical view of the system.
4	Architectural Review	Review the notional architecture to discover and document risks and issues.
5	Production Go/No-Go	Prioritize and list the risks and issues discovered during the architecture review and decide whether the architecture is ready for production (production step 6) or whether it needs to be refined (refine step 6).



Refine (No-Go) – Architecture needs to be refined		
6	Experiment Planning	Team creates experiments to mitigate risks and/or issues that were discovered during the review. Experiments are targeted, planned, technical prototypes that are for the purpose of exploring technical issues associated with the architecture or to further explore the architectural drivers.
7	Experiment Execution and Architecture Refinement	The team carries out the experiments and documents the results. The architecture is refined based on the results of the experiments.
Return to Stage 4, Architectural Review to review the refined architecture.		

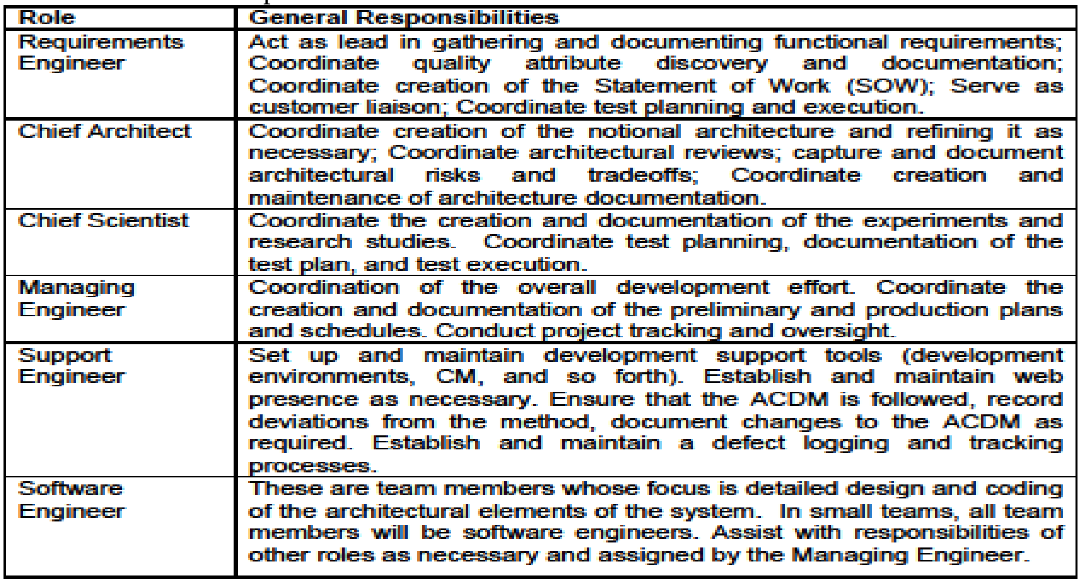


Production (Go) – System or elements of the system are ready for construction		
6	Production Planning	Team creates a detailed plan for the construction of the system based on the refined architecture. Each element of the architecture has an "owner" and shepards the construction of the element to completion. The plan schedules time and resources for detailed element design, reviews, construction, test, and so forth.
7	Production	The team executes the production plan and is actively engaged in building the system. Production includes construction of the elements of the architecture, integration of the system, as well as element and system test. Production may result in producing the whole system, parts of the system, or in deliverable increments of the system.
Return to stage 1 and iterate stages as necessary.		

While ACDM emerged from small teams and projects (4 to 6 team members, 1 to 2 year projects), it is designed to scale up to meet the needs of larger teams and projects as well. In larger projects, the ACDM is used by a core architecture team to create and refine the overall system architecture. The output from this ACDM cycle is an initial partitioning of the system (or system of systems) into sub-elements (or subsystems) and their interactions. Detailed architecting of the various elements is deferred to smaller teams, each using ACDM to architect their part of the system (which may be another system). Later integration of the entire system is undertaken in production stages 6 and 7. The ACDM has been evolved over a five year period (since 1999) on small projects and is now being further refined for use on larger projects in industry.

ACDM Preconditions

A precondition to beginning step 1 of ACDM is to establish the team roles for project. The recommended roles and responsibilities for ACDM are listed in the table below:



Role	General Responsibilities
Requirements Engineer	Act as lead in gathering and documenting functional requirements; Coordinate quality attribute discovery and documentation; Coordinate creation of the Statement of Work (SOW); Serve as customer liaison; Coordinate test planning and execution.
Chief Architect	Coordinate creation of the notional architecture and refining it as necessary; Coordinate architectural reviews; capture and document architectural risks and tradeoffs; Coordinate creation and maintenance of architecture documentation.
Chief Scientist	Coordinate the creation and documentation of the experiments and research studies. Coordinate test planning, documentation of the test plan, and test execution.
Managing Engineer	Coordination of the overall development effort. Coordinate the creation and documentation of the preliminary and production plans and schedules. Conduct project tracking and oversight.
Support Engineer	Set up and maintain development support tools (development environments, CM, and so forth). Establish and maintain web presence as necessary. Ensure that the ACDM is followed, record deviations from the method, document changes to the ACDM as required. Establish and maintain a defect logging and tracking processes.
Software Engineer	These are team members whose focus is detailed design and coding of the architectural elements of the system. In small teams, all team members will be software engineers. Assist with responsibilities of other roles as necessary and assigned by the Managing Engineer.

The ACDM also assumes that the functional requirements and constraints exist but does not discuss in detail how to get them, document them, and organize them. This may seem somewhat naive but this is intentional since requirement gathering, documenting, and organization varies widely even in our small studio projects. While ACDM does not address the gathering of initial requirements and constraints, it will help refine them, clarify them, as the architecture is designed and matures. The relative completeness of the functional requirements varies from project to project and may have to be discovered and refined as a consequence of building the system. Some clients provide a documented list of functional requirements; others just bring ideas to the team. The initial gathering of functional requirements is assumed to have occurred prior to beginning step 1 of ACDM. The requirements engineer will coordinate the gathering and documenting of functional requirements. The term “constraints” as applied in this context can be confusing. A “constraint” is an imposed design decision or a design decision that the architect is not at liberty to make or change. Example constraints include being forced to use a particular operating system, use a particular commercial off-the-shelf product, adhere to a particular standard, or build a system using a prescribed implementation framework.

Requirements documentation and specification

A Software requirements specification (SRS), a requirements specification for a software system, is a description of the behavior of a system to be developed and may include a set of use cases that describe interactions the users will have with the software. In addition it also contains non-functional requirements. Non-functional requirements impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints) .

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. This is achieved and refined with detailed and continuous communications with the project team and customer till the completion of the software.

Change management

Globalization and the constant innovation of technology result in a constantly evolving business environment. Phenomena such as social media and mobile adaptability have revolutionized business and the effect of this is an ever increasing need for change, and therefore changes management. The growth in technology also has a secondary effect of increasing the availability and therefore accountability of knowledge. Easily accessible information has resulted in unprecedented scrutiny from stockholders and the media and pressure on management.

With the business environment experiencing so much change, organizations must then learn to become comfortable with change as well. Therefore, the ability to manage and adapt to organizational change is an essential ability required in the workplace today. Yet, major and rapid organizational change is profoundly difficult because the structure, culture, and routines of organizations often reflect a persistent and difficult-to-remove "imprint" of past periods, which are resistant to radical change even as the current environment of the organization changes rapidly.[10]

Due to the growth of technology, modern organizational change is largely motivated by exterior innovations rather than internal moves. When these developments occur, the organizations that adapt quickest create a competitive advantage for themselves, while the companies that refuse to change get left behind. This can result in drastic profit and/or market share losses.
Organizational change directly affects all departments from the entry level employee to senior management. The entire company must learn how to handle changes to the organization.

Choosing what changes to implement
When determining which of the latest techniques or innovations to adopt, there are four major factors to be considered:

1. Levels, goals, and strategies
2. Measurement system
3. Sequence of steps

4. Implementation and organizational change

Managing the change process

Regardless of the many types of organizational change, the critical aspect is a company's ability to win the buy-in of their organization's employees on the change. 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

As a multi-disciplinary practice that has evolved as a result of scholarly research, organizational change management should begin with a systematic diagnosis of the current situation in order to determine both the need for change and the capability to change. The objectives, content, and process of change should all be specified as part of a Change Management plan.

Change management processes should include creative marketing to enable communication between changing audiences, as well as deep social understanding about leadership's styles and group dynamics. As a visible track on transformation projects, Organizational Change Management aligns groups' expectations, communicates, integrates teams and manages people training. It makes use of performance metrics, such as financial results, operational efficiency, leadership commitment, communication effectiveness, and the perceived need for change to design appropriate strategies, in order to avoid change failures or resolve troubled change projects.

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

1. Benefits management and realization to define measurable stakeholder aims, create a business case for their achievement (which should be continuously updated), and monitor assumptions, risks, dependencies, costs, return on investment, dis-benefits and cultural issues affecting the progress of the associated work
2. Effective communication that informs various stakeholders of the reasons for the change (why?), the benefits of successful implementation (what is in it for us, and you) as well as the details of the change (when? where? who is involved? how much will it cost? etc.)
3. Devise an effective education, training and/or skills upgrading scheme for the organization
4. Counter resistance from the employees of companies and align them to overall strategic direction of the organization
5. Provide personal counseling (if required) to alleviate any change-related fears
6. Monitoring of the implementation and fine-tuning as required

Examples

- Mission changes
- Strategic changes
- Operational changes (including Structural changes)
- Technological changes
- Changing the attitudes and behaviors of personnel
- Personality Wide Changes

Traceability of requirements

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

Other common definitions include to capability (and implementation) of keeping track of a given set or type of information to a given degree, or the ability to chronologically interrelate uniquely identifiable entities in a way that is verifiable.

Supply chain

In the supply chain, traceability is more of an ethical or environmental issue. Environmentally friendly retailers may choose to make information regarding their supply chain freely available to customers, illustrating the fact that the products they sell are manufactured in factories with safe working conditions, by workers that earn a fair wage, using methods that do not damage the environment.

Software development

In software development, the term traceability (or Requirements Traceability) refers to the ability to link product requirements back to stakeholders' rationales and forward to corresponding design artifacts, code, and test cases. Traceability supports numerous software engineering activities such as change impact analysis, compliance verification or trace back of code, regression test selection, and requirements validation. It is usually accomplished in the form of a matrix created for the verification and validation of the project. Unfortunately the practice of constructing and maintaining a requirements trace matrix (RTM) can be very arduous and over time the traces tend to erode into an inaccurate state unless date/time stamped. Alternate automated approaches for generating traces using information retrieval methods have been developed.

In transaction processing software, traceability implies use of a unique piece of data (e.g., order date/time or a serialized sequence number) which can be traced through the entire software flow of all relevant application programs. Messages and files at any point in the system can then be audited for correctness and completeness, using the traceability key to find the particular transaction. This is also sometimes referred to as the transaction footprint

Module – III
ESTIMATION, PLANNING, AND TRACKING

Identifying and Prioritizing Risks

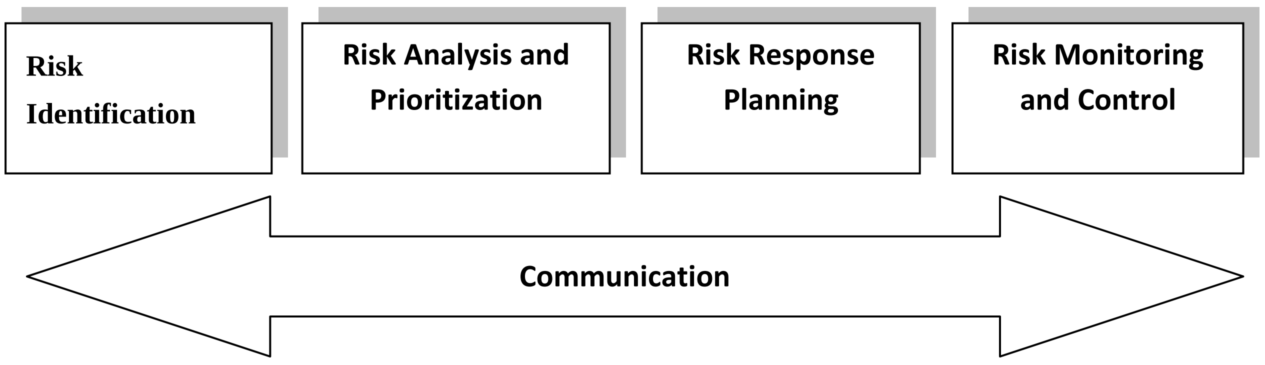
The formal process by which risks factors are systematically identified, assessed, and responded to. Risk management concentrates on identifying and controlling areas or events that have a potential of causing unwanted change. (Note that opportunities, also known as positive risk, should also be managed/exploited. This document is focused on mitigating negative risk, rather than maximizing positive risk.)

Definitions, Acronyms, and Abbreviations

Risk	A potential undesirable and unplanned event or circumstance, anticipated in advance, which could prevent the project from meeting one or more of its objectives.
Issue	An event or circumstance that has occurred with project impact that needs to be managed and resolved, with escalation if appropriate.
Task / Action Item	Work packages from the Work Breakdown Structure (WBS) or work resulting from project meetings or conversations.

Risk Management Approach

The project team will implement a continuous risk management process which entails two major processes – risk assessment and risk mitigation.
Risk assessment includes activities to identify risks, analyze and prioritize. Risk mitigation includes developing risk contingency and mitigation strategies, as well as monitoring the impact of the issue, action items, strategies and residual risks.



Risk Tolerance

The company has a very low threshold for risks to:

- The client experience
- The experience of users who directly support the client
- Non-public information (NPI)
- Potential for fraud or loss related to insufficient control or security

Risk Management Tasks

Risk Management activities are documented in the Risk Management workbook. The workbook is used to identify, prioritize, analyze, and plan a risk response.

Risk Identification**:** The process of determining which risks may affect the project and documenting their characteristics.

- Risk Assessment**:**The Risk Assessement 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. Each question has a potential rating of High, Medium, or Low in terms of potential impact.
- Risk Register**:**This is located on the project's SharePoint site where project specific risks can be entered. All risks identified through any means should be entered individually in the Risk Register on SharePoint. Like all company documentation, discretion should be used in documenting risk: all statements should be fact-based and conclusions should be reviewed by management (and if appropriate, Legal.) Risks should be stated in a standard format, to help the team stay focused on risks versus root causes and results: Cause – Risk – Effect.
    - Cause: specific situation that introduces risk
    - Risk: uncertain event that can impact the project
    - Effect: potential consequences of the risk occurring

  Example: A shortage of skilled Business Analysts (cause) could result in many missed requirements (risk), leading to rework or customer dissatisfaction (effect).

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 Response Planning: The process of developing options and actions to enhance opportunities and to reduce threat to project objectives. Mitigating actions are documented on the Risk Assessment and Mitigation tab in the Risk Management workbook and in the Risk Register. If a risk is prioritized as High, then mitigating actions must be documented (area is unshaded). If a risk is prioritized as

Medium, then mitigating actions are recommended, but not required. If a risk is prioritized as Low, then mitigating actions are not required.

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

- Risk-related change to Scope/Time/Cost

  The risk response planning process may result in a decision to avoid a risk by changing the project, or to mitigate a risk by taking action to lesser the probability and/or impact in the event the risk occurs. Whenever risk response planning results in potential change to the project, that change must first be requested, analyzed and approved in accordance with the project's Change Management Plan and related processes.

Risk Monitoring and Control: The process of implementing risk response plans, tracking identified risks, monitoring residual risks, identifying new risk, and evaluating risk process effectiveness throughout the project.

- Monitoring Risks:Project teams should review project risks and on regular basis to determine if there are any new project risks and to determine if any actions are needed (if a risk turns to an issue).
- Escalation:Ifa overall project risk is:
  - Low:Project risks are low and therefore no additional review need to occur.
  - Medium:Project risks should be reviewed on a monthly basis by the Business Owner, Technical Owner and core project team.
  - High:Project risks should be reviewed on a monthly basis by the Project Sponsor and Project Steering Committee.

Estimation Techniques

Estimation of software projects can be done by different techniques. The important techniques are:

1. Estimation by Expert Judgement
2. Estimation by Analogy
3. Estimation by Available Resources
4. Estimation by Software Price
5. Estimation by Parametric Modeling

Use Case Points

Use Case Points are used as an analysis phase technique for estimating software development. Assuming the Business Analyst (BA) composes system use cases for describing functional requirements, the BA can use this technique for estimating the follow-on implementation effort. This article reviews the process of estimating the follow-on development effort for use cases. 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. –

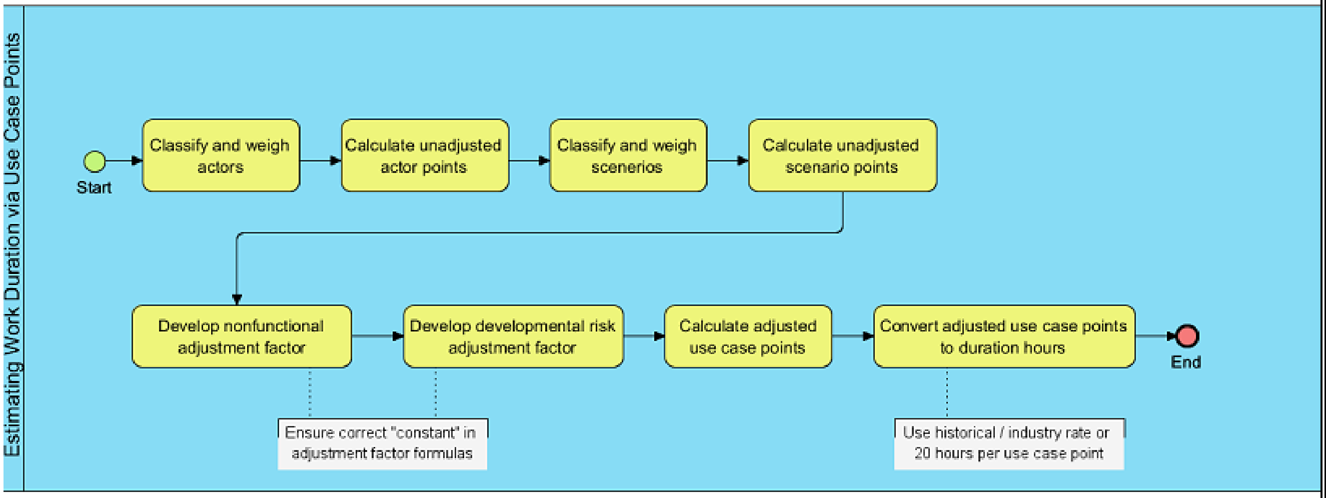

Estimating Work Duration via Use Case Points
Start
Classify and weigh actors
Calculate unadjusted actor points
Classify and weigh scenerios
Calculate unadjusted scenario points
Develop nonfunctional adjustment factor
Develop developmental risk adjustment factor
Calculate adjusted use case points
Convert 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

Functional 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

Project Scoping

A recommended approach for developing function point counts is to first functionally decompose the software into its elementary functional components (base functional components). This decomposition may be illustrated graphically on a functional hierarchy. The hierarchy provides a pictorial ‘table of contents’ or ‘map’ of the functionality of the application to be delivered. This approach has the advantage of being able to easily convey the scope of the application to the user, not only by illustrating the number of functions delivered by each functional area, but also a comparative size of each functional area measured in function points.

Assessing Replacement Impact

If the software to be developed is planned to replace existing production applications, it is useful to asses if the business is going to be delivered more, less or the same functionality. The replacement system’s functionality can be mapped against the functionality in the existing system. A quantitative assessment of the difference can be measured in function points. Note, this comparison can only be done if the existing applications have already been sized in function points

Assessing Replacement Cost

Multiplying the size of the application to be replaced by an estimate of the dollar cost per function point to develop, enables project sponsors to develop quick estimates of replacement costs. Industry derived costs are available and provide a ballpark figure for the likely cost. Industry figures are a particularly useful reference if the re-development is for a new software or hardware platform not previously experienced by the organisation. Ideally, organisations should establish their own ‘cost per function point’ metrics for their own particular environment, based on project history.

If you are considering implementing a ‘customised off the shelf’ package solution, then this provides a quick comparison of the estimated package implementation costs to compare with an in-house build. Package costs typically need to include the cost of re-engineering the business to adapt the current business processes to those delivered by the package. These costs are usually not a consideration for in-house developed software.

Negotiating Scope

Initial project estimates often exceed the sponsor's planned delivery date and budgeted cost. A reduction in the scope of the functionality to be delivered is often needed so that it is delivered within a predetermined time or budget constraints. The functional hierarchy provides the ‘sketch-pad’ to do scope negotiation. It enables the project manager and the user to work together to identify and flag (label) those functions which are: mandatoryfor the first release of the application; essential but not mandatory; or optional and could be held over to a subsequent release.

The scope of the different scenarios can then be quickly determined by measuring the functional size of the different scenarios. For example, the project size can be objectively measured to determine what the size (and cost and duration) would be if all functions are implemented, only mandatory functions are implemented, onlymandatory and essential functions are implemented. This allows the user to make more informed decisions on which functions will be included in each release of the application, based on their relative priority compared to what is possible given the time, cost and resource constraints of the project.

Evaluating Requirements

Functionally sizing the requirements for the application quantifies the different types of functionality delivered by an application. The function point count assigns function points to each of the function types: External Inputs, Outputs, Enquiries, and Internal and External Files.

Industry figures available from the ISBSG repository for projects measured with IFPUG function points indicate that ‘complete’ applications tend to have consistent and predictable ratios of each of the function types. The profile of functionality delivered by each of the function types in a planned application can be compared to that of the typical profile from implemented applications, to highlight areas where the specifications may be incomplete or there may be anomalies.

The following pie chart illustrates the function point count profile for a planned Accounts Receivable application compared to that from the ISBGS data. The reporting functions (outputs) are lower than predicted by industry comparisons. Incomplete specification of reporting functions is a common phenomenon early in a project’s lifecycle and highlights the potential for substantial growth creep later in the project as the user identifies all their reporting needs.

Estimating Project Resource Requirements

Once the scope of the project is agreed, the estimates for effort, staff resources, costs and schedules need to be developed. If productivity rates (hours per function point, $cost per function point) from previous projects are known, then the project manager can use the function point count to develop the appropriate estimates. If your organisation has only just begun collecting these metrics and does not have sufficient data to establish its own productivity rates, then the ISBSG industry data can be used in the interim.

Allocating Testing Resources

The functional hierarchy developed as part of the function point count during project development can assist the testing manager to identify high complexity functional areas which may need extra attention during the testing phase. Dividing the total function points for each functional area by the total number of functions allocated to that group of functions, enables the assessment of the relative complexity of each of the functional areas.

The effort to perform acceptance testing and the number of test cases required is related to the number and complexity of the user functions within a functional area. Quantifying the relative size of each functional area will enable the project manager to allocate appropriate testing staff and check relative number of test cases assigned.

Risk Assessment

Many organisations have large legacy software applications that, due to their age, are unable to be quickly enhanced to changed business needs. Over time, these applications have been patched and expanded until they have grown to monstrous proportions. Frustrated by long delays in implementing changes, lack of support for their technical platform and expensive support costs, management will often decide to redevelop the entire application. For many organisations, this strategy of rebuilding their super-large applications has proved to be a disaster, resulting in cancellation of the project middevelopment. Industry figures show that the risk of project failure rapidly increases with project size. Projects less than 500 function points have a risk of failure of less than 20% in comparison with projects over 5,000 function points which have a probability of cancellation close to 40%. This level of risk is unacceptable for most organisations.

Assessing planned projects for their delivered size in function points enables management to make informed decisions about the risk involved in developing large, highly integrated applications or adopting a lower risk phased approach described below.

Phasing Development

If the project manager decides on a phased approach to the project development, then related modules may be relegated to different releases. This strategy may require temporary interfacing functionality to be built in the first release, to be later decommissioned when the next module is integrated. The function point count allows project managers to develop 'what-if' scenarios and quantify the project scope of each phase as a means of making objective decisions. Questions to which quantitative answers can be provided are:

how much of the interfacing functionality can be avoided by implementing all of the related modules in release one?

what is the best combination of potential modules to group within a release to minimise the development of temporary interfacing functions?

If it is decided to implement the application as a phased development, then the size of each release can be optimised to that which is known to be manageable. This can be easily done by labelling functions with the appropriate release and performing 'what-if' scenarios by including and excluding functions from the scope of the count for the release.

FPA Uses and Benefits in Project Construction

Monitoring Functional Creep

Function point analysis provides project management with an objective tool by which project size can be monitored for change, over the project's lifecycle.

As new functions are identified, functions are removed or changed during the project, the function point count is updated and the impacted functions appropriately flagged. The project scope can be easily tracked and reported at each of the major milestones.

If the project size exceeds the limits allowed in the initial estimates, then this will provide an early warning that new estimates may be necessary or, alternatively, highlight a need to review the functionality to be delivered by this release.

Assessing and Prioritising Rework

Function Point Analysis allows the project manager to objectively and quantitatively measure the scope of impact of a change request, and to estimate the resulting impact on project schedule and costs. This immediate feedback to the user on the impact of the rework allows them to evaluate and prioritise change requests.

The cost of rework is often hidden in the overall project costs, and users and developers have no means to quantify its impact on the overall project productivity rates. Function point analysis enables the project manager to measure the functions that have been reworked due to user-initiated change requests. The results provide valuable feedback to the business on the potential cost savings of committing user resources early in the project to establish an agreed set of requirements and minimising change during the project life-cycle.

COCOMO II

COCOMO II MODELING METHODOLOGY

Preparation includes: Parameter definition, how to use parameters to produce effective estimates. The methodology tries to minimize risk of lost expert time and maximize estimates. Here the question arises whichparameters are significant (most estimate efficient), in which way and with which rating scale.

Seven modeling steps

1. analyse existing literature
2. review software cost modelling literature
3. to insight on improved functional forms potentially significant parameters
4. parameter definition issues (e.g. size)
5. identification of potential new parameters Process Maturity and Multisite Development
6. continuation of a number of parameters from COCOMO I
7. dropping of such COCOMO I parameters as turnaround time and modern programming practices (subsumed by process maturity)

The Bayesian approach was used for COCOMO II and is (will be) reused for COCOTS, COQUALMO, COPSEMOand CORADMO.

THE ROSETTA STONE

The ROSETTA STONE is a system that updates COCOMO I so that it can be used with COCOMO II models. TheROSETTA STONE permits users to translate project files from COCOMO 81 to COCOMO II (backwards compatibility).

EMERGING EXTENSIONS

Because of the rapid changes in software engineering development not all directions of impact could take

place in the COCOMO II model. So some emerging extensions were required to overcome the deficiencies. Allof them are complementary to the COCOMO II model and some of them are still experimental, their calibrationand counting rules aren't robust enough till now. Further research still has to be done.In the following section they are presented briefly. For further particulars please contact the list of references.

The discussed extensions are:

- estimating the cost of software COTS integration (COCOTS)

- Application Composition Model
- phase distributions of schedule and effort (COPSEMO)
- rapid application development effort and schedule adjustments (CORADMO)
- quality in terms of delivered defect density (COQUALMO)
- effects of applying software productivity strategies /improvement (COPROMO)
- System Engineering (COSYSMO)

Top Down and Bottom Up Estimation

There are two approaches to estimating unit costs: top-down, bottom-up, which can be combined to form a 'mixed approach'. Generally, a bottom-up approach is used to estimate the costs of service usage whereas top-down costing is more amenable to estimating the society level costs which are often intangible and where data is scarce.

Top-down unit cost estimation

The top-down approach is based on a simple calculation: divide total expenditure (quantum of funding available) for a given area or policy by total units of activity (e.g. patients served) to derive a unit cost. The units of activity are specific to the services that are being costed, for example the cost of a prison place, GP consultation, or social work assessment. Typically this approach uses aggregate, budgetary data to estimate a unit cost. The advantages of the top-down approach are:

- Availability of data: the availability of budgetary data means that top-down approaches can be applied easily;
- Simplicity: the calculation required to estimate unit costs is easy to understand and direct, providing a simple way to quantify the administrative and overhead costs associated with a range of public services and
- Low cost: the availability of aggregate cost data means that the time and costs required to estimate a top-down unit cost are minimal.

There are, however, two main limitations associated with a top-down approach. First, it does not identify what drives costs and therefore often masks the underlying factors that determine why unit costs vary within a single yet heterogeneous group of service users - for example, children in care. Second, top-down costing cannot be used to reliably forecast how costs might rise or fall as a result of changes in that way that people use services (e.g. the intensity, duration of service usage) or how costs might change due to improvements in outcomes. Therefore, using top-down unit costs may incorrectly estimate the savings from SIB interventions.

Bottom-up unit cost estimation

The bottom-up approach provides a greater level of granularity than the top-down method. It involves identifying all of the resources that are used to provide a service and assigning a value to each of those resources. These values are summed and linked to a unit of activity to derive a total unit cost – this provides a basis for assessment of which costs can be avoided as a result of reduced demand.

The advantages of using a bottom-up approach are:

- Transparency: detailed cost data allows potential errors to be investigated and their impact tested – this facilitates the quality assurance process;
- Granularity: detailed cost data can highlight variations in cost data, and enable practitioners to explore the drivers of variation and determine whether, for example, some service users account for a disproportionate share of costs; and
- Versatility: the methodology enables a practitioner to forecast how costs may change as a result of a reduction in service usage or demand.

Estimating efficiency savings through a bottom-up approach to unit cost estimation is a more robust method of estimating benefits to the commissioner,particuarly those related to time savings and reductions in demand for services. However, the main disadvantage associated with the bottom-up approach is that it is labour intensive; the cost, time and expertise require to apply it may be prohibitive for providers.

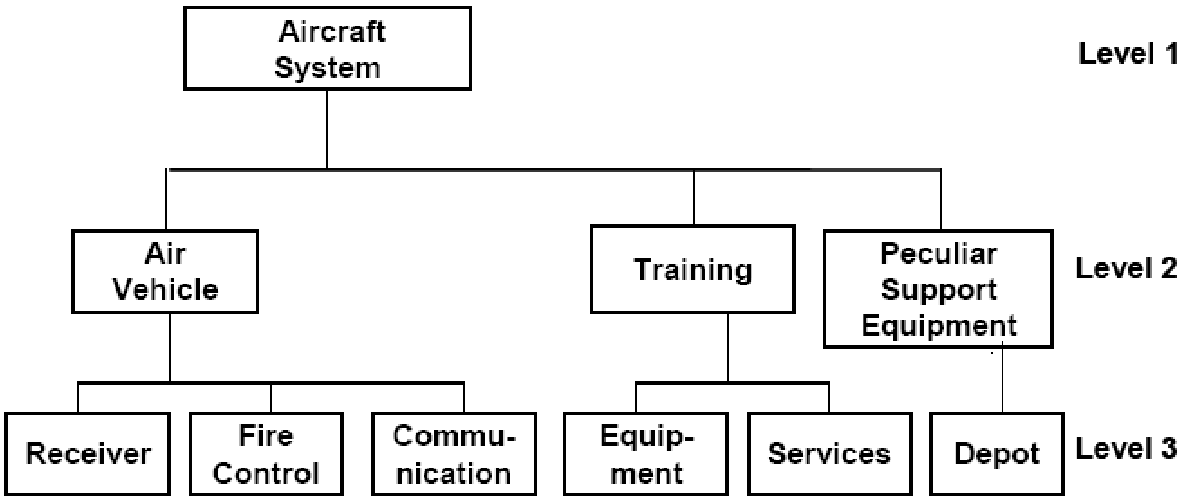
The strengths and weaknesses of top-down and bottom-up unit cost estimation are summarised in the table below. In practice, a combined approach may be most practicable for SIB developers due to time constraints and the availability of cost data. Regardless of which approach is used, the costs avoided through a SIB should take into account how individuals (or groups of individuals) use a service. Service users are heterogeneous and their service usage will reflect this - both in intensity and duration and this will impact on the costs of service provision. Costs should therefore consider the specific cohort to be targeted by a SIB intervention.

Approach to unit cost estimation	**Bottom-Up**	**Top-Down**
Granularity/Transparency	High	Low
Credibility	High	Low
Ease of exploring variation in costs	High	Low
Cost of data collection	Medium-High	Low
Data requirements	High: requires detailed, local data that is often unavailable	Low: budgetary data is often available
Level of Approximation (e.g. assumptions)	Low-Medium	Medium-High
Forecasting (changes in cost following the introduction or redesign of a service)	Medium-High	Low

Work Breakdown Structure

A work breakdown structure (WBS), in project management and systems engineering, is a deliverable-oriented decomposition of a project into smaller components.

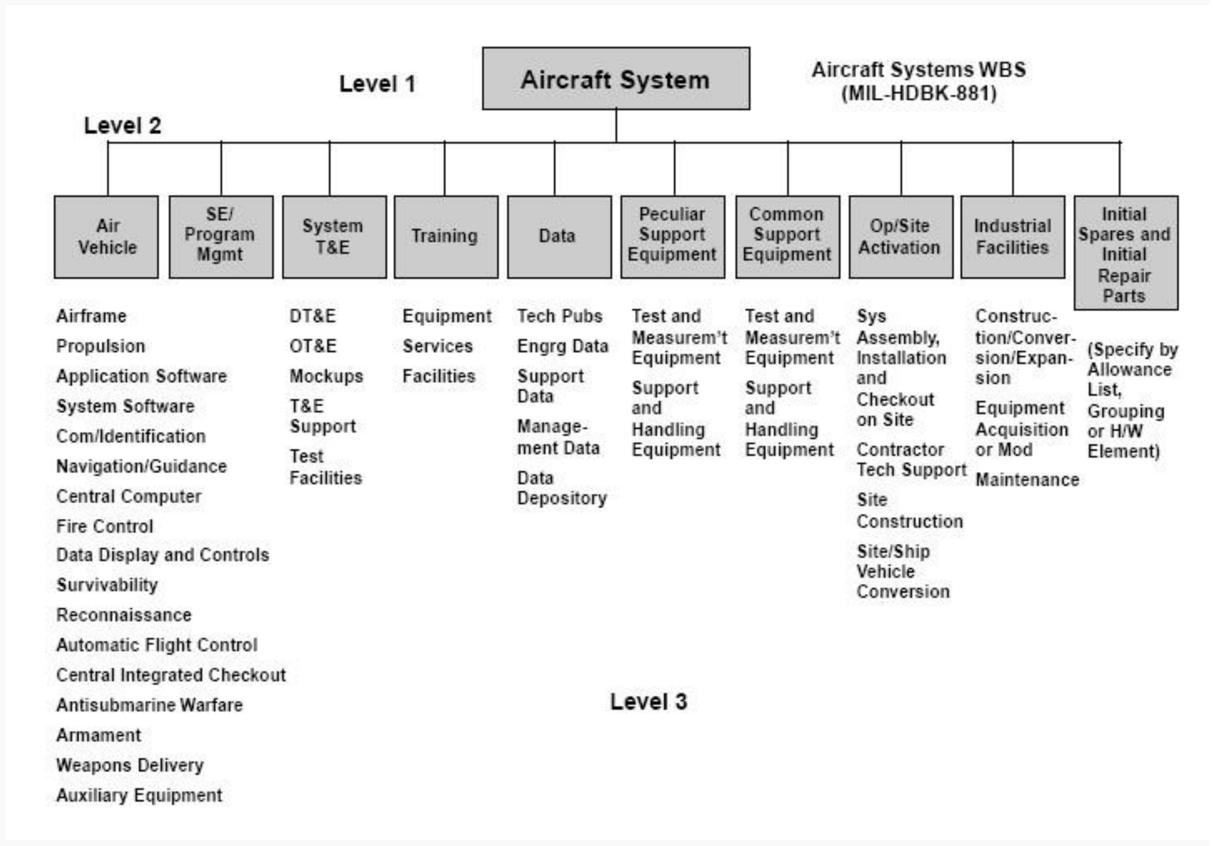
A work breakdown structure element may be a product, data, service, or any combination thereof. A WBS also provides the necessary framework for detailed cost estimating and control along with providing guidance for schedule development and control.



Elements of each WBS Element:

1. The scope of the project, "deliverables" of the project.
2. Start and end time of the scope of project.
3. Budget for the scope of the project.
4. Name of the person related to the scope of project.

.



Example from MIL-HDBK-881, which illustrates the first three levels of a typical aircraft system.[11]

Defense Materiel Item categories from MIL-STD-881C are:

- Aircraft Systems WBS
- Electronic Systems WBS
- Missile Systems WBS
- Ordnance Systems WBS
- Sea Systems WBS
- Space Systems WBS
- Surface Vehicle Systems WBS
- Unmanned Air Vehicle Systems WBS
- Unmanned Maritime Systems WBS
- Launch Vehicle Systems WBS
- Automated Information Systems WBS

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

Steps to developing the estimates

- Start with a Macro estimate then refine with a Micro estimate
- Develop the general project definition
- Perform a macro cost and time estimate
- Develop the detailed project definition and WBS
- Roll-up the WBS elements as part of a micro estimate
- Establish the project schedules
- Reconcile differences between the macro and micro estimates

Macro Estimates

- Scaling:Given a cost for a previous project then an estimate for a new project can be scaled from the known cost. E.g NASA, at times, uses spacecraft weight to estimate total cost.
- Apportion:Given a similar previous project, costs for major subunits of the new project would be proportional to similar subunits in the previous project.
- Weighted Variables:Certain types of projects can be characterized by specific parameters (e.g. number of inputs, number of detector channels). Historical costs & times for single units of these parameters are weighted by the numbers required for the new project.
- Learning Curve:If the same task is repeated a number of times there will be a cost / time savings relative to the first time the task is done.

Micro Estimates

- Template:Uses historical data to establish detailed costs and schedules for project subunits. A new project composed of some combination of these subunits can then be quickly estimated.
- Ratio:Similar to the Macro ratio method but applied to specific tasks associated with project subunits. For example, if it takes 1 day to build & test a particular sensor unit, then an instrument with 10 sensors would take 2 technicians, 5 days to complete.

WBS Roll-up:Times and costs associated with the lowest level WBS work packages are estimated and then these are added or rolled-up to yield the costs for higher level units. This method provides the most accurate estimates at the expense of time devoted to developing the estimate

Planning Poker

Planning poker, also called Scrum poker, is a consensus-based technique for estimating, mostly used to estimate effort or relative size of development goals in software development. In planning poker, members of the group make estimates by playing numbered cards face-down to the table, instead of speaking them aloud. The cards are revealed, and the estimates are then discussed. By hiding the figures in this way, the group can avoid the cognitive bias of anchoring, where the first number spoken aloud sets a precedent for subsequent estimates.

Planning poker is a variation of the Wideband Delphi method. It is most commonly used in agile software development, in particular the Scrum and Extreme Programmingmethodologies.

Process

The reason

The reason to use Planning poker is to avoid the influence of the other participants. If a number is spoken, it can sound like a suggestion and influence the other participants' sizing. Planning poker should force people to think independently and propose their numbers simultaneously. This is accomplished by requiring that all participants show their card at the same time.

Equipment

Planning poker is based on a list of features to be delivered, several copies of a deck of cards and optionally, an egg timer that can be used to limit time spent in discussion of each item.

The feature list, often a list of user stories, describes some software that needs to be developed.

The cards in the deck have numbers on them. A typical deck has cards showing the Fibonacci sequence including a zero: 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89; other decks use similar progressions.

Planning Poker card deck

The reason for using the Fibonacci sequence is to reflect the inherent uncertainty in estimating larger items.Several commercially available decks use the sequence: 0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100, and optionally a ? (unsure) and a coffee cup (I need a break). Some organizations use standard playing cards of Ace, 2, 3, 5, 8 and King. Where King means: "this item is too big or too complicated to estimate." "Throwing a King" ends discussion of the item for the current sprint.

Smartphones allow developers to use apps instead of physical card decks. When teams are not in the same geographical locations,collaborative software can be used as replacement for physical cards.

Procedure

At the estimation meeting, each estimator is given one deck of the cards. All decks have identical sets of cards in them.

The meeting proceeds as follows:

- A Moderator, who will not play, chairs the meeting.
- The Product Manager provides a short overview. The team is given an opportunity to ask questions and discuss to clarify assumptions and risks. A summary of the discussion is recorded by the Project Manager.
- Each individual lays a card face down representing their estimate. Units used vary - they can be days duration, ideal days or story points. During discussion, numbers must not be mentioned at all in relation to feature size to avoid anchoring.
- Everyone calls their cards simultaneously by turning them over.
- People with high estimates and low estimates are given a soap box to offer their justification for their estimate and then discussion continues.
- Repeat the estimation process until a consensus is reached. The developer who was likely to own the deliverable has a large portion of the "consensus vote", although the Moderator can negotiate the consensus.
- To ensure that discussion is structured; the Moderator or the Project Manager may at any point turn over the egg timer and when it runs out all discussion must cease and another round of poker is played. The structure in the conversation is re-introduced by the soap boxes.

The cards are numbered as they are to account for the fact that the longer an estimate is, the more uncertainty it contains. Thus, if a developer wants to play a 6 he is forced to reconsider and either work through that some of the perceived uncertainty does not exist and play a 5, or accept a conservative estimate accounting for the uncertainty and play an 8.

Planning poker benefits

Planning poker is a tool for estimating software development projects. It is a technique that minimizes anchoring by asking each team member to play their estimate card such that it cannot be seen by the other players. After each player has selected a card, all cards are exposed at once.

A study by Moløkken-Østvold and Haugen[6] found that [the] set of control tasks in the same project, estimated by individual experts, achieved similar estimation accuracy as the planning poker tasks. However, for both planning poker and the control group, measures of the median estimation bias indicated that both groups had unbiased estimates, as the typical estimated task was perfectly on target.

Wideband Delphi

The Wideband Delphi estimation method is a consensus-based technique for estimating effort. It derives from the Delphi method which was developed in the 1950-1960s at theRAND Corporation as a forecasting tool. It has since been adapted across many industries to estimate many kinds of tasks, ranging from statistical data collection results to sales and marketing forecasts.

Wideband Delphi Process

Barry Boehm and John A. Farquhar originated the Wideband variant of the Delphi method in the 1970s. They called it "wideband" because, compared to the existing delphi method, the new method involved greater interaction and more communication between those participating. The method was popularized by Boehm's book Software Engineering Economics (1981). Boehm's original steps from this book were:

- Coordinator presents each expert with a specification and an estimation form.
- Coordinator calls a group meeting in which the experts discuss estimation issues with the coordinator and each other.
- Experts fill out forms anonymously.
- Coordinator prepares and distributes a summary of the estimates
- Coordinator calls a group meeting, specifically focusing on having the experts discuss points where their estimates vary widely
- Experts fill out forms, again anonymously, and steps 4 to 6 are iterated for as many rounds as appropriate.

A variant of Wideband Delphi was developed by Neil Potter and Mary Sakry of The Process Group. In this process, a project manager selects a moderator and an estimation team with three to seven members. The Delphi process consists of two meetings run by the moderator. The first meeting is the kickoff meeting, during which the estimation team creates a work breakdown structure (WBS) and discusses assumptions. After the meeting, each team member creates an effort estimate for each task. The second meeting is the estimation session, in which the team revises the estimates as a group and achieves consensus. After the estimation session, the project manager summarizes the results and reviews them with the team, at which point they are ready to be used as the basis for planning the project.

- Choose the team. The project manager selects the estimation team and a moderator. The team should consist of 3 to 7 project team members. The team should include representatives from every engineering group that will be involved in the development of the work product being estimated.
- Kickoff meeting. The moderator prepares the team and leads a discussion to brainstorm assumptions, generate a WBS and decide on the units of estimation.

- Individual preparation. After the kickoff meeting, each team member individually generates the initial estimates for each task in the WBS, documenting any changes to the WBS and missing assumptions.
- Estimation session. The moderator leads the team through a series of iterative steps to gain consensus on the estimates. At the start of the iteration, the moderator charts the estimates on the whiteboard so the estimators can see the range of estimates. The team resolves issues and revises estimates without revealing specific numbers. The cycle repeats until either no estimator wants to change his or her estimate or the estimators agree that the range is acceptable.
- Assemble tasks. The project manager works with the team to collect the estimates from the team members at the end of the meeting and compiles the final task list, estimates and assumptions.
- Review results. The project manager reviews the final task list with the estimation team.

Documenting the Plan

To foster a successful planning phase, here are seven planning documents

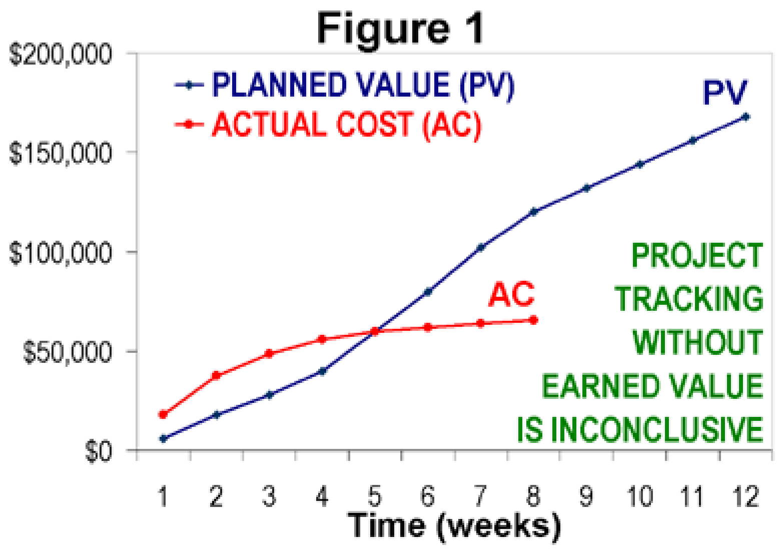
- Project management plan -- This is used as a reference index, encompassing all planning and project documents.
- High-level project schedule plan -- This document captures high-level project phases and key milestones. It is the document most project stakeholders will see or want to see.
- Project team planning -- This document provides a "who-is-doing-what" view of the project. This document fosters efficient project execution and effective project communication.
- Scope plan -- The scope plan documents the project requirements, the agreed scope and the RequirementsTraceabilityMatrix (RTM) summary.
- Detailed project work plan -- This keeps track of the activities, work packages, resources, durations, costs, milestones, project's critical path, etc. It will be an essential document and work guideline for your core project team.
- Quality assurance planning -- This document tracks the quality standards your project deliverables will have to align to. These may typically include product testing approach and tools, quality policies, quality checklists, deviations definitions, quality metrics, product defect severity grades, acceptance criteria, costof poor quality, etc.

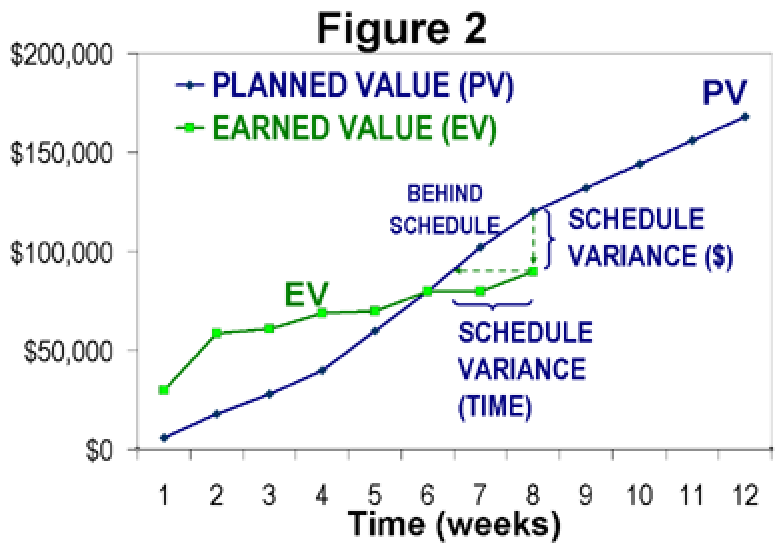
- Risk planning -- This document contains the project risks and the related mitigation plans; as well as the project opportunities and the related exploiting plans. The importance of this document is one of the most underestimated in project planning. Be prepared to have a contingency plan in case something goes wrong or to take advantage of opportunities when they arise.
- Start with this checklist when you sit down to plan for your next project-planning phase. Depending on your project's needs, fine tune the checklist and tailor it by adding and removing planning assets, determining the planning time frame, the underlying details and rigor.

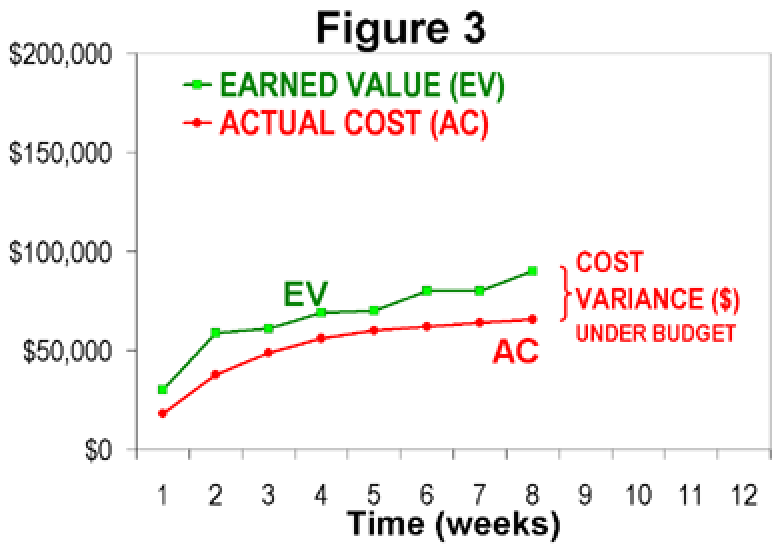
- Revisit this planning exercise, learn from it and enhance it, to continuously improve your project planning skills.

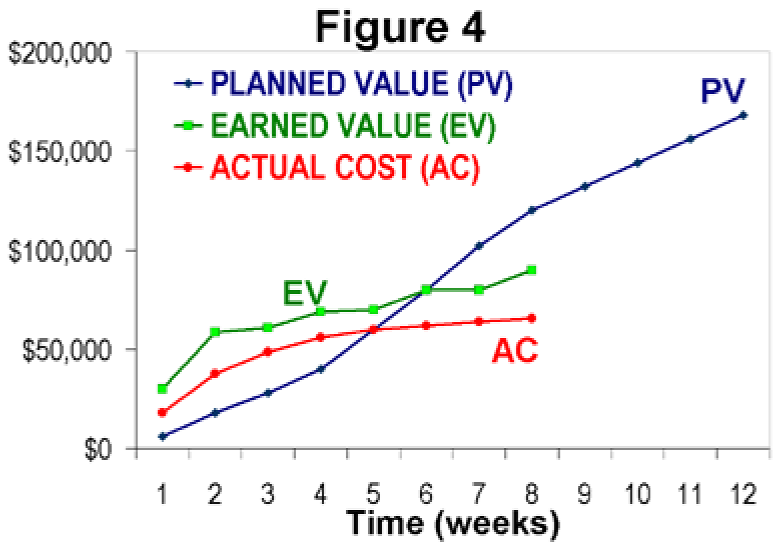
Tracking the Plan

Project tracking









It is helpful to see an example of project tracking that does not include earned value performance management. Consider a project that has been planned in detail, including a time-phased spend plan for all elements of work. Figure 1 shows the cumulative budget (cost) for this project as a function of time (the blue line, labeled PV). It also shows the cumulative actual cost of the project (red line) through week 8. To those unfamiliar with EVM, it might appear that this project was over budget through week 4 and then under budget from week 6 through week 8. However, what is missing from this chart is any understanding of how much work has been accomplished during the project. If the project was actually completed at week 8, then the project would actually be well under budget and well ahead of schedule. If, on the other hand, the project is only 10% complete at week 8, the project is significantly over budget and behind schedule. A method is needed to measure technical performance objectively and quantitatively, and that is what EVM accomplishes.

Project tracking with EVM

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.

Earned value management is a project management technique for measuring project performance and progress. It has the ability to combine measurements of:

- Scope
- Schedule, and
- Costs

In a single integrated system, Earned Value Management is able to provide accurate forecasts of project performance problems, which is an important contribution for project management.

Early EVM research showed that the areas of planning and control are significantly impacted by its use; and similarly, using the methodology improves both scope definition as well as the analysis of overall project performance. More recent research studies have shown that the principles of EVM are positive predictors of project success.[1] Popularity of EVM has grown significantly in recent years beyond government contracting, in which sector its importance continues to rise[2] (e.g., recent new DFARS rules), in part because EVM can also surface in and help substantiate contract disputes.[4]

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 Performed (BCWP).

EVM implementations for large or complex projects include many more features, such as indicators and forecasts of cost performance (over budget or under budget) and schedule performance (behind schedule or ahead of schedule). However, the most basic requirement of an EVM system is that it quantifies progress using PV and EV.

As a short illustration of one of the applications of the EVM consider the following example. Project A has been approved for duration of 1 year and with the budget of X. It was also planned, that after 6 months project will spend 50% of the approved budget. If now 6 months after the start of the project a Project Manager would report that he has spent 50% of the budget, one can initially think, that the project is perfectly on plan. However in reality the provided information is not sufficient to come to such conclusion, as from one side within this time project can spend 50% of the budget, whilst finishing only 25% of the work (which would mean project is not doing well), similarly a project can spend 50% of the budget, whilst completing 75% of the work (which would mean, that project is doing better, than planned). EVM' is meant to address such and similar issues.

Module-IV

CONFIGURATION AND QUALITY MANAGEMENT

Identifying Artifacts to be Configured

Identifying the artifacts that make up an application

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.

Although you can modify an application configuration with the Workbench, we recommend that modifications only be made in the AppConfig.xml file. That way, the application configuration will be saved on disk, ready for sharing between environments. You can use the Workbench for other tasks that do not involve modifying the configuration, such as reviewing the configuration, and starting or stopping individual components.

Note: Some parts of the AppConfig.xml file include settings that are environment specific, such as the application's name, file system paths, and host addresses in the environment. These settings should be collected and stored in a custom file. For more information about how to create this file, see the topic about Creating a custom file for environment-specific settings.

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.

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]/controldirectory.

The Deployment Template includes scripts such as baseline_update and set_baseline_data_ready_flag.

You can create additional scripts under the [appdir]/control directory. These scripts, together with their input data and output directories, are a part of the application definition. For example, a developer might create scripts to crawl web pages in preparation for a baseline update. These scripts might take as input a seed-list file, and create an output file in a custom directory under [appdir].

These command-line scripts, along with their input data and output directories, should be shared among the development, staging and production environments.

Library files

Many parts of an application use library files. For example, a Forge pipeline using a Java or Perl manipulator typically requires access to library files implementing those manipulators. 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.

In most cases these files *do not* need to be included as part of an application definition. However, when an application uses dimension values from auto-generated or external dimensions, then Forge state files *do* need to be synchronized across the environments. In this situation, the state files contain the IDs of these dimension values and ensure that the same dimension value always gets the same ID no matter how many times Forge is run. These dimension values may be used in a variety of ways, including dynamic business rules, landing pages, dimension ordering, and precedence rules.

In other words, Forge state files should be identified as part of the application definition if the application uses auto-generated dimensions or external dimensions, and values from these dimensions are referenced anywhere in the application configuration (for example, in dynamic business rules, Page Builder landing pages, explicit dimension ordering, or in precedence rules).

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

Configuration Item Type	Application Used	Naming Convention	Version Control

Document - Specifications	MS Word	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
			Manual – via version numbering

Configuration Item Type	Application Used	Naming Convention	Version Control

		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
		Reviews: If deliverable review comments and changes are provided within the deliverable itself, i.e. via Track Changes, the reviewer's initials are added to the file name.
		Example:
		HR OHSW-PM QualityPlan v0.1.doc
		Review Example:
		HR OHSW -PM QualityPlan v0.1 PW.doc	

Document – Test Plans	MS Word	As above	Manual – via version numbering
Document – Test Results	MS Word	As above	Manual – via version numbering
Document –	MS Word	As above	Manual – via

Configuration Item Type	Application Used	Naming Convention	Version Control
Project Management			version numbering
Document – Resource Request	MS Word	As Above	Manual – via date versioning
Document – Meeting Minutes	MS Word	As Above	Manual – via date versioning
Presentations	MS Power Point	As Above	Manual – via version numbering
Process Diagrams	Visio	As above	Manual – via version numbering
Organisation Charts	Visio	As Above	Manual – via date versioning
Logs and Registers	Excel	As Above Register held within project Status Report	Manual – via date versioning
Change Requests	MS Word	As Above	Manual – via date versioning
Risks	MS Word	As Above	Manual – via version numbering
Issues	MS Word	As Above	Manual – via version numbering
Defects	TBC		
Assignment Descriptions	MS Word	As Above	Manual – via date versioning
Work Schedules	MS Project or Excel	Descriptive name, e.g. HR OHSW - yyyymmdd	Manual – via date versioning

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’.

If the deliverable version is controlled by major and minor numbering the following process is followed:

1. The Original draft of the deliverable is versioned using zero as the major number and 1 as the minor number, i.e. v0.1
2. For each revision following internal reviews the minor number is incremented, i.e. v0.1 becomes v0.2 and then v0.3 etc.
3. Once the deliverable has completed internal review and is to be distributed to the business representatives for review minor number is incremented.
4. For each revision pertaining to business representatives reviews the minor number is incremented.
5. Following business representative review the deliverable is updated and when ready for acceptance the major number is set to 1 and the minor number set to zero.
6. If the deliverable requires a revision due to changes identified in the acceptance process the major number remains unchanged and the minor number is incremented.
7. The version numbering for changes to deliverables after acceptance follow the same process except that the starting number is the deliverable accepted version number. Upon completion of reviews the major number is incremented by one and the minor number is set to zero, e.g. v1.6 becomes v2.0.

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

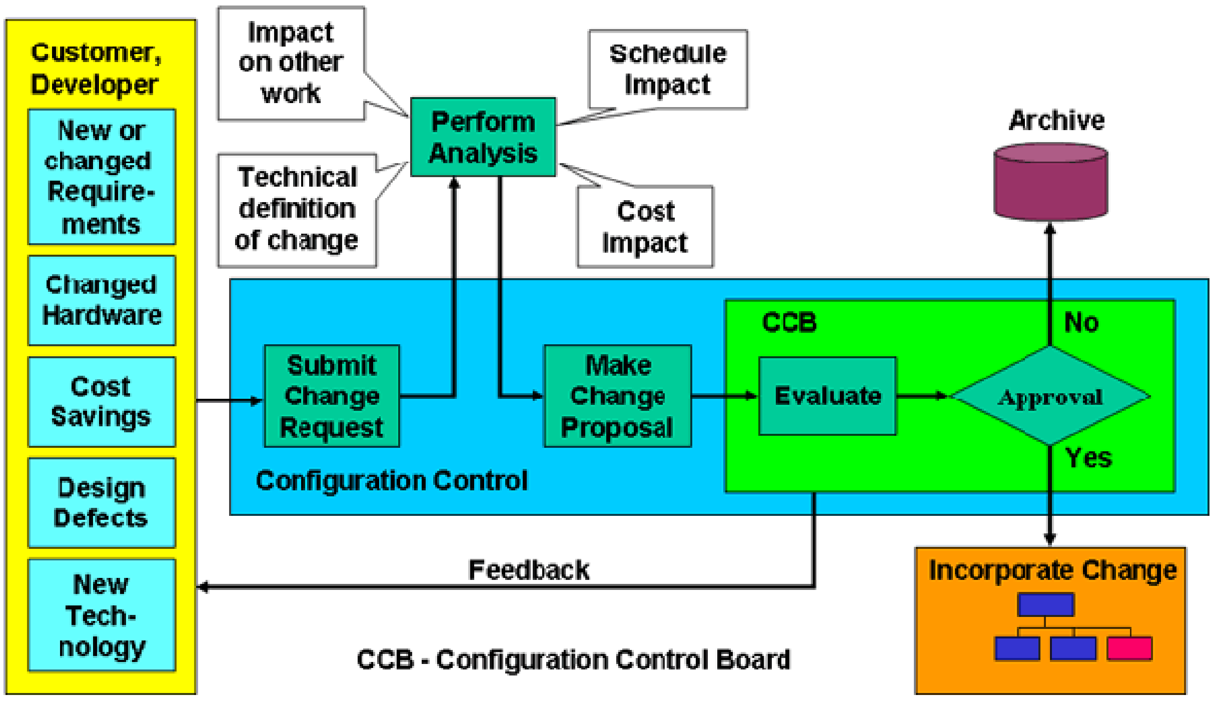
Configuration Control

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 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.

The Configuration Control Process



Why Configuration Control is Important

Configuration control is an essential component of a project's risk management strategy. For example, uncontrolled changes to software requirements introduce the risk of cost and schedule overruns.

Scenario - Curse of the Feature Creep

A project misses several key milestones and shows no sign of delivering anything.

WHY?

- The customer regularly talks directly to software developers asking them to make 'little changes' without consulting the project manager.
- The developers are keen to show off the new technology they are using. They slip in the odd 'neat feature' that they know the customer will love.

Solution: Implement configuration control. Document all requests for change and have them considered by a Configuration Control Board.

Quality Assurance Techniques

SQA

Software Quality Assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. It does this by means of audits of the quality management system under which the software system is created. These audits are backed by one or more standards, usually

ISO 9000.It is distinct from software quality control which includes reviewing requirements documents, and software testing. SQA encompasses the entire software development process, which includes processes such as software design, coding, source code control, code reviews, change management, configuration management, and release management. Whereas software quality control is a control of

products, software quality assurance is a control of processes.

Software quality assurance is related to the practice of quality assurance inproduct manufacturing. There are, however, some notable differences between software and a manufactured product. These differences stem from the fact that the manufactured product is physical and can be seen whereas the software product is not visible. Therefore its function, benefit and costs are not as easily

measured. What’s more, when a manufactured product rolls off the assembly line, it is essentially a complete, finished product, whereas software is never finished.Software lives, grows, evolves, and metamorphoses, unlike its tangiblecounterparts. Therefore, the processes and methods to manage, monitor, andmeasure its ongoing quality are as fluid and sometimes elusive as are the defects that they are meant to keep in check. SQA is also responsible for gathering and presenting software metrics.

For example the Mean Time between Failure (MTBF) is a common softwaremetric (or measure) that tracks how often the system is failing. This SoftwareMetric is relevant for the reliability software characteristic and, by extension theavailability software characteristic.SQA may gather these metrics from various sources, but note the importantpragmatic point of associating an outcome (or effect) with a cause. In this waySQA can measure the value or consequence of having a given standard process,
or procedure. Then, in the form of continuous process improvement, feedbackcan be given to the various process teams (Analysis, Design, Coding etc.) anda process improvement can be initiated.

Overview of methods

Software Quality Assurance takes several forms. A brief list of testing methodsthat should be considered

Methods:

Black box testing - not based on any knowledge of internal design or code.Tests are based on requirements and functionality.

White box testing - based on knowledge of the internal logic of an application'scode. Tests are based on coverage of code statements, branches,paths, conditions

Unit testing - the most 'micro' scale of testing; to test particular functionsor code modules. Typically done by the programmer and not by testers,as it requires detailed knowledge of the internal program design and code.Not always easily done unless the application has a well-designed architecture
with tight code; may require developing test driver modules or testharnesses• Incremental integration testing - continuous testing of an application asnew functionality is added; requires that various aspects of an application'sfunctionality be independent enough to work separately before all parts of
the program are completed, or that test drivers be developed as needed;done by programmers or by testers

Integration testing - testing of combined parts of an application to determineif they function together correctly. The 'parts' can be code modules,individual applications, client and server applications on a network, etc.This type of testing is especially relevant to client/server and distributedsystems

Functional testing - black-box type testing geared to functional requirementsof an application; this type of testing should be done by testers.This doesn't mean that the programmers shouldn't check that their codeworks before releasing it (which of course applies to any stage of testing)

System testing - black-box type testing that is based on overall requirementsspecifications; covers all combined parts of a system

End-to-end testing - similar to system testing; the 'macro' end of the testscale; involves testing of a complete application environment in a situationthat mimics real-world use, such as interacting with a database, usingnetwork communications, or interacting with other hardware, applications,or systems if appropriate

• **User acceptance testing -** determining if software is satisfactory to an end user or customer

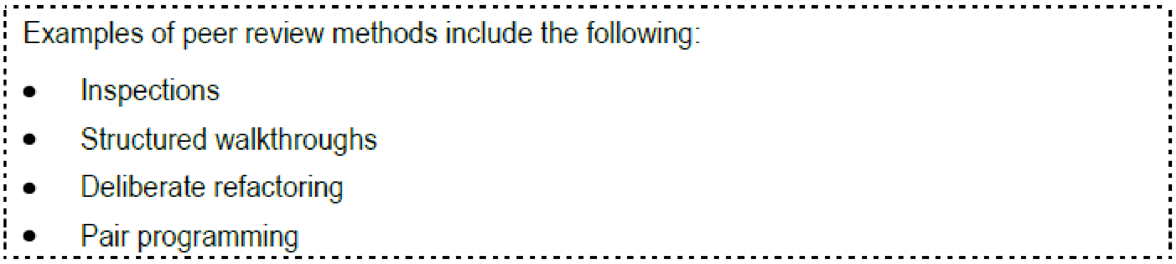
Peer Reviews

- Does not dictate specific techniques, but instead requires that:
  - A written policy about peer reviews is required
  - Resources, funding, and training must be provided
  - Peer reviews must be planned
  - The peer review procedures to be used must be documented

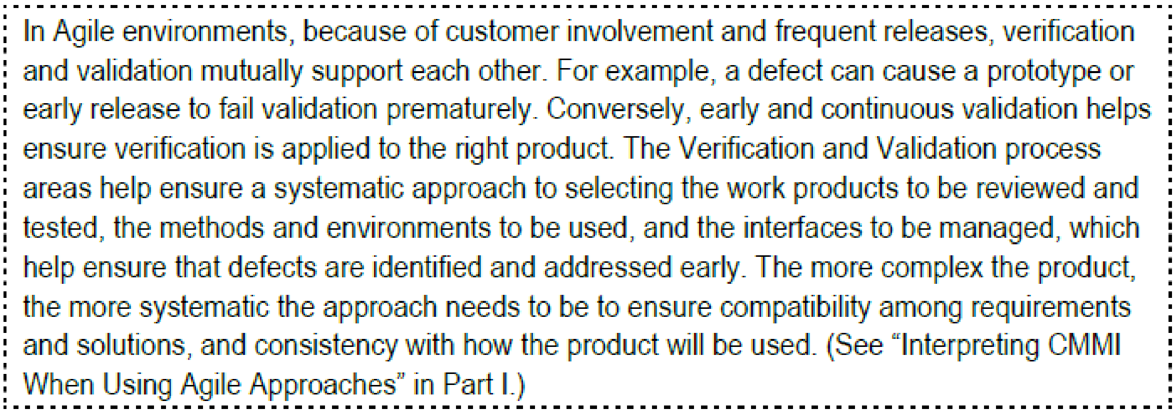
SEI-CMMI Checklist for Peer Reviews

- Are peer reviews planned?
- Are actions associated with defects that are identified during peer reviews tracked until they are resolved?
- Does the project follow a written organizational policy for performing peer reviews?
- Do participants of peer reviews receive the training required to perform their roles?
- Are measurements used to determine the status of peer review activities?
- Are peer review activities and work products subjected to Software Quality Assurance review and audit?

Peer review is a traditional organizational function designed to contribute to improving the quality of care and appropriate utilization of health care resources.


Examples of peer review methods include the following:

- Inspections
- Structured walkthroughs
- Deliberate refactoring
- Pair programming


In Agile environments, because of customer involvement and frequent releases, verification and validation mutually support each other. For example, a defect can cause a prototype or early release to fail validation prematurely. Conversely, early and continuous validation helps ensure verification is applied to the right product. The Verification and Validation process areas help ensure a systematic approach to selecting the work products to be reviewed and tested, the methods and environments to be used, and the interfaces to be managed, which help ensure that defects are identified and addressed early. The more complex the product, the more systematic the approach needs to be to ensure compatibility among requirements and solutions, and consistency with how the product will be used. (See "Interpreting CMMI When Using Agile Approaches" in Part I.)

Peer Review

- Reviews performed by peers in the development team
  - Can be from Fagan's inspections to simple buddy checks

– Peer Review Items
– Participants / Roles
– Schedule

Fegan Inspection

Researchers and Influencers

- Fagan
- Johnson
- Ackermann
- Gilb and Graham
- Weinberg
- Weigers

Inspection, Walkthrough or Review?

An inspection is ‘a visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications’

A walkthrough is ‘a static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems’

A review is ‘a process or meeting during which a software product is presented to project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval’

Families of Review Methods

Method Family	**Typical Goals**	**Typical Attributes**
Walkthroughs	Minimal overhead Developer training Quick turnaround	Little/no preparation Informal process No measurement Not FTR!
Technical Reviews	Requirements elicitation Ambiguity resolution Training	Formal process Author presentation Wide range of discussion

Inspections	Detect and remove all defects efficiently and effectively	Formal process Checklists Measurements Verify phase

Informal vs. Formal

- Informal
  - Spontaneous
  - Ad-hoc
  - No artifacts produced
- Formal
  - Carefully planned and executed
  - Reports are produced

In reality, there is also a middle ground between informal and formal techniques

Cost-Benefit Analysis

- Fagan reported that IBM inspections found 90% of all defects for a 9% reduction in average project cost
- Johnson estimates that rework accounts for 44% of development cost
- Finding defects, finding defects early and reducing rework can impact the overall cost of a project

Cost of Defects

What is the impact of the annual cost of software defects in the US?

$59 billion

Estimated that $22 billion could be avoided by introducing a best-practice defect detection infrastructure

- Gilb project with jet manufacturer
- Initial analysis estimated that 41,000 hours of effort would be lost through faulty requirements
- Manufacturer concurred because:
  - 10 people on the project using 2,000 hours/year
  - Project is already one year late (20,000 hours)
  - Project is estimated to take one more year (another 20,000 hours)

Software Inspections

Why are software inspections not widely used?

- Lack of time
- Not seen as a priority

- Not seen as value added (measured by loc)
- Lack of understanding of formalized techniques
- Improper tools used to collect data
- Lack of training of participants
- Pits programmer against reviewers

Twelve Reasons Conventional Reviews are Ineffective

1. The reviewers are swamped with information.
2. Most reviewers are not familiar with the product design goals.
3. There are no clear individual responsibilities.
4. Reviewers can avoid potential embarrassment by saying nothing.
5. The review is a large meeting; detailed discussions are difficult.
6. Presence of managers silences criticism.
7. Presence of uninformed reviewers may turn the review into a tutorial.
8. Specialists are asked general questions.
9. Generalists are expected to know specifics.
10. The review procedure reviews code without respect to structure.
11. Unstated assumptions are not questioned.
12. Inadequate time is allowed.

Fagan’s Contributions

- Design and code inspections to reduce errors in program development (1976)
- A systematic and efficient approach to improving programming quality
- Continuous improvement: reduce initial errors and follow-up with additional improvements
- Beginnings of formalized software inspections

Fagan’s Six Major Steps

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

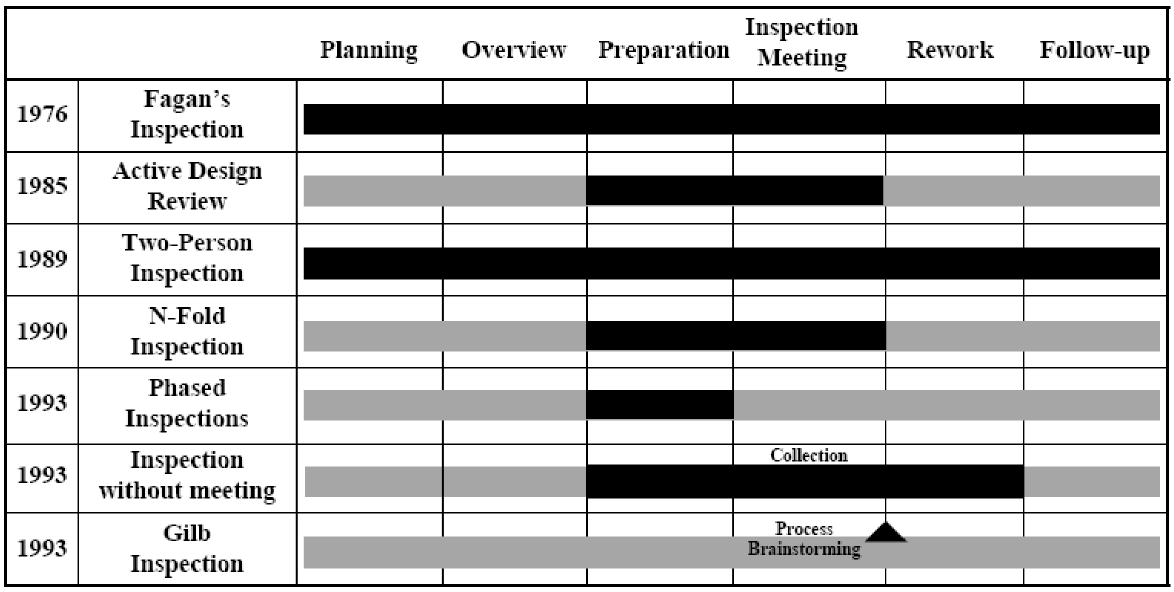
1. Planning: Form team, assign roles
2. Overview: Inform team about product (optional)
3. Preparation: Independent review of materials
4. Examination: Inspection meeting
5. Rework: Author verify defects and correct

6. Follow-up: Moderator checks and verifies corrections

Fagan's Team Roles

- Fagan recommends that a good size team consists of four people
- Moderator: the key person, manages team and offers leadership
- Readers, reviewers and authors
  - Designer: programmer responsible for producing the program design
  - Coder/ Implementer: translates the design to code
  - Tester: write, execute test cases

Common Inspection Processes



Unit , Integration , System , and Acceptance Testing

Testing methods

Testing levels

There are generally four recognized levels of tests: unit testing, integration testing, system testing, and acceptance testing. Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model.[32] Other test levels are classified by the testing objective.

Unit testing

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to ensure that the building blocks of the software work independently from each other.

Unit testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Rather than replace traditional QA focuses, it augments it. Unit testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process.

Depending on the organization's expectations for software development, unit testing might include static code analysis, data flow analysis, metrics analysis, peer code reviews, code coverage analysis and other software verification practices.

Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

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. The idea is to make them more cautious when developing their code or making additional changes. Some companies have a higher-level document called a test strategy.

Traceability matrix

A traceability matrix is a table that correlates requirements or design documents to test documents. It is used to change tests when related source documents are changed, to select test cases for execution when planning for regression tests by considering requirement coverage.

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.[47] This can be as pragmatic as 'for condition x your derived result is y', whereas other test cases described in more detail the input scenario and what results might be expected. It can occasionally be a series of steps (but often steps are contained in a separate test procedure that can be exercised against multiple test cases, as a matter of economy) but with one expected result or expected outcome. The optional fields are a test case ID, test step, or order of execution number, related requirement(s), depth, test category, author, and check boxes for whether the test is automatable and has been automated. Larger test cases may also contain prerequisite states or steps, and descriptions. A test case should also contain a place for the actual result. These steps can be stored in a word processor document, spreadsheet, database, or other common repository. In a database system, you may also be able to see past test results, who generated the results, and what system configuration was used to generate those results. These past results would usually be stored in a separate table.

Test script

A test script is a procedure, or programing code that replicates user actions. Initially the term was derived from the product of work created by automated regression test tools. 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. It definitely contains a section where the tester identifies the system configuration used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests.

Test fixture or 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.

Test harness

The software, tools, samples of data input and output, and configurations are all referred to collectively as a test harness.

Bug Tracking

Bug tracking systems as a part of integrated project management systems

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.

Distributed bug tracking

Some bug trackers are designed to be used with distributed revision control software. These distributed bug trackers allow bug reports to be conveniently read, added to the database or updated while a developer is offline.[3] Fossil and Veracity both include distributed bug trackers.

Recently, commercial bug tracking systems have also begun to integrate with distributed version control. FogBugz, for example, enables this functionality via the source-control tool, Kiln.

Although wikis and bug tracking systems are conventionally viewed as distinct types of software, ikiwiki can also be used as a distributed bug tracker. It can manage documents and code as well, in an integrated distributed manner. However, its query functionality is not as advanced or as user-friendly as some other, non-distributed bug trackers such asBugzilla.[5] Similar statements can be made about org-mode, although it is not wiki software as such.

Bug tracking and test management

While traditional test management tools such as HP Quality Center 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 and 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.

• Taking actions to complete the following:

• Remove causes and prevent the recurrence of those types of defects and problems in the future

• Proactively analyze data to identify potential problems and prevent them from occurring

• Incorporate the causes of successes into the process to improve future process performance

Reliance on detecting defects and problems after they have been introduced is not cost effective. It is more effective to prevent defects and problems by integrating Causal Analysis and Resolution activities into each phase of the project.Since similar outcomes may have been previously encountered in other projects or in earlier phases or tasks of the current project, Causal Analysis and Resolution activities are mechanisms for communicating lessons learned among projects.

Types of outcomes encountered are analyzed to identify trends. Based on an understanding of the defined process and how it is implemented, root causes of these outcomes and future implications of them are determined.

Since it is impractical to perform causal analysis on all outcomes, targets are selected by tradeoffs on estimated investments and estimated returns of quality, productivity, and cycle time.

Measurement and analysis processes should already be in place. Existing defined measures can be used, though in some instances new measurement definitions, redefinitions, or clarified definitions may be needed to analyze the effects of a process change.

Refer to the Measurement and Analysis process area for more information about aligning measurement and analysis activities and providing measurement results.

Causal Analysis and Resolution activities provide a mechanism for projects to evaluate their processes at the local level and look for improvements that can be implemented.

When improvements are judged to be effective, the information is submitted to the organizational level for potential deployment in the organizational processes.

The specific practices of this process area apply to a process that is selected for quantitative management. Use of the specific practices of this process area can add value in other situations, but the results may not provide the same degree of impact to the organization’s quality and process performance objectives.

Module –V
SOFTWARE PROCESS DEFINITION AND MANAGEMENT

Process Elements

Accountability

- Each   deliverables has an owner and a customer (i.e Requirements is owned by product manager.
- The lead architect is the customer).
- Each engineer or group of engineer responsible for a set of features or components should support
- them during Beta and early releases TeamSpirit
- Collaboration and proper diffusion of knowledge and information is key to productivity
- Management should intervene only in case of deviation from standards and baseline
- Performance Metrics

Overview

- The basic process is to define a baseline for key performance metrics following a clear but flexible set of procedures.
- Alerts , analysis and corrective/contingency actions should be planed when deviation from the baseline occurs.

Quality Coverage

- Each development artifacts (Requirements, Use Cases, Design, Source Code, Test cases, Unit Black-
- box Test, Build, Release, Usability) must have one or more quality criteria such as % approved or
- compliance deliverables.
- The overall quality index could be weighted by deliverables with higher weights with early deliverables which has more impact on the overall acceptance of the product by the customer.

Functionality Coverage

- Percentage of requirements validated by customers
- Percentage of user scenario validated by customers
- Productivity
- Ratio of failed build, release, test cycle, reworked requirements
- Accuracy of initial estimation
- Defects turnaround time
- Customer escalation turnaround time
- Activity-based Costing
- Average cost to fix a defect
- Average cost of a failed build
- Average cost of failed release
- Average cost of incorrect requirement
- Quality Function Deployment (QFD)

Probes are used to collect statistics on the development process

- API or Web services for statistics
- Alerts on deviation
- Review & Traceability

Here is a sample of procedures for the software development process:

- Creating/updating coding standards
- Creating/updating design guideline
- Creating/updating requirements
- Creating/updating use cases
- Creating/updating build scripts
- Creating/updating test cases
- Creating/updating user documentation
- Creating/updating training guide
- Unit testing
- Black-box testing
- Defect resolution
- Data Integrity tests
- Performance testing
- Functional testing
- Configuration management test lab
- Test automation
- Build automation
- Release automation
- Requirements review
- Code review
- Design review
- Test cases/plan review
- Build
- Release
- Customer escalation
- Update Product Roadmap
- Build vs buy
- Hiring
- Engineering Suggestions
- Reusability of components
- Orientation new engineer
- Hands-on training
- Management by objectives
- Performance appraisal
- Source code management
- Process Management
- Project management
- Selection of offshore teams
- Synchronization development context with offshore teams

Here is a sample list of automation tools that can be used in software engineering

- Build environment/portal
- Requirements tracking system
- Defect database
- Test configuration management
- Development environment
- Test harness
- Unit Test generator
- Execution profiler
- Code coverage analyzer
- Black-box automation tool
- Version control
- Collaboration platform

Process Architecture

Process architecture is the structural design of general process systems and applies to fields such as computers (software, hardware, networks, etc.), business processes (enterprise architecture, policy and procedures, logistics, project management, etc.), and any other process system of varying degrees of complexity.[1]

Processes are defined as having inputs, outputs and the energy required to transform inputs to outputs. Use of energy during transformation also implies a passage of time: a process takes real time to perform its associated action. A process also requires space for input/output objects and transforming objects to exist: a process uses real space.

A process system is a specialized system of processes. Processes are composed of processes. Complex processes are made up of several processes that are in turn made up of several processes. This results in an overall structural hierarchy of abstraction. If the process system is studied hierarchically, it is easier to understand and manage; therefore, process architecture requires the ability to consider process systems hierarchically. Graphical modeling of process architectures is considered by Dualistic Petri nets. Mathematical consideration of process architectures may be found in CCS and the π-calculus.

The structure of a process system, or its architecture, can be viewed as a dualistic relationship of its infrastructure and suprastructure.[1][2] The infrastructure describes a process system's component parts and their interactions. The suprastructure considers the super system of which the process system is a part. (Suprastructure should not be confused with superstructure, which is actually part of the infrastructure built for (external) support.) As one traverses the process architecture from one level of abstraction to the next, infrastructure becomes the basis for suprastructure and vice versa as one looks within a system or without.

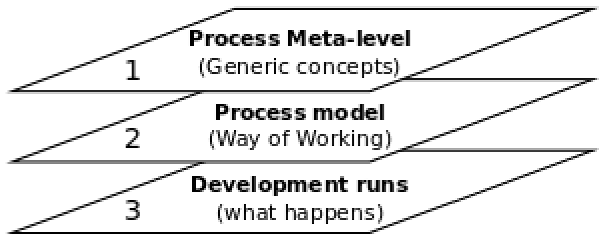
Requirements for a process system are derived at every hierarchical level.[2] Black-box requirements for a system come from its suprastructure. Customer requirements are black-box requirements near, if not at, the top of a process architecture's hierarchy. White-box requirements, such as engineering rules, programming syntax, etc., come from the process system's infrastructure.

Process systems are a dualistic phenomenon of change/no-change or form/transform and as such, are well-suited to being modeled by the bipartite Petri Nets modeling system and in particular, process-class Dualistic Petri nets where processes can be simulated in real time and space and studied hierarchically.

Relationship Between Elements

Process Modeling

The term process model is used in various contexts. For example, in business process modeling the enterprise process model is often referred to as the business process model.



Process models are processes of the same nature that are classified together into a model. Thus, a process model is a description of a process at the type level. Since the process model is at the type level, a process is an instantiation of it. The same process model is used repeatedly for the development of many applications and thus, has many instantiations. One possible use of a process model is to prescribe how things must/should/could be done in contrast to the process itself which is really what happens. A process model is roughly an anticipation of what the process will look like. What the process shall be will be determined during actual system development.[2]

The goals of a process model are to be:

Descriptive

- Track what actually happens during a process
- Take the point of view of an external observer who looks at the way a process has been performed and determines the improvements that must be made to make it perform more effectively or efficiently.

Prescriptive

- Define the desired processes and how they should/could/might be performed.
- Establish rules, guidelines, and behavior patterns which, if followed, would lead to the desired process performance. They can range from strict enforcement to flexible guidance.

Explanatory

- Provide explanations about the rationale of processes.
- Explore and evaluate the several possible courses of action based on rational arguments.

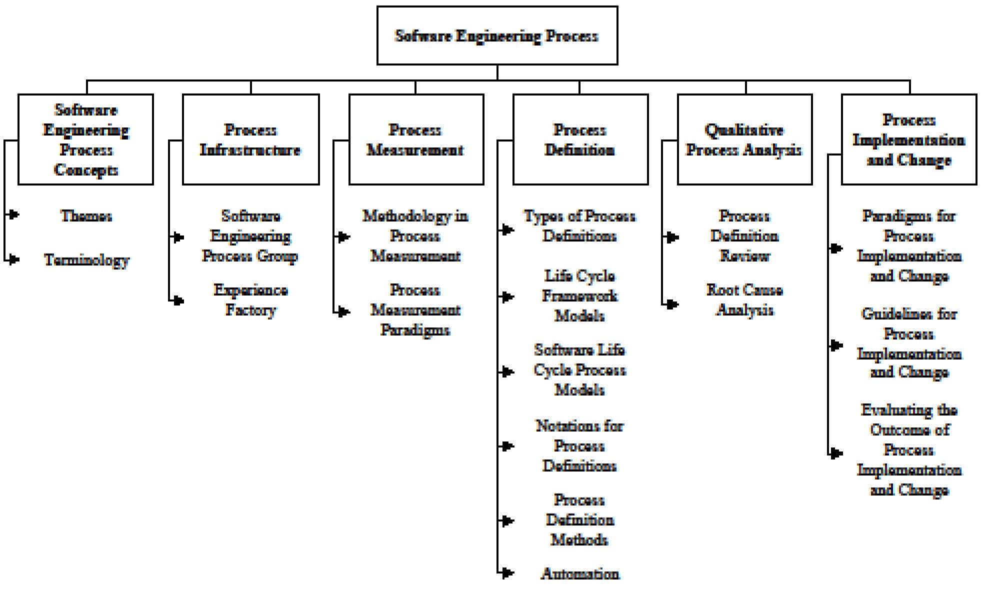
- Establish an explicit link between processes and the requirements that the model needs to fulfill.
- Pre-defines points at which data can be extracted for reporting purposes.

Process Definition Techniques

Software Engineering Process Concepts

Themes

Dowson [35] notes that “All process work is ultimately directed at ‘software process assessment and improvement’”. This means that the objective is to implement new or better processes in actual practices, be
they individual, project or organizational practices.


Sofware Engineering Process
Software Engineering Process Concepts
Themes
Terminology
Process Infrastructure
Software Engineering Process Group
Experience Factory
Process Measurement
Methodology in Process Measurement
Process Measurement Paradigms
Process Definition
Types of Process Definitions
Life Cycle Framework Models
Software Life Cycle Process Models
Notations for Process Definitions
Process Definition Methods
Automation
Qualitative Process Analysis
Process Definition Review
Root Cause Analysis
Process Implementation and Change
Paradigms for Process Implementation and Change
Guidelines for Process Implementation and Change
Evaluating the Outcome of Process Implementation and Change

We describe the main topics in the software process engineering (i.e., the meta-level that has been alluded to
earlier) area in terms of a cycle of process change, based on the commonly known PDCA cycle. This cycle highlights that individual process engineering topics are part of a larger process to improve practice, and that process
evaluation and feedback is an important element of process engineering.

Software process engineering consists of four activities as illustrated in the model in **Figure 1**. The activities are
sequenced in an iterative cycle allowing for continuous feedback and improvement of the software process.

The “Establish Process Infrastructure” activity consists of establishing commitment to process implementation and
change (including obtaining management buy-in), and putting in place an appropriate infrastructure (resources and

responsibilities) to make it happen. The activities “Planning of Process Implementation and Change” and “Process Implementation and Change” are the core ones in process engineering, in that they are essential for any long-lasting benefit from process engineering to accrue. In the planning activity the objective is to understand the current business objectives and process needs of the organization1, identify its strengths and weaknesses, and make a plan for process implementation and change. In “Process Implementation and Change”, the objective is to execute the plan, deploy new processes (which may involve, for example, the deployment of tools and training of staff), and/or change existing processes. The fourth activity, “Process Evaluation” is concerned with finding out how well the implementation and change went; whether the expected benefits materialized. This is then used as input for subsequent cycles. At the centre of the cycle is the “Process Experience Base”. This is intended to capture lessons from past iterations of the cycle (e.g., previous evaluations, process definitions, and plans). Evaluation lessons can be qualitative or quantitative. No assumptions are made about the nature or technology of this “Process Experience Base”, only that it be a persistent storage. It is expected that during subsequent iterations of the cycle, previous experiences will be adapted and reused. It is also important to continuously re-assess the utility of information in the experience base to ensure that obsolete information does not accumulate. With this cycle as a framework, it is possible to map the topics in this knowledge area to the specific activities where they would be most relevant. This mapping is also shown in **Figure 1**. The bulleted boxes contain the Knowledge Area topics. It should be noted that this cycle is not intended to imply that software process engineering is relevant to only large organizations. To the contrary, process-related activities can, and have been, performed successfully by small organizations, teams, and individuals. The way the activities defined in the cycle are performed would be different depending on the context. Where it is relevant, we will present examples of approaches for small organizations.

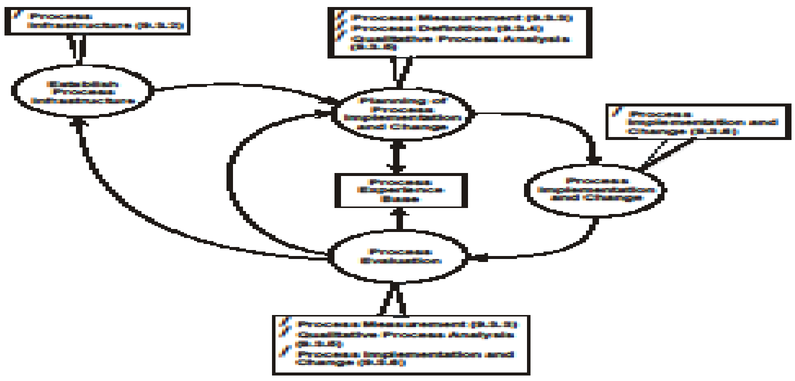

Process Infrastructure
Establish Process Infrastructure
Process Measurement
Process Definition
Qualitative Process Analysis
Planning of Process Implementation and Change
Process Implementation and Change
Process Experience Base
Process Implementation and Change
Process Evaluation
Process Measurement
Qualitative Process Analysis
Process Implementation and Change

Figure 1 A model of the software process engineering cycle, and the relationship of its activities to the KA topics.

The circles are the activities in the process engineering cycle. The square in the middle of the cycle is a data store.

The bulleted boxes are the topics in this Knowledge Area that map to each of the activities in the cycle. The numbers refer to the topic sections in this chapter. The topics in this KA are as follows:
Process Infrastructure: This is concerned with putting in place an infrastructure for software process engineering.
Process Measurement: This is concerned with quantitative techniques to diagnose software processes; to identify strengths and weaknesses. This can be performed to initiate process implementation and change, and afterwards to evaluate the consequences of process implementation and change.
Process Definition: This is concerned with defining processes in the form of models, plus the automated support that is available for the modeling task, and for enacting the models during the software process.
Process Definition
Software engineering processes are defined for a number ofreasons, including: facilitating human understanding andcommunication, supporting process improvement,supporting process management, providing automated
process guidance, and providing automated executionsupport .The types of process definitions
required will depend, at least partially, on the reason.It should be noted also that the context of the project and
organization will determine the type of process definition that is most important. Important variables to consider
include the nature of the work (e.g., maintenance ordevelopment), the application domain, the structure of the
delivery process (e.g., waterfall, incremental, evolutionary),and the maturity of the organization.There are different approaches that can be used to defineand document the process. Under this topic the approachesthat have been presented in the literature are covered,although at this time there is no data on the extent to which
these are used in practice.

Etvx (Entry Task Validation Exit)

(E)ntry Criteria

- Business Requirements available for review

(T)asks

- Review Business Requirements
- Log all discrepancies/ questions in Bug Tracker
- Provide High-level Test Estimates

(V)alidation

- All items in the bug tracker are closed with valid comments
- Business Requirements document is updated

(E)xit

- Signed-off Business Requirement Document

- High-level Test estimates are accepted by all stake holders

<u>**Design Phase**</u>

(E)ntry Criteria

- Signed-off Business Requirements
- Functional (FS) & Technical(TS) Specifications available for review

(T)asks

- Review FS & TS
- Raise questions/discrepancies in Bug Tracker to clarify if anything is ambiguous
- Provide detailed Test Estimates
- Review project plan and provide feedback to Program Manager
- Create master test plan & automation test plan and get it reviewed by appropriate stakeholders
- Generate traceability matrix
- Collaborate with all test teams to have complete coverage on integration areas

(V)alidation

- All items in the bug tracker are in closed state
- All review comments of master test plan and automation test plan are closed

(E)xit

- Signed-off TS
- Signed-off FS
- Detailed Test Estimates are accepted by all stake holders and incorporated appropriately in schedule
- Signed-off project plan
- Signed-off MTP
- Signed-off ATP

<u>**Build/Coding Phase**</u>

(E)ntry Criteria

- Signed-off TS
- Signed-off FS
- Signed-off MTP
- Signed-off ATP
- Availability of Test Environment details

(T)asks

- Write test cases to cover entire functionality and affected areas both from UI and DB perspective
- Get the test cases reviewed by appropriate stakeholders and get sign off
- Work with Operations /Support team to get test environments
- Validate SQL scripts against test cases in Dev/Unit Test environments

(V)alidation

- Test cases review

- Sanity check of test environments
- Sanity check of SQL scripts and UI automation scripts

(E)xit

- Signed-off test cases
- SQA environments are available for build deployment
- Validated SQL scripts against test cases
- Validated UI automation scripts

<u>Stabilization/Testing Phase</u>

(E)ntry Criteria

- Signed-off test cases
- Validated UI automation scripts
- Validated SQL scripts against test cases
- SQA environments are available for build deployment

(T)asks

- Execute test cases both manual and automation
- Publish daily report with test cases execution progress as well updated info on bugs
- Raise bugs in Bug Tracker appropriately
- Track bugs to closure
- Collaborate with all test teams to have complete coverage on integration areas

(V)alidation

- Execution of test cases is completed
- All appropriate bugs are tracked to closure

(E)xit

All the BVTs passed on all the builds/patches

Code Complete Build:

Test should be able to execute 100% of test cases by end of code freeze build and 80% of test cases should pass

By start of Test final build :

- All failed test cases of Code complete build + other test cases planned should pass.
- All bugs raised during the Code complete build execution should be resolved & closed
- No S1 & S2 bugs should be in proposed/active/resolved state
- Active S3 & S4 bugs count should be within permissible limit like 5% of total bugs or moved to future releases
- Any known issues with technical constraints or anything should have an agreed resolution
- Test Final Build:
- All planned test cases for final build should pass

- No S1 & S2 bugs should be in proposed/active/resolved state
- No S3 & S4 bugs should be in proposed/active/resolved state

Process Baselining

There are many different steps that organizations follow in benchmarking. However, most baselining processes have these four steps:

- Develop a clearly defined baseline in your organization: This means that all of the attributes involved in your baseline are defined. In our example of defects per lines of code, clearly defining what is meant by defect and a line of code would meet the objective of this step.
- Identify the organizations you desire to baseline against: Many factors come into this decision, such as do you want to benchmark within your industry, do you want to benchmark what you believe are leading organizations, do you want to benchmark an organization that uses the same tools that are used in your organization, and do you want to benchmark against organizations with a similar culture.
- Compare baseline calculations: Compare how your baseline is calculated versus the baseline calculation in the company you want to benchmark against. Benchmarking is only effective when you benchmark against an organization who has calculated their baseline using approximately the same approach that your organization used to calculate the baseline.
- Identify the cause of baseline variance in the organization you benchmarked against: When you find a variance between the baseline calculation in your company and the baseline calculation in the organization you are benchmarking against, you need to identify the cause of variance. For example, if your organization was producing 20 defects per thousand lines of code, and you benchmarked against an organization that only had 10 defects per thousand lines of code you would want to identify the cause of the difference. If you cannot identify the cause of difference, there is little value in benchmarking. Let us assume that the company you benchmarked against had a different process for requirement definition than your organization. For example, assume they use JAD (joint application development) and you did not. Learning this, you may choose to adopt JAD in your organization as a means for reducing your developmental defect rates.

Process Assessment and Improvement

Software process improvement (SPI) started in the 1990s from the process based approach to software development. The main problem of product-centred development was the ignoring of activities that had no visible results and regarding them as unimportant. Process-based approach in software development puts emphasis on organisation development and the reaching of business goals. A similar understanding of software processes creates a feeling of unity among the developers in a company and a continuity in the development, that in turn guarantee higher production capability and quality of the results.

As the first promoter of software process improvement Watts Humphrey said, the main problems in software development are not caused by insufficient skills, but by unawareness of how to use the best available methods and inability to efficiently solve detailed problems related to the process and product. The result of software process improvement will be the following of a detailed description of activities in every situation of development.

Software process improvement begins with an assessment of software processes. Different process models and standards have been created for software process assessment. We employ the two most widely used models CMMI or Capability Maturity Model Integration and ISO/IEC 15504 (SPICE – Software Process Improvement and Capability Determination). The processes related to development are assessed on the basis of a benchmark model, i.e. each specific process to be assessed in relation to the development will be compared to the requirements described in the benchmark model, at the same time taking into account also the unique character of the development company or a project. Process assessment is project-based – a recently ended project is chosen and the activities related to its development will be evaluated.

Assessment starts with a meeting for developers that introduces software process improvement and assessment and in the course of which the project as well as the processes are chosen that are considered by developers to be the most important for assessment. Often these tend to be the processes that the developers consider to have been insufficient in several projects. The detailed assessment of the chosen project and processes will be done during an interview with the developers who took part in the project. Software process assessment also ends with a development team meeting, where the capabilities of the assessed processes are described according to process models, as well as the shortcomings of the processes and a process improvement plan is put together. Despite the fact that the processes are assessed in a specific project, the development processes will be improved in each of the following development projects. It is only constant improvement that leads to the software productivity growth in a company.

CMMI

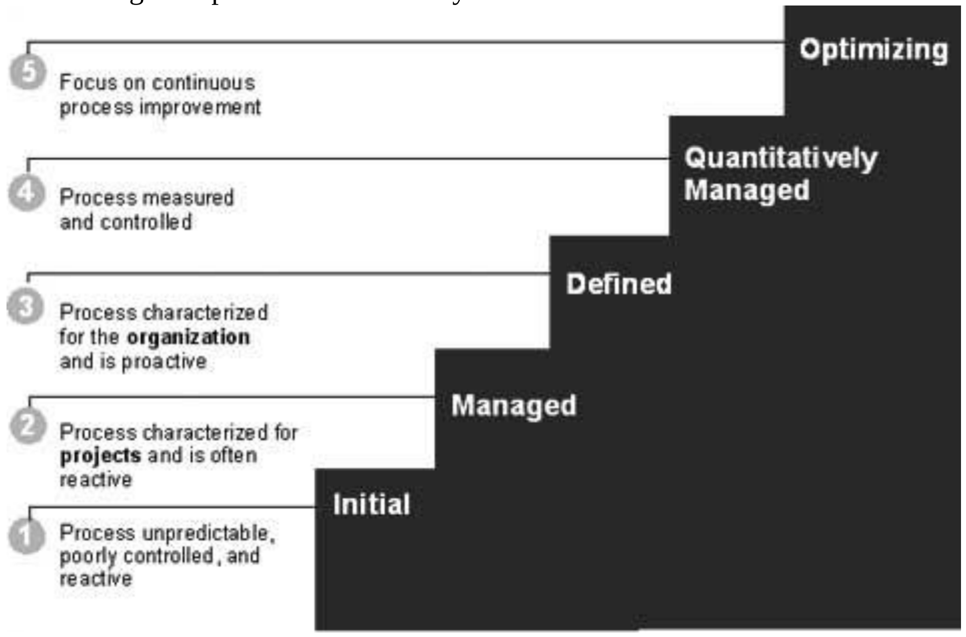
A maturity level is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement.

In CMMI models with a staged representation, there are five maturity levels designated by the numbers 1 through 5

1. Initial
2. Managed

3. Defined
4. Quantitatively Managed
5. Optimizing

CMMI Staged Represenation- Maturity Levels


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

Now we will give more detail about each maturity level. Next section will list down all the process areas related to these maturity levels.

Maturity Level Details:

Maturity levels consist of a predefined set of process areas. The maturity levels are measured by the achievement of the **specific** and **generic goals** that apply to each predefined set of process areas. The following sections describe the characteristics of each maturity level in detail.

Maturity Level 1 - Initial

At maturity level 1, processes are usually ad hoc and chaotic. The organization usually does not provide a stable environment. Success in these organizations depends on the competence and heroics of the people in the organization and not on the use of proven processes.

Maturity level 1 organizations often produce products and services that work; however, they frequently exceed the budget and schedule of their projects.

Maturity level 1 organizations are characterized by a tendency to over commit, abandon processes in the time of crisis, and not be able to repeat their past successes.

Maturity Level 2 - Managed

At maturity level 2, an organization has achieved all the **specific** and **generic goals** of the maturity level 2 process areas. In other words, the projects of the organization have ensured that requirements are managed and that processes are planned, performed, measured, and controlled.

The process discipline reflected by maturity level 2 helps to ensure that existing practices are retained during times of stress. When these practices are in place, projects are performed and managed according to their documented plans.
At maturity level 2, requirements, processes, work products, and services are managed. The status of the work products and the delivery of services are visible to management at defined points.
Commitments are established among relevant stakeholders and are revised as needed. Work products are reviewed with stakeholders and are controlled.
The work products and services satisfy their specified requirements, standards, and objectives.
Maturity Level 3 - Defined
At maturity level 3, an organization has achieved all the **specific** and **generic goals** of the process areas assigned to maturity levels 2 and 3.
At maturity level 3, processes are well characterized and understood, and are described in standards, procedures, tools, and methods.
A critical distinction between maturity level 2 and maturity level 3 is the scope of standards, process descriptions, and procedures. At maturity level 2, the standards, process descriptions, and procedures may be quite different in each specific instance of the process (for example, on a particular project). At maturity level 3, the standards, process descriptions, and procedures for a project are tailored from the organization's set of standard processes to suit a particular project or organizational unit. The organization's set of standard processes includes the processes addressed at maturity level 2 and maturity level 3. As a result, the processes that are performed across the organization are consistent except for the differences allowed by the tailoring guidelines.
Another critical distinction is that at maturity level 3, processes are typically described in more detail and more rigorously than at maturity level 2. At maturity level 3, processes are managed more proactively using an understanding of the interrelationships of the process activities and detailed measures of the process, its work products, and its services.
Maturity Level 4 - Quantitatively Managed
At maturity level 4, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, and 4 and the **generic goals** assigned to maturity levels 2 and 3.
At maturity level 4 Subprocesses are selected that significantly contribute to overall process performance. These selected subprocesses are controlled using statistical and other quantitative techniques.
Quantitative objectives for quality and process performance are established and used as criteria in managing processes. Quantitative objectives are based on the needs of the customer, end users, organization, and process implementers. Quality and process performance is understood in statistical terms and is managed throughout the life of the processes.
For these processes, detailed measures of process performance are collected and statistically analyzed. Special causes of process variation are identified and, where appropriate, the sources of special causes are corrected to prevent future occurrences.

Quality and process performance measures are incorporated into the organizations measurement repository to support fact-based decision making in the future.

A critical distinction between maturity level 3 and maturity level 4 is the predictability of process performance. At maturity level 4, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable. At maturity level 3, processes are only qualitatively predictable.

Maturity Level 5 - Optimizing

At maturity level 5, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, 4, and 5 and the **generic goals** assigned to maturity levels 2 and 3.

Processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes.

Maturity level 5 focuses on continually improving process performance through both incremental and innovative technological improvements.

Quantitative process-improvement objectives for the organization are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement.

The effects of deployed process improvements are measured and evaluated against the quantitative process-improvement objectives. Both the defined processes and the organization's set of standard processes are targets of measurable improvement activities.

Optimizing processes that are agile and innovative depends on the participation of an empowered workforce aligned with the business values and objectives of the organization. The organization's ability to rapidly respond to changes and opportunities is enhanced by finding ways to accelerate and share learning. Improvement of the processes is inherently part of everybody's role, resulting in a cycle of continual improvement.

A critical distinction between maturity level 4 and maturity level 5 is the type of process variation addressed. At maturity level 4, processes are concerned with addressing special causes of process variation and providing statistical predictability of the results. Though processes may produce predictable results, the results may be insufficient to achieve the established objectives. At maturity level 5, processes are concerned with addressing common causes of process variation and changing the process (that is, shifting the mean of the process performance) to improve process performance (while maintaining statistical predictability) to achieve the established quantitative process-improvement objectives.

Maturity Levels Should Not be Skipped:

Each maturity level provides a necessary foundation for effective implementation of processes at the next level.

Higher level processes have less chance of success without the discipline provided by lower levels.

The effect of innovation can be obscured in a noisy process.

Higher maturity level processes may be performed by organizations at lower maturity levels, with the risk of not being consistently applied in a crisis.

Maturity Levels and Process Areas:

Here is a list of all the corresponding process areas defined for a S/W organization. These process areas may be different for different organization.

This section is just giving names of the related process areas, for more detail about these Process Areas go through CMMI Process Areas Chapter.

Level	**Focus**	**Key Process Area**	**Result**
5 .Optimizing	Continuous Process Improvement	Organizational Innovation and Deployment Causal Analysis and Resolution	Highest Quality / Lowest Risk
4.Quantitatively Managed	Quantitatively Managed	Organizational Process Performance Quantitative Project Management	Higher Quality / Lower Risk
3.Defined	Process Standardization	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Mgmt (with IPPD extras) Risk Management Decision Analysis and Resolution Integrated Teaming (IPPD only) Org. Environment for Integration (IPPD only) Integrated Supplier Management (SS only)	Medium Quality / Medium Risk
2.Managed	Basic Project Management	Requirements Management Project Planning Project Monitoring and Control Supplier Agreement Management Measurement and Analysis Process and Product Quality Assurance	Low Quality / High Risk

		Configuration Management	
1.Initial	Process is informal and Adhoc		Lowest Quality / Highest Risk

Six Sigma

Six Sigma has following two key methodologies:

1. DMAIC: refers to a data-driven quality strategy for improving processes. This methodology is used to improve an existing business process.
2. DMADV**:** refers to a data-driven quality strategy for designing products & processes. This methodology is used to create new product designs or process designs in such a way that it results in a more predictable, mature and defect free performance.

There is one more methodology called DFSS - Design For Six Sigma. DFSS is a data-driven quality strategy for designing design or re-design a product or service from the ground up.

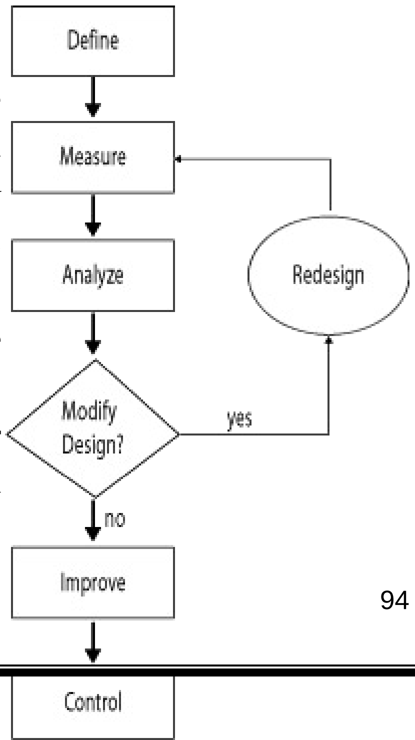
Sometimes a DMAIC project may turn into a DFSS project because the process in question requires complete redesign to bring about the desired degree of improvement.

DMAIC Methodology:

This methodology consists of following five steps.

Define --> Measure --> Analyze --> Improve -->Control

1. Define : Define the Problem or Project Goals that needs to be addressed.
2. Measure: Measure the problem and process from which it was produced.
3. Analyze: Analyze data & process to determine root causes of defects and opportunities.
4. Improve: Improve the process by finding solutions to fix, diminish, and prevent future problems.
5. Control: Implement, Control, and Sustain the improvements solutions to keep the process on the new course.



6. DMAIC is a data-driven quality strategy used to improve processes. It is an integral part of a Six Sigma initiative, but in general can be implemented as a standalone quality improvement procedure or as part of other process improvement initiatives such as lean.
7. DMAIC is an acronym for the five phases that make up the process:
8. Define the problem, improvement activity, opportunity for improvement, the project goals, and customer (internal and external) requirements.

9. Measure process performance.
10. Analyze the process to determine root causes of variation, poor performance (defects).
11. Improve process performance by addressing and eliminating the root causes.
12. Control the improved process and future process performance.

DMADV Methodology:

This methodology consists of following five steps.

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

1. Define : Define the Problem or Project Goals that needs to be addressed.
2. Measure: Measure and determine customers needs and specifications.
3. Analyze: Analyze the process for meet the customer needs.
4. Design: Design a process that will meet customers needs.
5. Verify: Verify the design performance and ability to meet customer needs.

DFSS Methodology:

DFSS - Design For Six Sigma is a separate and emerging discipline related to Six Sigma quality processes. This is a systematic methodology utilizing tools, training and measurements to enable us to design products and processes that meet customer expectations and can be produced at Six Sigma Quality levels.

This methodology can have following five steps.

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

1. Define : Identify the Customer and project.
2. Identify: Define what the customers want, or what they do not want.
3. Design: Design a process that will meet customers needs.
4. Optimize: Determine process capability & optimize design.
5. Verify: Test, verify, & validate design.