T.Y. B.Sc. (IT)

SEMESTER - VI

PROJECT MANAGEMENT

Prof. Suhas Pednekar
Vice-Chancellor,
University of Mumbai

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai

Prof. Prakash Mahanwar
Director,
IDOL, University of Mumbai

Programme Co-ordinator : **Shri Mandar Bhanushe**
Head, Faculty of Science and Technology
IDOL, University of Mumbai, Mumbai

Course Co-ordinator : **Ms. Gouri Sawant**
Asst. Professor, B.Sc. IT,
IDOL, University of Mumbai, Mumbai

Course Writers : **Mr Sumedh Shejole**
Asst. Professor,
IDOL, University of Mumbai, Mumbai

: **Ms Sujata Rizal Kotian**
Asst. Professor,
S.M.Shetty College Powai, Mumbai

: **Ms Aarti sahitya**
Asst. Professor,
Kjsieit, Mumbai

December 2021, Print - I

Published by : Director
Institute of Distance and Open Learning ,
University of Mumbai,Vidyanagari, Mumbai -400 098.

DTP Composed : Mumbai University Press
Printed by Vidyanagari, Santacruz (E), Mumbai - 400 098

CONTENTS

T.Y.B.Sc.IT SEMESTER – VI
Project Management

SYLLABUS

Unit-I	Conventional Software Management : The waterfall model, conventional software Management performance. Evolution of Software Economics : Software Economics, pragmatic software cost estimation. Improving Software Economics : Reducing Software product size, improving software processes, Achieving required quality, peer inspections.
Unit-II	The old way and the new : The principles of conventional software Engineering, principles of modern software management, transitioning to an iterative process. Life cycle phases : Engineering and production stages, inception, Elaboration, construction, transition phases. Artifacts of the process : The artifact sets, Management artifacts, Engineering artifacts, programmatic artifacts. Model based software architectures : A Management perspective and technical perspective.
Unit-III	Work Flows of the process : Software process workflows, Iteration workflows, Checkpoints of the process : Major mile stones, Minor Milestones, Periodic status assessments. Iterative Process Planning : Work breakdown structures, planning guidelines, cost and schedule estimating, Iteration planning process, Pragmatic planning.
Unit-IV	Project Organizations and Responsibilities: Line-of-Business Organizations, Project
Unit-V	Project Control and Process instrumentation: The seven core Metrics, Management indicators, quality indicators, life cycle expectations, pragmatic Software Metrics, Metrics
Unit-VI	Future Software Project Management: Modern Project Profiles, Next generation Software economics, modern process transitions.

1

CONVENTIONAL SOFTWARE MANAGEMENT

Unit Structure

1.1 INTRODUCTION

Software management is a process through which a software is managed throughout its development process.

The software management also called as software project management (SPM) deals with managing the critical aspects right from start of the planning of the project to deployment and maintenance.

SPM manages the following aspects of software development; Version tracking, developing, testing, integration, configuring, installing, and distributing

In short, SPM is about managing:

software engineering work encouraging the stakeholder interaction in the early SDLC stages so as to avoid paying attention to the technical tools and methods used in the process planning the product objectives and scope, cost

5PM specially focuses on four P's:

1. People
2. Product
3. Process
4. Project

1.2 CONVENTIONAL SOFTWARE MANAGEMENT

As per the analysis of the software engineering industries of mid 1990s, it yielded that the Software Development was highly unpredictable because:

- Only about 10% of software projects were delivered successfully in time, in budget, and meeting the user requirements.

- The management discipline acted more as a discriminator in success or failure or when in case of technology advances.

- The level of software scrap and rework was indicated to be at an immature level process. The reasons behind this analyzation may either be bad theory or bad practice orboth.

1.3 THE WATERFALL MODEL

We have already learnt about waterfall modelin's of ware engineering' subject. A brief overview of this model is shown in Fig.1.3.1;

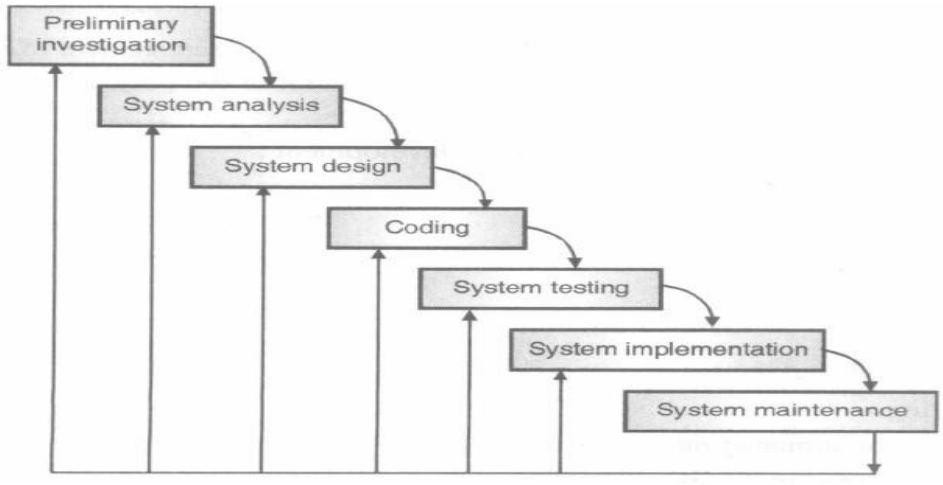


Fig. 1.3.1: Waterfall model

The detail description about the working of this model is not necessary, here. Therefore, we will only see the various perspectives of the 'waterfall model' in earlier days (historical perspective) and how it is updated nowadays. This linear waterfall model was first proposed by 'Winston Royce'. It suggests systematic sequential approach for software development. It is the oldest paradigm for software engineering i.e. it is the oldest software development life cycle.

1.3.1 Preliminary Investigation:

Preliminary investigation means total inspection of the existing system i.e. clear understanding of the system. Its basic task is to find out real problem of the system with causes and complexity of the problem. Its secondary but very important task is to find out all possible solutions to solve that problem and according to that which solution is feasible for the system in terms of technology, cost, operational. Its last task is to mention all benefits can be expected after problem is solved.

So, this phase is divided into three main goals as follows:

- Problem identification
- Possible and feasible problem solution i.e., Feasibility study. Expected benefits after the problems are solved.

1.3.1.1 Problem Identification (Problem Analysis):

It requires to completely investigate the environment of the system. Generally it requires studying two environments - Internal environment and external environment, which are listed below:

Sr. No.	Internal Environment	External Environment
1	Company Management	Customers
2	Employees of all departments	Management consultant
3	Internal auditors	External auditors
4	Data Processing department	Government Policies
5	Financial Reports	Competitions

There are normally seven types of problems encountered in the system:

- **Problem of Reliability:** If system may not work properly or same procedures give different (i.e., unreliable)result.
- **Problem of Validity:** Reports contain misleading information.
- **Problem of Economy:** System is costly to maintain. **Problem of Accuracy:** Reports have many errors. **Problem of Timeliness:** Every work requires large time.
- **Problem of Capacity:** Capacity of the system is inadequate.
- **Problem of Throughput:** System does not produce expected results, or we can say system has more capacity but it accomplishes very less work as compared to capacity. The main advantageof waterfall model is, it exactly pins points the problem. So, it is very useful in setting all goals of the system as well as used to decide system boundaries.

1.3.1.2 Project Feasibility Study:

Feasibility study is essential to evaluate cost and benefit of the proposed system. This is very important step because on the basis of this; system decision is taken on whether to proceed or to postpone the project or to cancel the project.

Need of Feasibility Study:

- It determines the potential of existing system.
- It finds or defines all problems of existing system. It determines all goals of the system.
- It finds all possible solutions of the problems of existing system. We can call it as proposed system.
- It finds technology required to solve these problems.
- It determines really which solution is easy for operational from the point of view of customer and employees such that it requires very less time with 100% accuracy.
- It determines what hardware and software is required to obtain solution of each problem or proposed system.

- It determines cost requirements of the complete proposed system in terms of cost of hardware required, software required, designing new system, implementation and training, proposed maintenance cost.
- It avoids costly repairs, crash implementation of new system.
- It chooses such system which is easy for customer to understand and use so that no special training is required to train the customer. It may give some training to employees of the system.

Method:

Steering committee:

This committee conducts detailed study. This committee first studies existing system and identifies all problems and looks into three types of feasibility study. Those are given below:

Technical feasibility Operational feasibility Economical feasibility Organizational feasibility Cultural feasibility

Technical Feasibility:

- The committee first finds out technical feasibility of the existing system. It involves following steps:
- It determines available hardware.
- It determines available computer with configuration. It determines available software.
- It determines operating time of available system that is computer, hardware software.

After that it finds out technical feasibility required for the proposed system. It involves following steps:

- It mentions new hardware requirements of proposed system.
- It mentions computer with new configuration requirement of proposed system. It mentions new software requirements of proposed system.
- It mentions new operating time of available system that is computer, hardware, software. It mentions old as well as new facilities which will be provided by the proposed system.
- It mentions all benefits of the system.

Operational Feasibility:

It is also called as behavioral feasibility. It finds out whether the new technology or proposed system will be suitable using three type of aspects; that are human, organizational and political aspects. It involves following steps:

1. It finds out whether there is any direct-indirect resistance from the user of this system or not?
2. It finds whether the operation of proposed system is easy or not as compare to existing system.
3. It finds out whether the user or customer of the system requires extra training or not?
4. If it requires any retraining then it is accepted by user as well as customer or not?
5. It finds if any job reconstruction is required or not?
6. It finds if this reconstruction of the job is accepted in organization?
7. It also finds if it is acceptable then what should be the skill sets of thatjob.
8. Watches the feelings of the customers as well asusers.
9. It should provide right and accurate information to user or customers at right place as well as at righttime.

Economical Feasibility:

Here, steering committee finds total cost and all benefits as well as expected savings of the proposed system.

There are two types of costs - **onetime cost and recurring costs**. **One time cost** involves following:

1. Feasibility study cost.
2. Cost converting existing system to proposed system.
3. Cost to remolding architecture of the office, machineries, roomsetc.
4. Cost of hardware's.
5. Cost of Operating system software.
6. Cost of Application software.
7. Technical experts consulting costs.
8. Cost of training.
9. Cost of Documentation preparation

Recurring cost involves following:

1. Cost involves in purchase or rental of equipment.
2. Cost of phones and mobiles Communication equipment.
3. Cost of Personnel search, hiring, staffing.
4. Cost of Salaries of employee's.
5. Cost of supplier's.
6. Cost of maintenance of equipment.

Organizational Feasibility:

This demonstrates the management capability and availability, employee involvement, and their commitment. This shows the management and organizational structure of the project, ensuring that the system structure is as described in the requirement analysis or SRS and is well suited to the type of operation undertaken.

Some organizational risks impacting new system:

- Low level of computer competency among employees Perceived shifting of organizational power
- Fear of employment loss due to increased automation Reversal of long-standing work procedures
- One way to overcome these risks: training sessions.

Cultural Feasibility:

- It deals with the compatibility of the proposed new project with the cultural environment of the project.
- In labor-intensive projects, planned functions must be integrated with the local cultural practices and beliefs.

Example: Religious beliefs may influence what an individual is willing to do or not do.

Example: An enterprise's own culture can clash with the results of the project.

In these conditions, the project's alternatives reevaluated for their impact on the local and general culture.

1.3.2 System Analysis:

This phase is about complete understanding of all important facts of the business based on the preliminary investigation. It involves following activities:

1. It is the study of all components of the system as well as inter relation between all components of the system and relation between components and environment.
2. It determines what is to be done in the organization.
3. It finds the procedures of how to do that.
4. What is the input data?
5. What is the procedure through which inputs are to be converted into output?
6. When should the transactions occur on the data?

7. When problems arise, determine the solutions to solve it and what are the reasons behind those problems.

Methodology of System Analysis:

1. Identifying the system boundary
2. Understanding the role and interrelationship of elements with other elements of the same system.

The above two methodologies generate:

The capability to analyze and compare various alternatives regarding components and system functioning and system objectives.

Need /Advantages of System Analysis:

- Provides greater understanding of the complex structures
- It acts as a tradeoff between functional requirements of subsystems to the total system
- It helps in understanding and comparing the functional impacts of subsystems to the total system
- It helps in achieving the inter-compatibility and unity of purpose of sub systems helps in discovering means to design systems
- Helps in placing each subsystem in its proper place and context, so that the system as a whole may achieve its objectives with minimum available resources.

Objectives of System Analysis:

1. Define the system.
2. Divide the system into smaller parts.
3. Finds all nature, function and inter-relationship of various parts of the systems.

If the system is not analyzed properly then there may be problem in the Preliminary investigation phase.

1.3.3 Software Design:

The main objective of this phase is to design proposed system using all information collected during preliminary investigation and directed by the system analyst. This is a challenging phase and includes following steps:

1. Design of all types of inputs of proposed system.
2. Design of all types of outputs of proposed system.
3. Design of the procedures which convert input to output.
4. Design of the flow of information.

5. Design of the information which is required to store within a files and data bases Volumes.
6. Design of collection of inputs using forms (Manual forms).
7. Design in terms of program specification i.e., logical design.
8. It determines the hardware cost, hardware capability.
9. It determines the speed of software.
10. It determines error rates, and other performance characteristics are also specified.
11. It also considers the changes to be made in the organizational structure of the firm in design.
12. This phase also designs standards for testing, documentation.

Generally traditional tools are used for the designing of the procedures that are as follows:

Flowcharts Algorithms

IPO (i.e., Input Processing and output) and HIPO (Hierarchy of IPO) charts Decision tables

Data Flow Diagrams

1. If system design phase is facing problem during the design, then first go back to the system analysis phase and redesign the system but if problem is not solved then go for preliminary investigation.

2. If System design phase produces all expected results, then it goes to next phase.

1.3.4 Coding:

This phase is just implementing the design in to programming language that means it actually develops the proposed system. It involves the following steps:

1. It first of all, finds out the best suitable programming language that is suitable for the design as well as also suitable in the organization.

2. It accepts design and break system modules into smaller programs.

3. It develops or writes program for each part in selected programming language.

4. Prepares documentation that means add necessary comment lines wherever necessary within a program.

5. Now it combines all small programs together and builds one big program.

6. If any problem occurs during the coding phase, then waterfall model tries to solve it by repeating system design phase:

- If that problem does not get solved then waterfall model repeats system analysis phase and system design phase.
- If that problem does not get solved then waterfall model repeats from first phase i.e., preliminary phase through system analysis phase and system design phase.

7. If coding phase produces all expected results, then it goes to next phase.

1.3.5 Testing:

This phase includes the testing of the code or programs developed by the coding phase. This includes following steps:

1. First of all, it finds out all possible expected results (i.e. output data) for the set of input data.

2. It also checks the validity of the input data as well as checks expected output data.

3. It finds out all wrong results and immediately tries to correct it by repeating coding phase.

4. It finds the speed of functions using special codes.

5. It determines whether each program can perform the intended tasks or not?

6. It checks result by test data.

7. It checks the logic of the individual programs.

8. It checks interfaces between various programs.

9. It checks quality of code in terms of speed and space.

10. It checks whether system has produced correct and desired results which lead to designated goals.

11. If testing finds out that the system does not produce expected result then this problem is solved by repeating the previous phases as needed.

12. If testing phase finds out that the system produces all expected results then it goes to next phase.

1.3.6 System Implementation:

System Implementation is not creative process but it is somewhat difficult task. This phase has two parts- implementation and evaluation of the system.

Implementation:

There are two ways of implementation. Those are as follows:

1. Implement proposed system with existing old system and find out performance of the both systems and slowly replace new system with older one.

2. Totally replace old system with proposed new system.

Risk factor of second type of implementation is more as compare to first one. Second step needs strict evaluation.

Both types of implementations consist of following steps:

1. It prepares site for new proposed system.
2. It installs required hardware within a system.
3. It installs required software in a system.
4. It installs a developed code i.e., programs in a system.
5. It prepares training program for user of the proposed system as well as customers of the system.
6. It prepares user manual which includes all the steps which give guidance to user.
7. It gives training to all types of users of proposed system.
8. Observe the system when users of system are using it.
9. If users are facing any problems regarding the new system, it tries to find out exact phase from where root cause of the problem starts, and accordingly starts waterfall model.

Evaluation:

Evaluation is nothing but feedback for the system. It is very essential check point of the system which is the process of verifying the capability of a system. It continuously evaluates and checks whether proposed system is meeting the objectives or not. It includes:

1. Development Evaluation:

It checks whether the system is developed within time.

It checks whether the system is developed within the budget. System is passed by development methods and tools.

2. Operational Evaluation:

- It checks response time of proposed system.
- It checks whether it is really easy to use or not?
- It checks accuracy of computations (It is seen in testing also). It checks storage capacity.
- It checks reliability.
- It checks functioning of the existing system. Collects necessary feedback from users.

- It finds all benefits of the proposed system.
- Collects information of attitude of different persons regarding proposed system. It evaluates cost, time and effort taken for the overall project.

1.3.7 Maintenance:

Maintenance is the process in which it finds out essential changes (i.e. new trends) of the market or business to correct some errors and tries to implement it in the existing system.

There are usually three types of maintenance, that are:

1. Correction:

- Sometimes, proposed system has few types of errors; and it is the duty of software engineer to correct it as soon as it is encountered by the user. Generally, there are four types of errors, that are as follows:
- Minor changes in the processing logic. Errors detected during the processing. Revisions of the formats for data inputs. Revisions of the formats of the reports.
- These errors can be corrected by repeating waterfall model from coding phase through testing, implementation and maintenance.

2. Adaptation:

Sometimes, our proposed system is executable on Windows environment, but somebody wants to run it in LINUX environment, or some other operating system. Then we are required to design our proposed system from third phase that is from System Design phase.

3. Enhancement:

Because of new technology and business competition, organization needs to imply or to add new functions or additional capabilities to the proposed system.

After some time, people think that some techniques may be used in the system so some additional features can also be added into it. Sometimes, new hardware is required to add some extra features. For enhancement it may repeat whole system or may repeat it from design phase or sometimes from coding phase.

Advantages of Waterfall Model:

1. It defines very first software development process.
2. The product of waterfall model always defines all constraints of the organization.
3. It always produces a good quality product in terms of space and time.

Drawbacks of the Waterfall Model are as follows:

1. Real products rarely follow this sequential flow.
2. Because of iteration, changes can cause confusion as the project team proceeds
3. It is very difficult for customer to state all the requirements in onetime.
4. Many projects face this uncertainty at beginning only, so it is very difficult to design next phases.
5. Time span required for each phase could not be specified.
6. Naturally project requires more time.
7. Project becomes lengthy also.
8. Customer should have patience.

It tries to solve time consumption in early stages.

1.4 HISTORICAL PERSPECTIVE

1. In earlier days, developers gave more importance to the two most essential steps i.e., 'analyses' and 'coding'. These two steps were interpreted as the two most important steps at that time.

2. So as to manage and control the activities of the software development process, the developers introduced various 'overhead' steps such as the system requirements definition, software requirements definition, program design, and testing. All these steps supported in enhancing the analysis and coding steps."

3. The conventional basic waterfall model is risky and failure prone. This is because as the testing phase occurs at the end of the development life cycle, and as a result, the changes identified are likely to be so troublesome to implement right from the software requirements on which the design is based.

1.4.1 Suggested Changes

1. Earlier, **"Program design comes first".**

- Mostly, Designing was done between SRS and analysis phases. Program designer first checks the storage, timing, and data.
- During the Analysis phase, the Designer imposes timing and operational constraints so as to cross-check the consequences.
- The software development design process is always built by the program designers, and not by the analysts. The program designers define and allot various data processing modes (such as allocating functions, designing databases, interfacing, processing modes (such as the i/o processors), operating procedures (such as the one entering into a branch even when it is wrong.

- Training is given to the staff until they learn and use the software
- Now;"Architecture comes first" is given importance and comes first rather than program design.
- That means, nowadays, the basic ARCHITECTURE comes FIRST.
- It includes the elaboration of architecture by distributing the system into components, and representing these components in a layered architecture.
- For example, the Rational Unified process (RUP) includes use-case driven, architecture-centric, iterative development process.
- These architectures are THEN designed and developed in parallel with planning and requirements definition.

2. Earlier;**"Program design comes first"** then, document these Designs

- In earlier days of software development, the Development required huge amounts of documentation i.e. everything had to be represented using manuals.
- These manuals are called as User manuals. These included operational manuals, software maintenance manuals, staff user manuals, test manuals and etc.
- Maintaining such a huge amount of documentation becomes really troublesome.
- It was MUST for each designer to communicate with various stakeholders such as interface designers, managers, customers, testers, developers so as to finalize the designs and prepare the manuals.

Now, **"Document the Design first",** then derive the actual designs

- Nowadays, 'artifacts' are given primary importance. These models (documentation / artifacts) are derived from the developed architecture, analysis report, captured requirements, and these models (documentation / artifacts) are then used in deriving a design solution.
- These documentation /artifacts include Use Cases, static models such as class diagrams, state diagrams, activity diagrams, dynamic models such as sequence and collaboration diagrams, domain models, glossaries, supplementary specifications (such as constraints, operational environmental constraints, distribution constraints, etc.)
- Modem designing tools and notations, and new methods produce self-documenting artifacts from development activities. For example, Rational Rose is a very popular design tool that produces the self-documenting artifacts and program code from the developed model.

- Therefore, we can say that, Visual modeling provides significant documentation.

3. Earlier; "Do it twice" THEN, document these Designs

- It is sometimes very confusing when we do the same program twice i.e., the first version of the program may contain some errors or say, some loop holes, so we make changes in it and derive it's second version. But, later on, keeping track of the most recent version of these programs become troublesome.

- Version 1 has major problems and therefore, alternatives are addressed – for example, the changes are done in 'big cookies' such as communications, user interfaces, data models, hardware/software platforms, operational or any other constraints. Therefore, there is a need to track the correct version of the program and also, if needed the first version is thrown away sometimes.

- Then version 2 is called as the refinement of version 1 which includes the implementation of major requirements.

Now; "Architecture -first Development" THEN, derive the actual designs

This approach is the base to architecture-first development. This is known as 'Initial engineering' which forms the basis for Iterative development and also helps in addressing the risks.

4. Earlier; "Plan, Control, and Monitor Testing"

- Testing phase utilizes the greatest number of project resources which includes manpower, computation time, ultimately cost and schedule.

- Therefore, it involves greatest risks also in terms of cost and schedule.

- The testing phase occurs in the last of software development and this last phase is reached when any other changes or any alternatives are least available and expenses are at a maximum.

- This phase involves:

1. Employing a team of test specialists and these specialists not involved in the original design and development of the software product.

2. Employing visual inspections to detect the obvious errors. These inspections include the procedure of code reviews, technical reviews and interfaces.

3. Testing every logical path

4. Final check out on the customer's computer.

Now; "Plan, Control, and Monitor Testing"

- The 1st and 4th testing activities are still validi.e.;

1. Employing a team of test specialists and these specialists not involved in the original design and development of the software product.

4. Final checkout on the customer's computer.

- But the 2nd and 3rd activities differ from the earlier perspectives;

2. Software inspections - are conducted by automated tools but they are assisted by the code analyzers, optimizing compilers, static and dynamic analyzers.

3. Testing every path - This is practically impossible. It is especially very difficult with distributed systems and reusable components and there are even many other factors that need to be tested.

5. Earlier; "Involvement of the Customers".

- Itinvolvescustomers'participationinrequirements'definition,inprelimin arysoftwarereview, and also in preliminary program design.
- Now, "Involvement of the Customers"
- It involves the customers also and all other stakeholders also. This involvement is necessary for the overall project success.
- It demonstrates the increments, the customer feedback, making favorable changes, cyclic and iterative activities that yield in an evolving software.
- Involvement of customers helps in addressing the risks in very initial phases of software development.

1.5 THE SOFTWARE DEVELOPMENT PLAN

- If you observe the Old Versions of Software Development Plan, you will see that the success rate is less than 20%.
- The old versions of software development plan define:
- Precise requirements
- Precise plan to deliver and deploy the system constrained by specific time and budget. Executes and tracks to plan

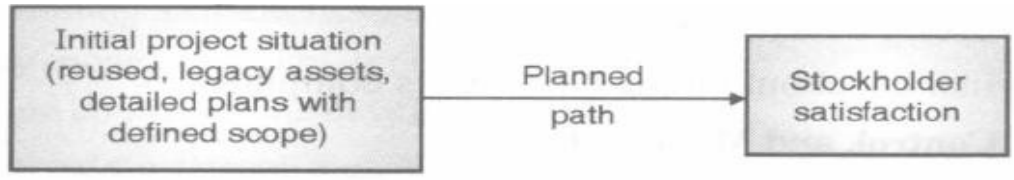


Fig. 1.5.1 : Old Version of Software Development

- In practice, these activities of software development plan didn't help much in increasing the success rate.
- Projects are not delivered on-time, not within planned budget, and rarely met the user requirements.
- Projects that are developed using Conventional waterfall process often faced following symptoms:

1. Protracted integration and late design breakage
2. Late risk resolution
3. Requirements-driven functional decomposition
4. Adversarial stakeholder relationships
5. Focus on documents and review meetings

1.5.1 Protracted (Prolonged/Late) Integration and Late Design Breakage

- The activities that are carried out in conventional waterfall process are listed down in a sequential manner: Requirements - Design - Code - Integration–

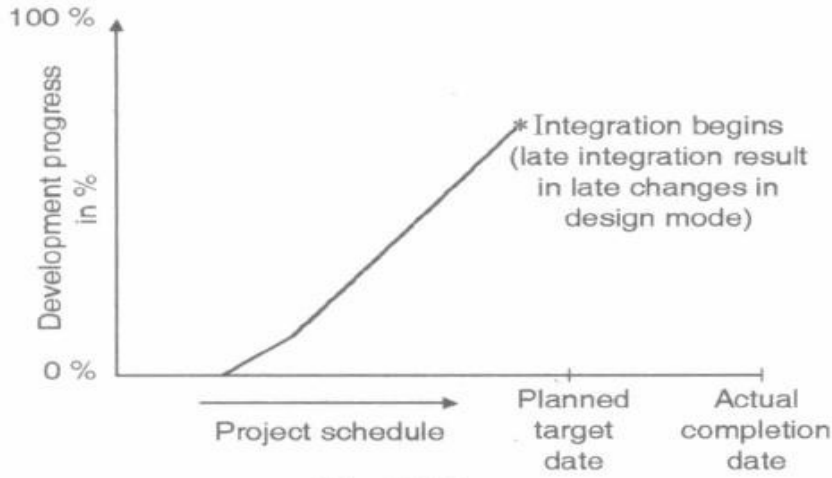


Fig. 1.5.2

The integration of components very late in the SDLC, then it results in late changes in the designed models. This is known as late design i.e. coming back to design phase when you are at the last phases of the life cycle. This late designing causes the breakages in the developed product.

- The late integration test and designing also increases the expenditures of the software project.

Table 1.5.1: Expenditures per activity for a Conventional Software Project

Activity	Cost
Management	– 5%
Requirements	– 5%
Design	– 10%
Code and unit test	– 30%
Integration and Test	– 40%
Deployment	– 5%
Environment	– 5%
Total	**– 100%**

- • Lot of time is spent on finalizing the software design and then after spending so much of time on designs and after perfecting them to their satisfaction, then the task of coding is started.
- Generally, the requirements are noted down in English, designs are drawn using flowcharts, detailed designing is doneinpal, and implementations are basically done in Fortran, Cobol, or C.
- The problems of‘ Water fall model-late integration’ that effect the software performance are
- Only unit testing is performed from the start of coding but all other tests could be performed only ‘at the end’ of the SDLC phases,
- Thus, the late integration and testing phase consumes 40% of life-cycle resources: which will not if we start the integration of components and testing at the right time.

1.5.2 Late Risk Resolution

- Focuses on early prepared paper artifacts.
- Actual risks and issues are still unknown and difficult to detect and understand.
- The below figure shows a sample risk profile for projects that follow waterfall model. It involves four different periods of risk exposure;

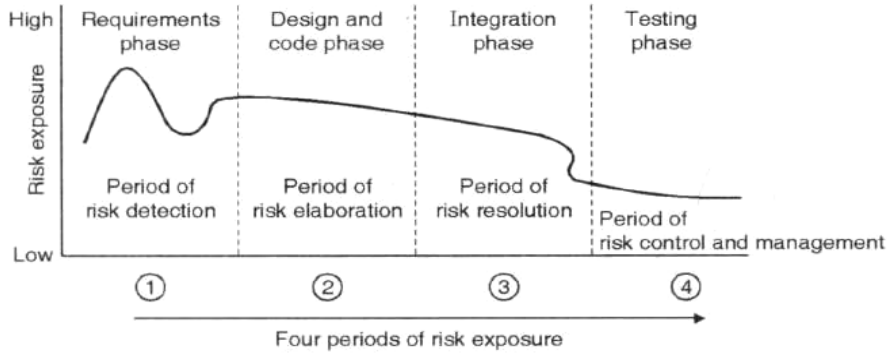


Fig 1.5.3 : Four Different Periods Of Risk Exposure

- Difficult to identify and resolve the risks right during the requirement gathering because this is the initial phase of SDLC and in this phase, many key items are still not fully understood.
- Even in the design phase, where we are clear enough with the software requirements and have better understood them, then also it is still difficult to analyze and finalize the objectives.
- During the coding phase, some of the risks are resolved, but more than that, especially at the time of integration, many of the risks become quite clear and accordingly changes are made to the artifacts and reduction in expenditures can be achieved.
- Even if we achieve success in reducing the expenditures, it often results in extending the scheduled dates of delivering the product.
- This may also hamper the quality a little bit due to frequent changes and extensibility, or in the process of maintainability. Thus, it ultimately results in the loss of original design and integrity.

1.5.3 Requirements-driven Functional Decomposition

- From the traditional perspective, the software development processes have been interpreted as requirements-driven as shown in the below Fig.1.5.4.
- Developers must focus on gathering and writing complete, clear, precise, consistent, necessary, and feasible user requirements. But this case was rarely observed.
- Very often, it is seen that too much time is spent on treating all requirements (such as the requirements that are normally listed in. condition-action tables, decision-logic tables, flowcharts, or plain text) equally rather than spending more time on critical ones.

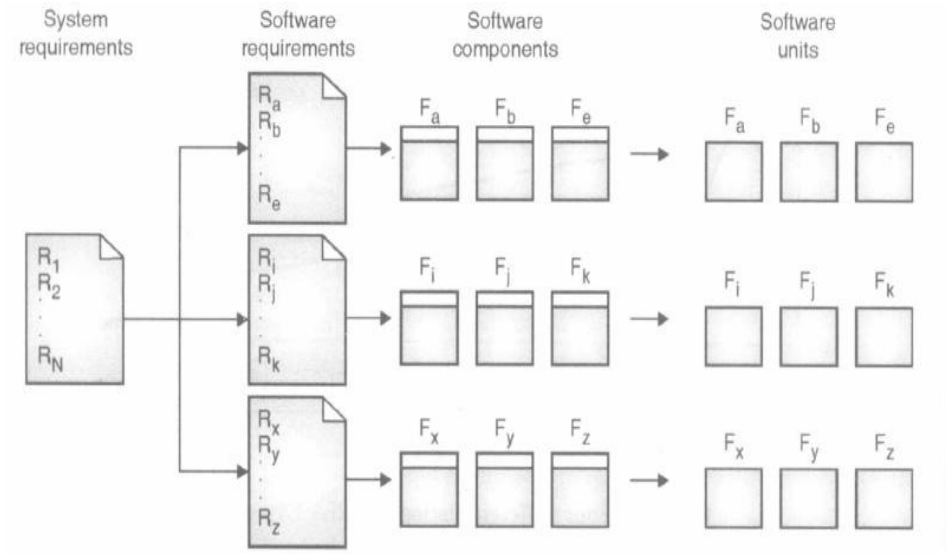


Fig. 1.5.4 : Component Organization interpreted from A Requirements-Driven Approach

- Muchofthebrainpowerandtimeiswastedonthe‘lessimportant’requirements .Thatmeans, in traditional way of development, prioritizing of requirements was given least importance.
- Also, much time is spent on documentation i.e., documenting the features (traceability, testability, etc.) of the software which were later made obsolete as the requirements and designs subsequently evolve.
- Also, there is a false assumption in conventional software development that all the requirements can be implemented as ‘functions’ in the coding phase and this as sumption leads to the decomposition of these functions.
- This conversion of requirements into functions and then leading to decomposition of Functions into sub-functions, etc. has become the basis for contracts and work distribution, while ignoring the major architectural-driven approaches that are threaded throughout the functions and that surpass every individual function such as the security; authentication; persistency; and performance.

1.5.4 Adversarial Stakeholder Relationships

- First of all, there is a need to decide ‘who are stakeholders?
- Lot of misunderstandings occur between the stakeholders about the matter written in the documentation because of the English jargons.
- Requirements are only documented on papers but in actual practice, they are not really modeled.
- Universally-agreed notations are rarely used, no common notations, no GUIs. Subjective reviews and different stakeholder opinions are not given enough value.

Few common events that take place in contractual software are:

1. Contractor prepared a draft that constitutes of contract-deliverable document that involves intermediate artifacts and delivered it to the customer for approval. This is usually done after interviews, questionnaires, and meetings.
2. Customer was ought to give feedback within 15-30days.
3. Contractor incorporated this feedback and submitted within 15-30 days. This is interpreted as the final version for approval.

Observation:

- Huge paperworksuchthatitbecame'intolerable'andoftensomeofthedocumentation was under-read or un-read.
- Tense contractor/customer relationships
- Mutual distrust which was the basis for much of the problems.
- It was often seen that, once approved is rendered obsolete later.

1.5.5 Focuson Documents and Review Meetings

It is a documentation-intensive approach i.e., focusing much on documentation.

But in this procedure, little attention is given on producing credible increments of the desired products.

- It follows the Big bang approach i.e. all FDs are delivered at once. That means, all Design Specifications are made 'OK' at once.
- Milestones are decided and passed over to all stakeholders via review meetings. These meetings may be technical or managerial.
- Lot of energy is spent on producing paper documentation to show progress versus efforts and to address the real risk issues and also the integration issues. These issues can be such as;
  - Stakeholders did not go through design.
  - Verylowvaluetothestakeholders'opinionsinthemeetingsbuthavetopayhighcostsontravel and accommodations.
- Many issues could have been avoided during early life-cycle phases rather that caused serious problems in later life cycle phases.

Typical Software product design Reviews:

1. Big briefing to a diverse audience

- Only very small percentage of audience understands the software programming and development.

- Briefings and documents represent few of the important assets and risks of that particular complex software.

2. A compliant design

But there is no tangible proof of compliance with unclear requirements is of little value.

3. Requirement Coverage

Only few are the real design drivers, but many are presented as to be real Dealing with all requirements diverts the focus on critical drivers.

4. A design is considered 'innocent until proven guilty'

The design errors are not detected in the early phases are exposed in the later life cycle.

1.6 CONVENTIONAL SOFTWARE MANAGEMENT PERFORMANCE

Most of the conventional software development generally describe the fundamental economic relationships that are derived from years of practice.

Basic Software Economics:

1. Detecting and fixing the software bugs after the delivery to customer costs 100 times more than finding and fixing the problem in early design phases.

2. You can compress i.e., minimize the software development schedules up to 25% but no more. Addition of staff requires more management overhead and training of the new staff.

Some compression is sometimes proved to be troublesome to add new people.

3. For every penny, you spend on software development, you will spend the double on maintenance.

Successful products have much higher ratios of "maintenance to development".

A good development organization will most likely NOT spend this kind of extra money on maintenance.

4. Software development and maintenance costs are primarily the results of number of source lines of code.

- Component-based development weakens this by increasing the reuse but not in common use in the past.

5. Variations among stakeholders and especially variations in staff results in the biggest differences of software productivity.

Development organization must always try to hire good people.

Build the 'team concept.' With no "I" in 'team", and there must be an implicit "we."

6. Overall ratio of software to hardware costs is growing.

- Impacting these figures is the ever-increasing demand for functionality and attending the complexity.

7. Only about 15% of software development efforts are devoted to actual coding.

- And, this 15% is only for the programming. And near about, some 65% - 70% of the overall total life cycle expenses are based on maintenance.

8. Software systems and products generally cost 3 times as much per SLOC (Source Lines of Code) as individual software programs.

Software-system products i.e., system of systems costs nine times as much. The more software you build, the more expensive it is per source line.

9. Walkthroughs detect 60% of the errors.

Walkthroughs are good for catching errors, but they require deep analysis to catch significant shortcomings.

Major problems such as performance and resource contention are also not caught.

10. 80% of the contribution is from 20% of the contributors i.e., 80/20 rule applies to many things.

Review Questions

Q.1 What is software project management?

Q.2 What was the historical perspective of conventional software development?

Q.3 Suggest the changes in earlier and 'new' version of software development.

Q. 4 What are the problems faced due to late risk resolution?

Q. 5 Write a note on typical software development reviews?

Q. 6 Write in brief about conventional software management performance.

Q. 7 What is Late design breakage?

Q. 8 What do you mean by software economics?

2

EVOLUTION OF SOFTWARE ECONOMICS

Unit Structure

2.1 Software Cost Estimation

2.2 Software Economics

2.3 Pragmatic Software Cost Estimation

2.4 Cocomomodel

2.5 Reducing Software Productsize

2.6 Improving Software Process

2.7 Improving Automation

2.8 Peerinspections

Syllabus:

Evolution of Software Economics: Software Economics, Pragmatic Software Cost Estimation.

Improving Software Economics: Reducing Software Product size, improving soft ware process, improving team effectiveness, improving automation, achieving required quality, peer inspections

2.1 SOFTWARE COST ESTIMATION

- Planning and estimating are iterative processes which continue throughout the course of a project.
- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

Costs of Software Engineering:

The total cost required in developing a software product can be categorized as below;

2.1.1 60% of development costs

2.1.2 40% are testing costs.

But these costs differ depending on the type of the project being developed and the required system attributes such as the performance, reliability, flexibility, security andso on.

The required cost in various phases also depends on the development model (either prototyping or spiral or waterfall or any other model) that is used.

Cost Estimation:

- Is the art of approximating the probable cost of something based on information available at the time?
- Leads to a better understanding of the problem.
- Improves management insight into resource allocation problems. Provides an objective baseline to measure progress.
- The reliability of cost estimates varies over time. The closer you get to the actual completion of a project; the estimate becomes more accurate.

Cost estimating problems occur most often because of the:

Inability to accurately size a software project,

Inability to accurately specify a software development and support environment, Improper assessment of staffing levels and skills, and

Lack of well-defined requirements for the specific software activity being estimated

Four types of cost estimates represent various levels of reliability:

Conceptual Estimate: Often inaccurate because there are too many unknowns.

Preliminary Estimate: Used to develop initial budget, more precise.

Detailed Estimate: Serves as a basis for daily project control.

Definitive Estimate: Accuracy should be within 10% of final cost.

Cost Estimations are constructed based on the following tasks:

- • Identifying the purpose and scope of the new system-new software development, software reuse, COTS integration,etc.
- Choosing an estimate type - Conceptual, preliminary, detailed, or definitive type estimates. Identifying system performance and/or technical goals.

- Laying out a program schedule. Collecting, evaluating, and verifying data.
- Choosing, applying, cross-checking estimating methods to develop the cost estimate. Performing risk and sensitivity analysis
- Providing full documentation.

2.1.1 SoftwareCostEstimationProcess

Software Cost Estimation process comprises of 4 main steps:

Step 1: Estimate the size of the development product

Size of the software may depend upon: lines of code, inputs, outputs, functions, transactions, features of the module andetc.

Step 2: Estimate the effort in person-hours

- The effort of various Project tasks expressed in person-hours is influenced by various factors such as:
- Experience/Capability of the Team members technical resources
- Familiarity with the Development Tools and Technology Platform

Step 3: Estimate the schedule in calendar months.

The Project Planners work closely with the Technical Leads, Project Manager and other stakeholders and create a Project schedule. Tight Schedules may impact the Cost needed to develop the Application.

Step 4: Estimate the project cost in dollars (or other currency).

Based on the above information the project effort is expressed in dollars or any other currency.

2.1.2 Cost Estimation Techniques

1. Expert Opinion:

Also called as Delphi method, proposed by Dr. Barry Boehm is useful in assessing differences between past projects and new ones for which no historical precedent exists.

Advantages:

Little or no historical data needed. Suitable for new or unique projects.

Disadvantages:

- Very subjective.
- Experts may do partiality
- Qualification of experts may be questioned.

2. Analogy:

- Estimates costs by comparing proposed programs with similar, previously completed programs for which historical data isavailable.
- Actual costs of similar existing system are adjusted for complexity, technical, or physical differences to derive new costestimates
- Analogies are used early in a program cycle when there is insufficient actual cost data to use as a detailedapproach
- Compares similarities and differences
- Good choice for a new system that is derived from an existing subsystem.

Advantages:

- Inexpensive Easily changed
- Based on actual experience (of the analogous system)

Disadvantages:

- VerySubjective
- Large amount of uncertainty
- Truly similar projects must exist and can be hard to find
- Must have detailed technical knowledge of program and analogous system

3. Parametric:

Utilizes statistical techniques. Can be used prior to development.

Advantages:

- Can be excellent predictors when implemented correctly Once created, CERs are fast and simple to use
- Easily changed
- Useful early on in a program Objective

Disadvantages:

- Often lack of data on software intensive systems for statistically significant CER Does not provide access to subtle changes
- Top level; lower level may be not visible
- Need to be properly validated and relevant to system

4. Engineering:

- Also referred to as bottoms up or detailed method.
- Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate.
- Future costs for a system are predicted with a great deal of accuracy from historical costs of that system.
- Involves examining separate work segments in detail. Estimate is built up from the lowest level of system costs. Includes all components and functions.
- Can be used during development and production.

Advantages:

- Objective
- Reduced uncertainty

Disadvantages:

- Expensive
- Time Consuming Not useful early on
- May leave out software integration efforts

5. Actual:

- Decides future costs on recent historical costs of same system.
- Used later in development or production.
- Costs are calibrated to actual development or production productivity for your organization

Advantages:

- Most accurate
- Most objective of the five methodologies

Disadvantages:

- Data not available early Time consuming
- Lab our intensive to collect all the data necessary

Choice of methodology depends upon:

- Type of system - software, hardware, etc.
- Phase of program - Development, Production, Support

- Available data - Historical data points from earlier system versions or similar system or technical parameters of system.

2.1.3 Cost EstimationParameters

Various models (such as COCOMO, Costar etc.) are available for estimating the cost of software development.

All these cost estimating models can be represented on the basis of five basic parameters:

1. Size: The size of the proposed software product is weighed in terms of components i.e., ultimately in terms of the number of source code instructions or the number of functions required in developing the proposed product.

2. Process: The process includes the phases and activities carried out in each phase. So whatever process used to produce the proposed product is measured on the ability of the Target process to avoid unnecessary activities such as rework, bureaucratic delays, communications overhead and such other overhead activities which may delay the delivery of theproduct.

3. Personnel: This deals with the capabilities and experience of software engineering **personnel** (team members) in the field of computer science issues and the applications domain issues of theproject. Italsodepends uponthe personnel's' familiarity with the Development Tools and Technology Platform.

4. Environment: The **environment** constitutes of the tools and techniques that are required to develop efficient software and also to automate theprocess.

5. Quality: The required **quality** of the proposed product depends upon the features, performance, reliability, and adaptability of thesoftware.

The above described five parameters (size, process, personnel, environment and quality) can be related with each other so as to calculate the estimated cost for the proposed software development:

Process

Effort/Cost = (Personnel) (Environment) (Quality) (Size)

2.2 SOFTWARE ECONOMICS

The most important aspects of **software economics** according to that represented in today's software cost models is the "relationship between **effort** and **size".** This relationship represents a **diseconomy** of scale.

This diseconomy of the software development is because the **process** exponent is greater than 1.0 i.e., the more software (that means, the more function units) you build, the more expensive it is per unit code.

2.2.1 Three Generations of Software Economics:

- The below figure demonstrates how the technology, tools, component development and processes evolved and enhanced through the three generations of software economy. 1. Con
- The estimated level of quality and personnel (person capabilities per hour) are, considered to be constant.
- The ordinate of the graph represents the software unit costs interpreted by an organization.

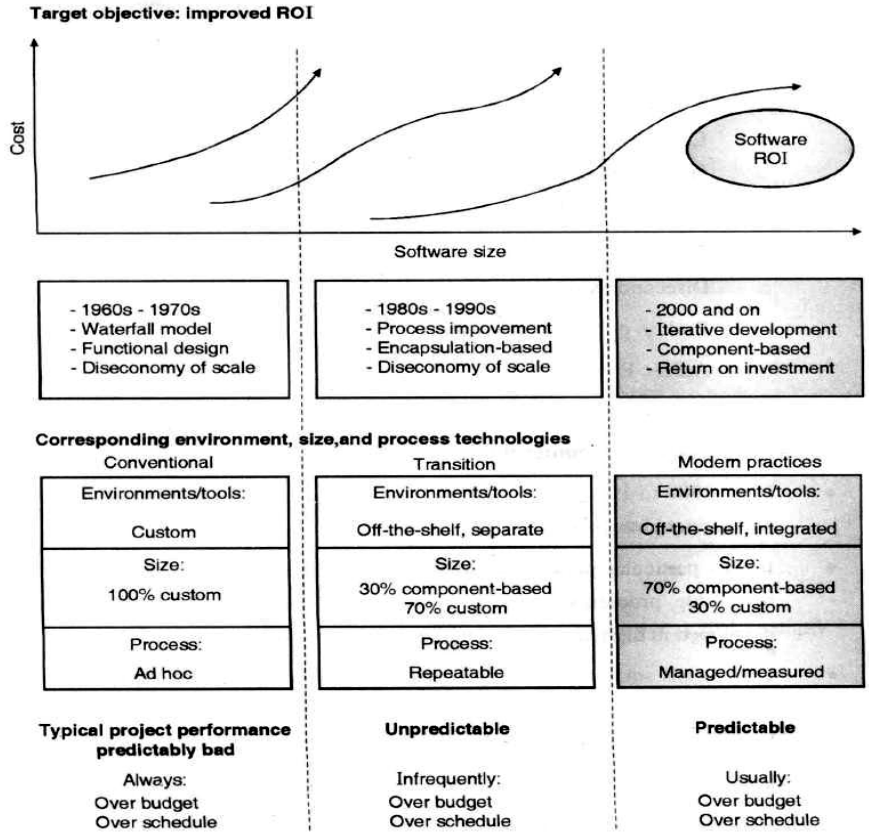


Fig. 2.2.1: Three Generations of Software Economics

1. Conventional (Craftsmanship):

- From 1960 to 1970, the conventional software development process was referred toas craftsmanship such as the Waterfall model.
- In this particular period, software development companies made use of tools, processes and components built in primitive languages.

That means, project performance was highly as expected and that too in the estimated cost and schedule.

But the quality was not as expected.

Main features:

Custom processes and tools Functional design

Custom environment (100%)

Drawbacks:

Diseconomy of scale

Built in primitive languages Ad Hoc Process

Always over budget and over schedule

Transition (Software Engineering):

- From 1980 to 1990, the conventional software development process was called as software engineering.
- In this particular period, the software development companies used frequently repeatable processes and off-the-shelf tools, and lot of custom components developed in high level programming languages such as Java and.Net.
- That means some of the components also included commercial products such as Networking and graphical user interfaces, operating systems like Windows and Linux, database systems like Oracle and SQLServer.
- **Main features:**
  - maturity and creativity
  - research-intensive
  - Process improvement
  - Encapsulation-based
  - Built in higher-level languages
  - Repeatable process
  - Off-the-shelf environment (70% custom and 30%component-based)
- **Drawbacks:**
  - Diseconomy ofscale
  - Infrequently on budget and onschedule

3. Modern practices (software production):

- Since 2000, the software development process is referred to as software production.

- Now a days, software development companies make use of managed and measured processes, integrated environments such as automated tools and technologies like the Rational Rose tool that we use for drawing UML diagrams, more of off-shelf components and few custom components.

- **Main features:**
  - production-intensive
  - automation
  - economies ofscale
  - Iterativedevelopment
  - Component-based
  - Off-the-shelf environment (70% component-based and 30%custom)
  - Process -managed/measured
  - Usually on budget and onschedule

Throughout each of these three software economic generations, the main focus was on the corresponding growth of technologies. For example, consider the process advances that cannot be used successfully without the involvement of new component technologies and the enhanced tool automation.The development organizations are constantly achieving better economies of scale as shown in every successive generation with very large projects, long-lasting software products, and business cases

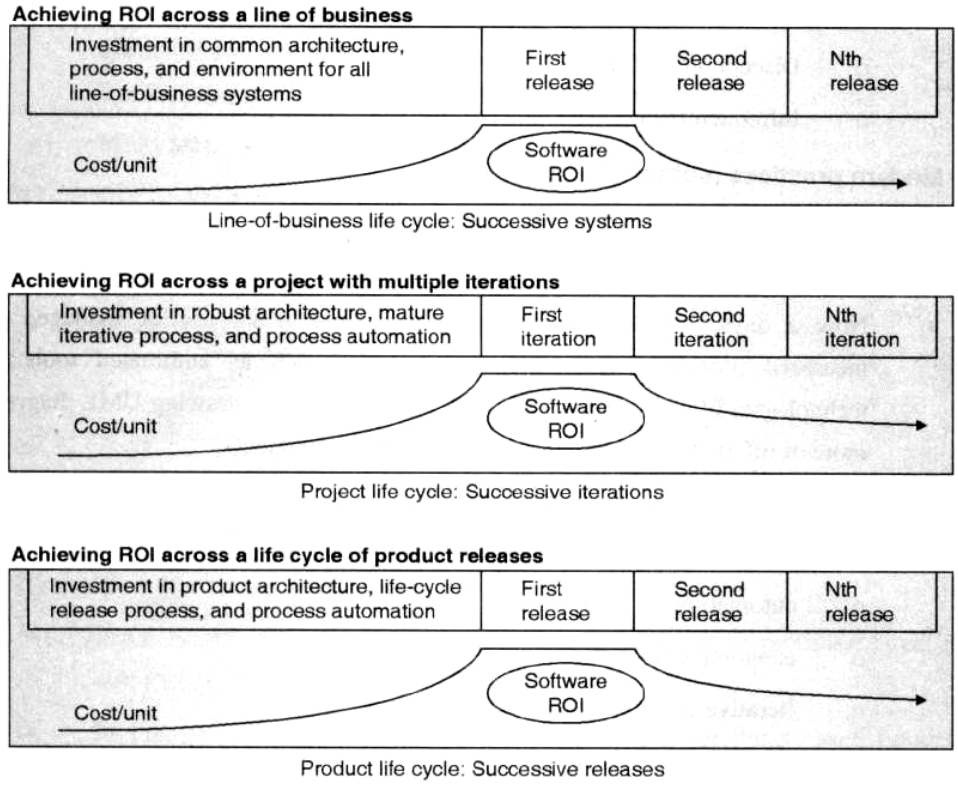


Fig. 2.2.2 : ROI in various Business Domains

comprising of multiple similar projects.The figure below gives an overview of how the return on investment (ROI) can be achieved through continuous efforts across the life cycles of various business domains.

2.3 PRAGMATIC SOFTWARE COST ESTIMATION

- Major issue of software cost estimation is the inability of developing well-documented case studies in the projects that use iterative development approach (also called as Rational Unified Process (RUP)).
- Software development companies don't have aclearidea abouts of tware metrics and measures which means that the data storage and retrieval process is not clear in terms of data consistency and the actual quality attributes of the projects don't match with planned attributes.
- It is easy to collect related set of data within one particular organization but it is difficult to collect the related data from different companies following different development processes, different programming languages and different domains.
- Developers and customers always had an argument on software cost estimation models and tools.

Three most common topics of their argument are:

1. Selection of Cost Estimation Model

- There are various cost estimation models (COCOMO, COSTXPERT, CHECKPOINT, SLIM, ESTIMACS, Knowledge Plan, SEER, SOFTCOST, Costar, REVIC, Price-S, Pro QMS etc.) that are based on statistically derived cost estimating relationships (CERs) and various estimating methodologies.

2. Whether to measures oft ware sizeon the basis of lines of code (LOC) or function points?

- Most of the cost estimation models are bottom-up i.e., substantiating a target cost rather than top-down i.e., estimating the 'should' cost.
- The below figure depicts predominant cost estimation practices where:
  - First, project manager defines the target cost of thesoftware.
  - Later, he manipulates the parameters and does the sizing until the target cost is justified.
- The target cost is defined so as:
  - to win the proposal,
  - to solicit the customer funding,
  - to attain the internal corporate funding, or
  - to achieve some other goal.

- The process described in this figure forces the software project manager to check the risks associated in achieving the target costs and also discuss this problem with other stakeholders.

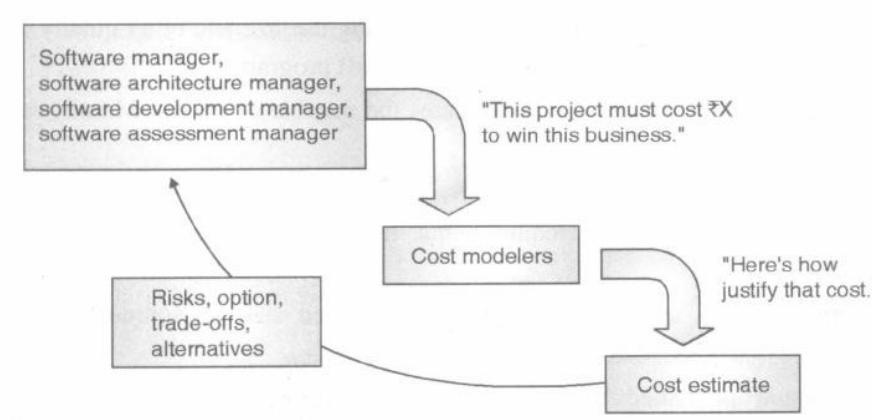


Fig. 2.3.1 : Predominant Cost Estimation Process

3. Factors that lead to good cost estimation

Good software cost estimation may be based on the following attributes:

- The cost is estimated collectively by the project manager, architectural team, development team, and test team.
- The estimated cost is acknowledged and supported by all stakeholders. The cost estimation is based on some well-defined model.
- The cost estimation is based on the databases of similar type of project experiences that use similar methodologies, similar technologies, and similar environment with similar type of stakeholders.
- • The key risk areas are detected and accordingly the probability of success isdetermined.

2.4 COCOMOMODEL

- COnstructiveCOstMOdel (COCOMO) is one of the earliest cost models widely used by the cost estimating community.
- COCOMO was originally published in Software Engineering Economics by Dr. Barry Boehm in 1981.
- COCOMO is a regression-based model that considers various historical programs software size andmultipliers.
- COCOMO's most fundamental calculation is the use of the Effort Equation to estimate the number of Person-Months required in developing aproject.
- COCOMO stands for COnstructiveCOstMOdel. It is the oldest cost estimation model that is popularly used in the process of costestimation.

- COCOMO was first published by Dr. Barry Boehm in1981.
- COCOMO model estimates the cost by considering the size and other quality aspects of the similar type of historical (previously developed) programs.
- COCOMO calculates Efforts i.e., it estimates the number of Person-Months required in developing a project.

Number of person months * loaded lab our rate = Estimated Cost

- Most of the other estimates (requirements, maintenance, etc.) are derived from this quantity. COCOMO requires as input the project's estimated size in Source Lines of Code (SLOC).
- Initial version published in 1981 was COCOMO-81 and then, through various instantiations came COCOMO 2.
- COCOMO 81 was developed with the assumption that a waterfall process would be used and that all software would be developed from scratch.
- Since then, there have been many changes in software engineering practice and COCOMO 2 is designed to accommodate different approaches to software development.

The COCOMO model is based on the relationships between the two formulas:

- **Formulae 1:** Development effort is based on system size. MM = a.KDSIb where,

MM is the effort measured in Man per Moths

KDSI is the number of Source Instructions Delivered in a Kilo (thousands).

- **Formulae 2:** Effort (MM) and Development Time. TDEV = c.MMd where,

TDEV is the development time.

In both the above formulas, we have used the coefficients a, b, c and d which are dependent upon the 'modeofdevelopment'.According to **Boehm, the mode of development canbe classified into following 3 distinct modes:**

1. **Organic** mode of development - talks about the projects that involve small development teams whose team members are familiar with the project and work to achieve stable environments. This category includes the projects like the **payroll systems.**
2. **Semi-detached** mode of development **-** talks about the projects that involve mixture of experienced team members in the project. This category includes the projects like the **interactive banking system.**

3. **Embedded** mode of development - talks about the complex projects that are developed under tight constraints with innovations in it and have a high volatility of requirements. This category includes the projects like the **nuclear reactor control systems.**

Drawbacks:

- It is difficult to accurately estimate the KDSI in early phases of the project when most effort estimates are still not decided yet.
- Easily thrown misclassification of the development mode.
- Its success largely depends on tuning the model to the needs of the organization and this is done based upon the historical data which is not always available.

Advantages:

- COCOMO is transparent that means we can se it working.
- Allows the estimator to analyse the different factors that affect the project costs.

2.4.1 C0C0M0 2 Model

- COCOMO 2 constitutes of sub-models so as to produce in detail software estimates. The sub models are listed as follows;
- Application composition model. It is applied when software is being developed from existing parts.
- Early design model. It is applied when system requirements are gathered and concluded but design has not yet started,
- Reuse model. It is applied when software is being developed using the reusable components so as to compute the effort of integrating these components,
- Post-architecture model. It is applied when once the system architecture has been designed and when more system information is gathered. Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.

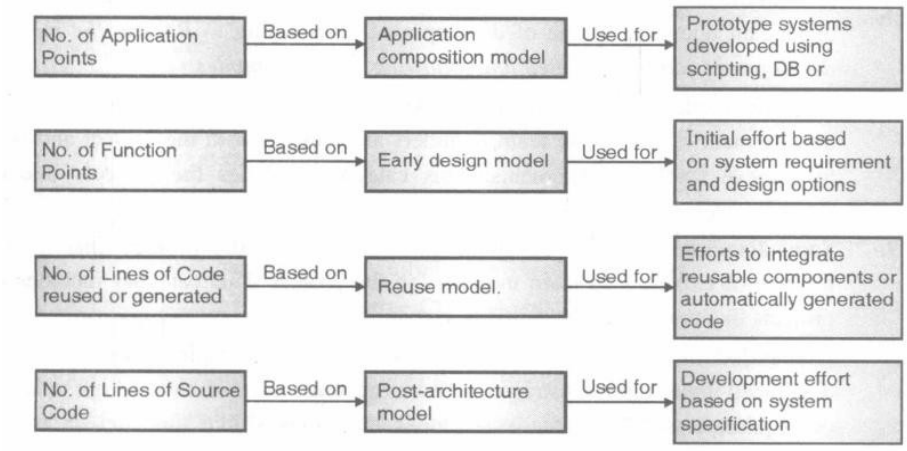


Fig. 2.4.1 : Use of COCOMO2 Model

Product attributes describe the required characteristics of the software product being developed.

Computer attributes describe the constraints imposed on the software product by the hardware platform. Personnel attributes describe the experiences and capabilities (of the project development team members) that are taken into account. Project attributes describe particular characteristics of the project. Estimating the calendar time required to complete a project and when staff will be required using a COCOMO 2 formula TDEV = 3 ' (p m)(0-33+0-2*<b-101»

- PM is the effort computation and B is the exponent (B is 1 for the early prototyping model). This computation predicts the nominal schedule for the project.

- The time required is independent of the number of people working on the project. Improving Software Economics,the software economics canbe improved by using a'balanced' approach as the key. The Five Key parameters that can help in improving the software economics are;

Reducing the product size (Number of Lines of Code) and complexity of the software

1. Improving the software development process
2. Using more-skilled personnel i.e., improving the team effectiveness.
3. Creating better environment by improving the automation with more appropriate tools and technologies.
4. Achieving required quality by peer inspections

Table 2.4.1 : Trends on which the Five Key Parameters depend

Five Key parameters that effect the Cost Model	Trends
Size: describes abstraction and component based development technologies	High level programming Languages (C++, Java, VB, Object Oriented Methods and Visual modeling (analysis, design and programming) Reusability Commercial Exponents Packages
Process: involves methods and techniques	Iterative Development Process Maturity Models such as CMM Architecture-first development
Personnel: describes the effectiveness of the development team	Training to develop the personnel skills Team work Win-win culture
Environment: involves automated tools and technologies.	Integrated tools such as compiler, editors, and debuggers. Hardware performance Automated coding, documentation, testing and analyses.
Quality: describes the Performance, reliability, accuracy issues	Hardware platform performance Peer inspections Statistical quality control

2.5 REDUCING SOFTWARE PRODUCTSIZE

• The more the number of lines of code, the larger becomes the size of the product; larger the size of the product, more expensive becomes the product and these expenses are measured 'per line'.

• The most proper way of improving the affordability and ROI (return on investment) is to develop a product that achieves the design goals with minimum number of LOC (Lines of Code) i.e., minimum amount of source code.

Parameters that affect the size complications are:

- Use of Component-based development i.e., breaking up the software product into simple modules and these modules are coded separately and after complete coding is over, all these modules are then integrated together. This decomposition of the product first and then later integration raises complications in the code.
- Automatic code generation - Various design tools generate automatic code along with the modeling of the designs. Such tools sometimes generate lot of unwanted code.
- Graphical User Interface (GUI) builders include the code that is needed to build an easy to access user interface. In such process, more lines of code are included in the program.
- 4th Generation Languages (4GLs) make use of classes, structures, dynamic memory allocations and many such new concepts that increase the complexity of the code.

- Object Oriented (OO) Modeling Languages used for analysis, design and modeling also increases the complexity of the software.

2.5.1 ReducingSoftwareProductSize-Languages

UFP (Universal Function Points):

- These points are language independent.
- The basic units of the UFP are external user inputs, external outputs, internal logical data groups, external data interfaces, and external inquiries. SLOC (Source Lines of Code) m metrics:
- These metrics are useful estimators when a solution is formulated and programming language is known.

Table 2.5.1: Various Comparisons of Function points to lines of code based upon the programming language used.

Language	SLOC per UFP
Assembly	320
C	128

Language	SLOC per UFP
Fortran 77	105
Cobol 85	91
Ada 83	71
C++	56
Ada 95	55
Java	55
Visual Basic	35

Advantages:

- Use of higher-level programming languagesre duces the size; thus, the 'level of abstraction' also changes allowing more focus on architecture.

- There duced size make siteasier to understand, reuse, maintain and import the packages of classes and objects.
- But these higher-level abstractions often use high storages and communication bandwidths.

2.5.2 Reducing Software Product size-OO Methods and Visual Modelling

• OO (Object Oriented) technology reduces the size of the program. How the use of OO methodology helps in reducing the size can be summarized asbelow;

Benefits of using OO methodology:

- An OO methodology improves the team work and interpersonal communications by improving the understanding between the end users and developers of the system. This ultimately increases the productivity andquality.
- OOSE (OO software Engineering) is achieved using the OO modeling languages like UML and the configurable processes like RUP (Rational Unified Process) which involves iterative development.
- OO methodology supports continuous integration of subsystems, classes, interfaces thus, increasing the chances of early detection of risks and incremental corrections without hampering the stability of the development process.
- OO methodology allows 'Architecture first approach' in which integration is an early and continuous life-cycle activity that brings stability in development, allows development and configuration of components in parallel.
- OO architecture creates a clear separation between the unrelated elements of a system, and includes firewalls that prevent a change in one part of the system that may be caused due to the errors in other part of the system thus, rending the structure of the entire architecture.

2.5.3 ReducingSoftwareProductSize-Reusability

- The 'reusability' is most lyimplemented with stored functions and sub programs. There are many forms of reuse:
- Reuse of Old stuff such as data descriptions, document, and a collection of similar-old artifacts, designs, and architectures.

  Reuse of Functions, classes from any phases of software development

  Reuse of Common architectures such as common processes, and common environments. Such reuse is very common during huge project development.

- Differences in OS platforms and HAV environments such as the middleware, and also the GUI Builders have hampered the reuse potentiality.

Examples: Common Microsoft platforms, Linux MACs resulting from distributed applications.

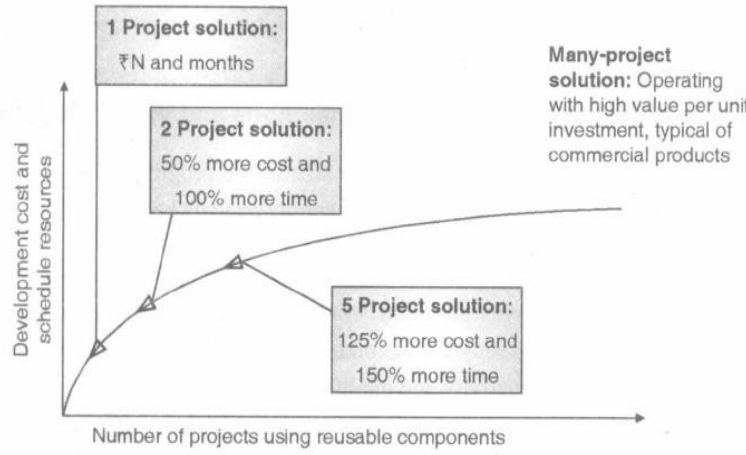


Fig. 2.5.1 : Graph showing the Development cost in the projects using Reusable Components

The main reason for Reuse is 'lack' of money.

- Costs needed to build reusable and configure reusable components. Reuse is implemented across many projects if they are similar.
- Few commercial organizations sell commercial components encouraging the Reusability.

2.5.4 Reducing Software Product Size-Commercial Components

Main advantage of using Commercial components is that it saves custom development efforts Commercial components usually need tailoring i.e. they need finishing before they are used.

Commercial components have an impact on quality, cost, supportability, and the architecture.

Table 2.5.2 : Pros and cons of Commercial Components v/s Custom Software

Approach	Pros (Advantages)	Cons (Disadvantages)
Commercial Components	Predictable costs largely used and is a matured technology Available now a days supports organizations Hardware and Software independence Rich in functionalities	Needs frequent upgrades and maintenance Charges Up-front license fees Charges Recurring maintenance fees Highly Depends on vendor Run-time efficiency is little bit less Lot of functionality constraints Integration is always needed at each step Inclusion of Unnecessary features that consume extra resources mostly, inadequate reliability and stability Multiple vendor incompatibility
Custom Development	Freedom of making changes Provides smaller and simpler implementations Provides better performance Control on development and enhancement	Unpredictable development Cost Unpredictable availability date Undefined maintenance model Often immature and easily breakable Single-platform dependency Consumes expert resources

2.5.5 ReducingSoftwareProduct size-Packages

- Various elements (use cases; classes, other model elements) of analysis model are categorized into packages that are given a representative name.

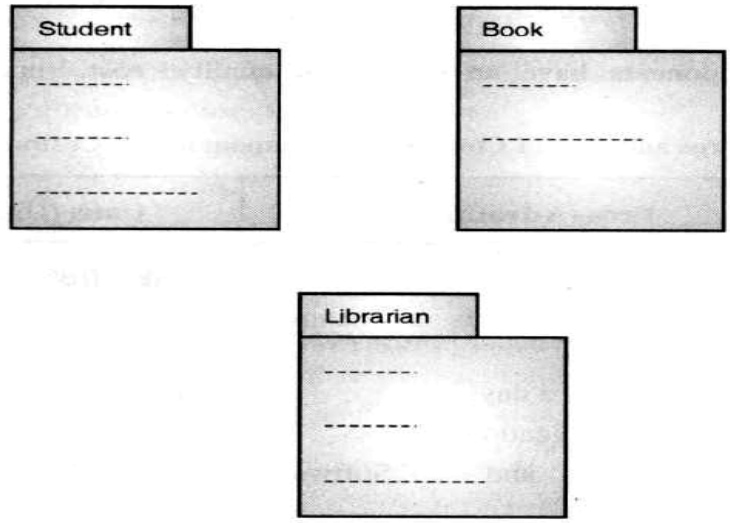
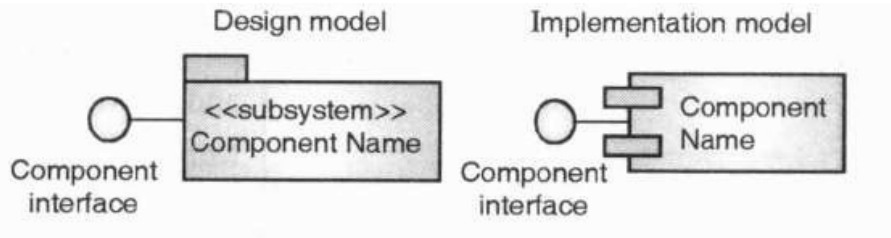


Fig. 2.5.2 : Packages

Example: Student class, Book class, Librarian class.

- Package is a model element that can contain other model elements - use case models, classes, objects, subsystems, components, other model elements and packages.
- The Components help to model the physical aspect of an Object-Oriented software system representing the relationship between various components.



- A component diagram contains components and dependencies. The dependencies between the components show how changes made to one component may affect the other components in the system. Dependencies in a component diagram are represented by a dashed line between two or more components.

The component diagram contains:

- Components are denoted by rectangle with two smaller rectangles protruding from its left side. They represent the subsystems in the design model.
- Dependencies are denoted by dashed lines between two or more components. They represent how changes made to one component may affect other components of the system.

Example: Component Diagram for Library Management System

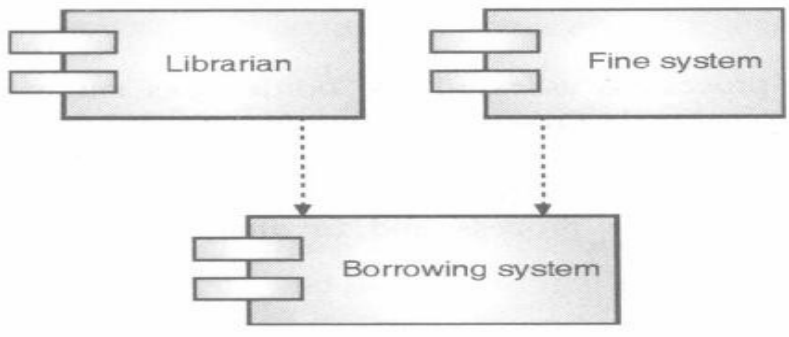


Fig. 2.5.3 : Component Diagram for LMS

The above diagram shows 3 main components of the library management system in which the first component (Librarian) manage seach student's profile and also who manages library items (issuing and returning).

Second component (Fine system) manages fines applied to the student who exceed the borrowing period. Third component (Borrowing system) manages all borrowing items.

2.6 IMPROVING SOFTWARE PROCESS

- It involves understanding the existing processes and introducing changes in it so as to improve the product quality, reduce costs and accelerate schedules.
- Process improvement work focuses mostly on defect reduction and improving the development process.
- Process improve mentisacyclic activity as shown in the below figure:

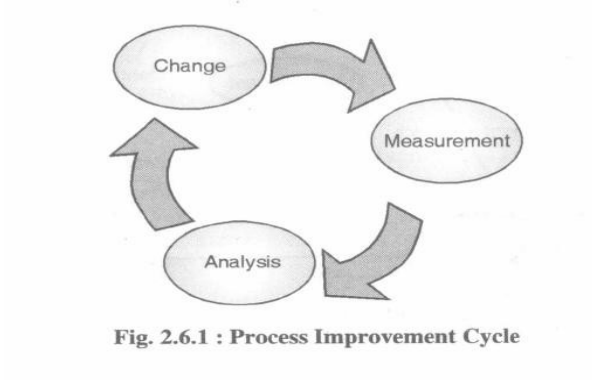


Fig. 2.6.1 : Process Improvement Cycle

It involves three principal stages:

1. Process measurement:

Attributes of the current process and product are measured. These form a baseline for assessing the improvements.

2. Process analysis:

- The current process is assessed and bottlenecks and weaknesses are identified. Process models that describe the system process are usually developed during this stage.
- It is about studying the existing processes to understand the relationships between different parts of the process and to analyze i.e. compare them with other processes.

3. Process change:

Changes to the process that have been identified during the analysis are introduced.

2.6.1 Need of Process Improvement:

- SPI framework defines the characteristics of an effective software process in an effective manner.

- SPI framework is used to assess existing organizational approach of software development against those characteristics.
- SPI framework defines a meaningful strategy for improvement.
- SPI framework transforms the existing approach of software development into more focused, more repeatable and more reliable process.
- • SPI framework assesses the maturity of an organization's software development process and identifies its maturity level.

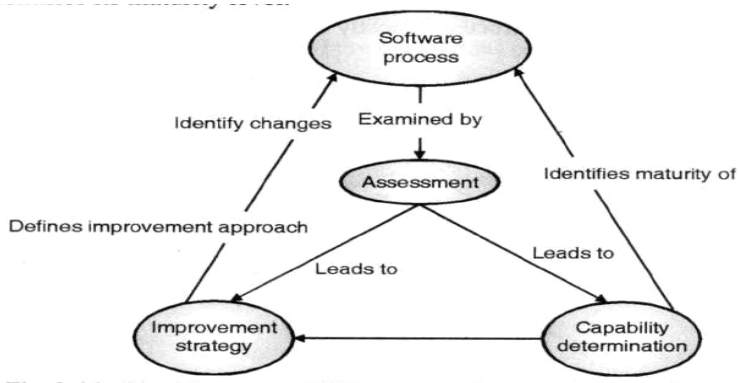


Fig. 2.6.2 : Key Elements of SPI Framework and their Interrelationship

2.6.2 Objectives of Software Process Improvement (SPI):

- To enhancecon currency activities-Fewactivities of aproject are done inparallel others in sequential. But if the parallel activities are carried out, this will reduce the cost and schedule. To minimize the overhead and instead use these efforts towards production activities.
- Few activities are overhead and others are production activities
- The Overhead activities include project planning, progress monitoring, risk management, configuration and version control, quality control, component integration, testing, rework, personnel training, etc.
- • The Production activities include the project itself that composes of requirements elicitation, modeling, analysis, and design and implementation activities.
- To eliminate the late rework and fixing.

Advantages:

A good-quality process reduces:

- The required efforts and ultimately reduces the required schedule The required Project time schedule but, yet with improved quality. The three main improvements are:
- Improves efficiency of each step in the process

Attributes	Meta-process	Macro-process	Micro-process
Metrics	Project predictability Revenue, market share	Predictability of budget and schedule Major milestone success Project scrap and rework	Predictability of budget and schedule Major milestone progress Release and iteration scrap and rework
Concerns	Bureaucracy v/s standardization	Quality v/s financial performance	Content v/s Schedule
Time scales	6 to 12 months	1 to many years	1 to 6 months

- Eliminates some of the overhead steps in the process which yields improved ROI. Carries out same number of steps but makes use of concurrency wherever possible

Table 2.6.1: Three levels of Processes and their Attributes

Attributes	Meta-process	Macro-process	Micro-process
Subject	Line of business	Project	Iterations
Objectives	Line-of-business Profitability Competitiveness	profitability Risk management Project budget and schedule Project quality	Resource management Risk resolution Milestone budget, schedule quality
Audience	Acquisition of authorities, customers Organizational management	Software project managers Software engineers	Subproject managers Software engineers

2.6.3 CMMIProcessImprovementFramework

• Capability Maturity Model (CMM) is a maturity model applied within the context of SP I framework.

CMM is a specific approach taken for quality assurance.

The CMMI (Capability Maturity Model Integration) is used to implement Quality Assurance in an organization.

- The CMMI describes an evolutionary improvement path from an adhoc, immature process to a mature, disciplined process which describe the key elements of an effective software process.

- The CMMI includes key practices for planning, engineering, and managing the software development and maintenance which helps in improving the ability of organizations to meet goals for the cost, schedule, functionality, and product quality.

- The CMMI helps in judging the maturity of an organization's software development process and compares it to the state of practice of the industry.

- The CMMI categorizes five levels of process maturity:

Level 1: Initial: An adhoc software process is used where only few of the processes are defined and where the project accomplishment success depends upon individual team member's efforts.

Level 2: Repeatable: A project management process is derived to track the utilized cost, schedule, and efforts. This same process is used again and again to develop the projects of similar applications so as to repeat the success achieved in the earlier projects.

Level 3: Defined: Both the activities i.e. project management and software engineering are well documented and integrated.

Level 4: Managed: It includes better understanding and planning of software process and product quality.

Level5: Optimize:The developing organization's focusison continuous software process improvement (SPI) which is achieved by continuous process feedback and by incorporating innovative ideas and technologies in project development.

- At this level, the entire organization focuses on continuous Quantitative feedback from previous projects which is used to improve the project management.

- The software process at this level can be characterized as continuously improving the process performance of their projects.

- Improvement is done both by incremental enhancements in the existing process and by innovations using new technologies and methods.

These levels are decomposed into several key process areas.

- **Process Change Management:** To identify the causes of defects and prevent them from re- occurring.

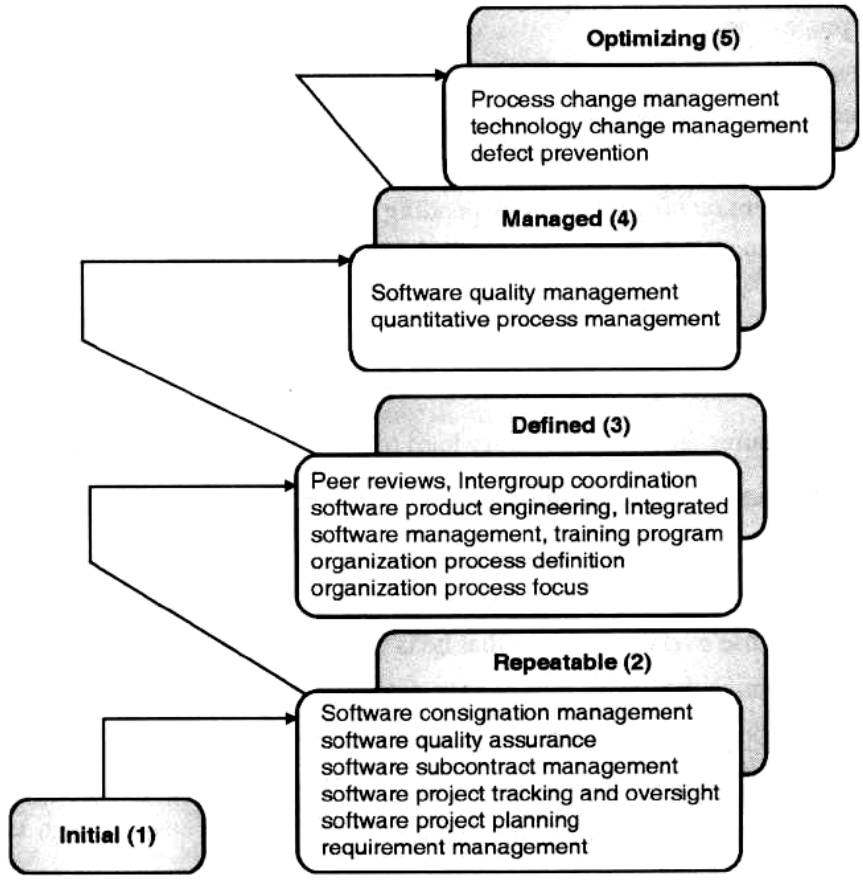


Fig. 2.6.3 : CMMI and Key Process Areas

- **Technology Change Management:** To identify beneficial new technologies and incorporate them in an orderly manner.
- **Defect Prevention:**To continuously improve the process in order to improve the quality productivity and thus decrease development schedule andcost.

The Process Area Activities performed include:

A software process improvement program is established which motivates the members of the organization to improve the processes of the organization. The group responsible for the organization's software process activities controls the software process improvement activities also. The organization develops and maintains the plan for improving the software process according to the documentedprocedure. The software process improvement activities are performed corresponding to the software process improvement plan. Members of the organization participate in teams to improve software process for assigned process areas. The software process improvements are installed to determine their benefits and effectiveness before they are introduced into normal practice. Records of software process improvement activities are maintained. Software engineers receive feedback depending on the status and results of the software process improvement activities on an event-driven basis.

Drawbacks of CMM:

The CMM does not describe how to create an effective software development organization. The traits it measures are practically very hard to implement in an organization.

2.6.4 Improving TeamEffectiveness

It has been observed that poor personnel yield poor productivity. But on the contrary, it is also impossible to manage a team with all stars in it. There are always disputes in such a great team because everyone thinks that he is more intelligent than the other. Managing the team is the key to improve the team effectiveness. Best pragmatic approaches to improve the team effectiveness are:

- **Balance:**A project development team must constitute highly talented people in key positions and less talented in other positions. It is easy enough to manage such a balanced team.

- **Coverage:**A project development team constitutes of strong skill people in key positions.

Boehm's Principles (Recommendations) in order to improve the team 's effectiveness:

- **Principle of top talent:** use talented and less number of people i.e. 'use better and fewer people'.

- **Principle of job matching (skills and motivations)** : Individuals in the development team must have a vision of promotion right from the programmer to project manager or to architect or to designer. All individuals in a team don't have same skill sets - Great programmers are not necessarily great managers and conversely,

- **PrinciplesofCareerProgression:**

i. An organization does best in the long run by helping its staff to self-actualize.

ii. Organization training greatly contributes in improving the productivity.

iii. Posting for new jobs must depend upon their skills and previous work area.

iv. Organization should focus on the factors that are the prime motivators such as increments, bonus andetc.

- **Principleofteambalance:**

i. Select people who will complement and go with one another.

ii. It represents the balance of: raw skills (intelligence, objectivity, creativity, analytical thinking)

iii. Psychological makeup (leaders and followers; risk takers, visionaries andnitpickers)

- **Principle ofPhase-out:**

Disrupt team balance, horribly de-motivating.

Availing a nonconformist in the team doesn't benefit anyone.

Overall team guidance:

- A culture of teamwork is necessary where people complement one another and go with each other.
- Balanced Teamwork
- Strong and 'knowledgeable' leader (Project Manager) is essential:

Required Project Manager Skills:

Hiring skills. Selecting right person for the right job.

Customer-interface skill. Avoiding adversarial relationships among stake-holders is must for success.

- **Decision-making skill.** Consider diverse opinions and make nopartiality
- Team-building skill. Keep the teamtogether.
- Recognize individual needs and excellentperformers Nurture the newcomers
- Facilitate contributions from everyone and make every individual feel that he is important.
- **Selling skill.** A successful project manager must:
  - Sell the stakeholders based on decisions and priorities,
  - sell candidates on job positions,
  - sell changes to the status quo in the face of resistance, and
  - sell achievements against objectives.
  - Practically, selling requires continuous negotiation, compromises, and patience.

Problems faced while achieving Team Effectiveness:

- It may not be possible to appoint the ideal people to work on a project because:
  - Project budget may not allow for the use of highly-paidstaff;
  - Staff with the appropriate experience may not be available;

- An organization may wish to develop employee skills on a software project.

Managers have to work within constraints especially when there are shortages of trained staff. The number of people working on a project varies depending on the phase of the project.

As more people work on the project, the more total effort is required.

Number of staff required can't be estimated by dividing the development time by the required schedule.

2.7 IMPROVING AUTOMATION

- The environment that is used means the tools and technologies that are used have a drastic impact on productivity and effort and thus, ultimately has an impact on schedule and cost also.
- Large number softools are available in the markets that are required for supporting a process.
- But...
- Make careful selection of the right combination of tools and recognize the tools that are important for process automation. Highly integrated tools facilitate proper management of the process
- A prime motivation for the staff is to train them to work with the modem tools and learn about the environment.
- Yields robust and integrated development process
- Hires talented and right people and equips them with modern tools.

2.7.1 Problems Faced while Improving Automation:

- While buying tools, be careful of tool vendor claims.
- Prior to using the tools, they must be integrated into the development environment.
- The table below represents the General Quality improvements realizable with a modern process.

Table 2.7.1 : General Quality Improvements with a Modern Process

Quality Driver	Conventional Process	Modern Iterative Processes
Requirements misunderstanding	Misunderstandings are discovered late	Misunderstandings are discovered and resolved early
Development risk	Risks are Unknown until late	Risks are detected and resolved early
Commercial components	Mostly unavailable	Available but the tradeoffs must be resolved early in the life cycle
Change management	Late in life cycle; chaotic and nasty	Early in life cycle; straight-forward and benign
Design errors	Such errors are discovered late	Any type of errors are resolved early
Automation	Mostly error-prone manual procedures	Mostly automated and error-free artifacts
Resource adequacy	Unpredictable	Predictable
Schedule	Unpredictable	Tunable to quality, performance, and technology
Target performance	Paper-based analysis or separate simulation	Executing prototypes, early feedbacks, quantitative understanding
Software process rigor	Document-based	managed, measured, and tool-supported

- Key practices that improve overall software quality:
- Focus on requirements driving the process - it addresses the critical use cases early in the life cycle and traceability late in the life cycle, focuses on requirement completeness.
- Use metrics and indicators to measure the progress and quality of the architecture as it evolves from a high-level prototype to a fully compliant product
- Use integrated life-cycle environment that facilitates early and continuous configuration and change control, fast design methods, document automation, and regression test automation.
- Use visual modeling and Higher-Level Language (HLL) that supports architectural control, abstraction, design reuse, reliable programming, andself-documentation.
- Early and Continuous insight into performance issues through demonstration-based evaluations.

Performance issues:

Be careful with commercial and custom-built components Performance analysis can degrade as we progress through our process.

2.8 PEERINSPECTIONS

- Conducting peer Inspections is a very old way of verifying the results.
- It is suitable and good in cases where there is a need to nurture less-experienced team members.
- It is also useful in catching the real bad errors early in the life cycle phases. Inspections can be applied at various phases during a development life cycle:

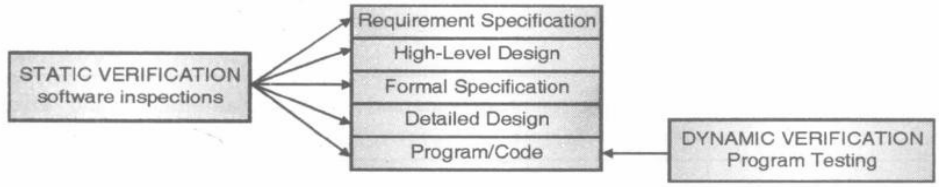


Fig 2.8.1: Software Inspection applied at any phase v/s Program Testing applied only at coding phase

- Inspections for transitioning the engineering info from one artifact set to another so as to assess the consistency, feasibility, understandability, and technology constraints involved in the engineering artifacts.

Example:

1. Analysis classes will transit into design classes which will then become the part of packages or components. In doing so, if any desired functionalities are lost then they are traced and re- included.
2. Inspections for demonstrating major milestones. This forces the artifact assessment against tangible criteria for relevant usecases.
3. Inspections of the Environment tools
4. Life-cycle testing provides insight into requirements compliance.
5. Inspections manage the changes and studies about how change requests can impact both quality and progress goals.

Inspection pre-conditions:

- A precise specification must be available.
- Team members must be familiar with the organization standards. Syntactically correct code or other system representations must be available. An error checklist should be prepared.
- Management must accept that inspection will increase costs early in the software process. Management should not use inspections for staff appraisal i.e. finding out who makes mistakes.

Inspection Process:

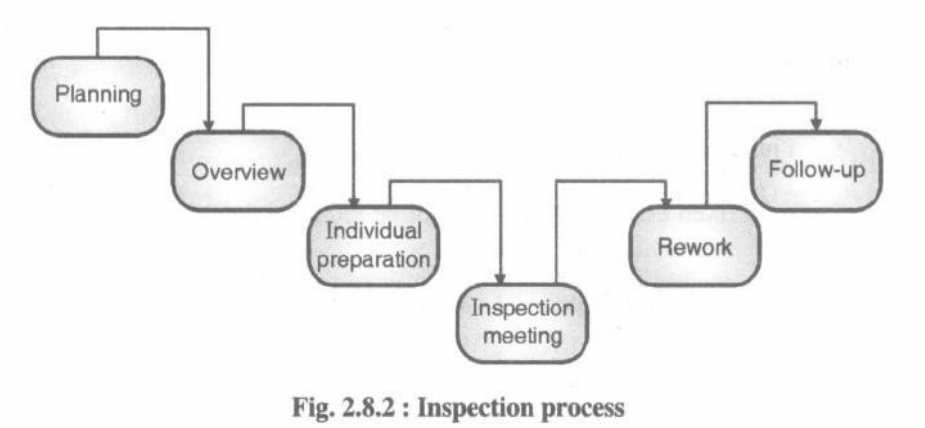


Fig. 2.8.2 : Inspection process

Step 1:Planning:

- Selection of the review team which must include not more than five people. Decide who will be the moderator and what his responsibility.
- Preparing the package (work product and supporting documents) that is to be distributed among the team members for their self-study.

Step 2:Overview:

- • All team members individually review the work product
- They answer the checklist according to the given guidelines and
- They note down the issues that they find in a self-preparation log

Step 3: Preparation:

- Each reviewer studies the project individually.
- He notes down the issues that he has come across while studying the project. He decides how to put up these issues and makes a note of it.

Step 4: Inspection Meeting:

- The reviewer reads line by line of the product Participants can raise any type of issue at any line Discussions are done to identify if any defect
- Scribe records all the decisions made in the meeting
- Scribe gives the list of detected defects (and issues) to the developer
- If there are only few defects, then the work product is accepted b; else it is sent for making changes in it and to undergo another review.
- The meeting does not propose any solutions, but if there are nay suggestions, then they are recorded and given to the developer.

- A summarized report of the meeting (in our language we say it as MOM i.e., minutes of meeting) is prepared which helps in effective evaluation of the product.

Step 5 &6: Rework and Follow Up:

- Author fixes the detected defects by making modifications in it. Once the errors are fixed, author gets it OKed by the moderator.
- Re-inspection on fixed errors may or may not be required depending upon the criticality of the problem ..
- Once the rework is completed and addressed satisfactorily, the collected data is submitted.

Inspection Roles:

Author (Owner) of the Code:	The programmer or designer responsible for producing the program or document. Responsible for fixing defects discovered during the inspection process.
Inspector	Finds errors, omissions and inconsistencies in programs and documents. May also identify broader issues that are outside the scope of the inspection team.
Reader	Presents the code or document at an inspection meeting.
Scribe	Records the results of the inspection meeting.
Chairman or Moderator	Manages the process and facilitates the inspection. Reports process results to the Chief moderator.
Chief Moderator	Responsible for inspection process improvements, checklist updating, standards development etc.

Advantages:

- The goal of this method is to detect ail faults, violations, and other side-effects. This method finds out more and more number of defects by:
- A complete preparation (studying the documents) is done by authors and other reviewers before conducting inspection.
- As a group of people are involved in the inspection procedure, multiple diverse views are enlisted.
- Every person of the inspection team is assigned a specific role.
- The reader in the inspection reads out the document sequentially in a structured manner so that all the points and all the code is inspected thoroughly.

Disadvantages:

- Needs lot of time as it first involves some special time for preparations also prior to conducting formal meetings.
- Logistics and scheduling always is a point for issue since these tasks constitute of lot of people. Checking every line of code is not possible every time even though it ensures the correctness of the logic, avoids the side-effects and appropriately handles the errors.

Inspection Issues:

- Ensure that complex and critical components are really inspected and verified by the primary stakeholders.
- It is difficult to really look at all artifacts.
- Inspecting too many artifacts will increase the cost.
- Mostly, many of the artifacts don't deserve any scrutiny.
- Most inspections end up looking at style and simple semantic issues rather than inspecting real issues.

Usually, most of the quality fact or sandother important features of the project such as system, performance, concurrency, distribution, etc. canbe discovered through following activities:

- Analysis and prototyping.
- Constructing design models will help in tracing the missing requirements and architectural constraints.
- Transitioning the current state of the designs into an executable implementation
- Illustrating the current implementation strengths and weaknesses in context of critical subsets of use cases and scenarios.
- Incorporating the user feedbacks into the models, use cases, implementations, and plans.

Review Questions

Q. 1 What is software economics and list out different cost estimation models.

Q. 2 Describe the cost estimation process.

Q. 3 Write short notes on cost estimation techniques.

Q. 4 List and describe the cost estimation parameters.

Q. 5 What is cost estimation and explain COCOMO model.

Q. 6 Explain the three generations of software economics.

Q.7 What are the various way of reducing the software product size. Explain in brief.

Q. 8 What is the need of improving the software process.

Q. 9 What are the benefits of improving team effectiveness and how to improve it.

Q. 10 How to achieve required quality?

Unit II

3

THE OLD WAY AND THE NEW

Unit Structure

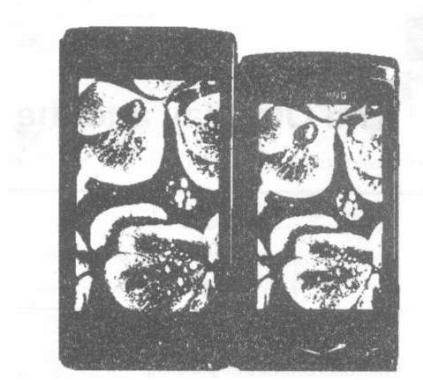
Syllabus:

The principles of conventional software engineering, Principles of modern software management, Transitioning to an iterative process

3.1 INTRODUCTION

As we know, every product in the market grows and gets updated every day. Each and every part of the product renews and re-polishes every minute.

Most of the organizations generate the new version of the older product by making various additional advanced changes in it. If we consider the example of the mobile company, one mobile comes in this one month, and the next generated advanced mobile comes soon in the next month. Following is the example of the Samsung mobile company which launched 'Wave I' in 2010 which was later upgraded with additionalvarieties(suchasenlargedsize)andcameupinthemarketas'WaveII'.



1 : Upgradation Versions: Samsung's Wave II and Wa

- Many years ago, in case of software companies, they can make naming to their upgraded version with ‘x’ suffixtoit, even they can not know the meaning of the ‘x’. Forexample:lx, 2x and soon.

- There are many parameters, by which software industry can improve the software economic development;

1. Require andMeet:

Initially, whatever matter was available with the developers, only by using that, the software product was generated. But now as per customer requirements and for his easy handling, the software market meets his requirements by meeting him.

2. Easy handling

It is seen that today’s customer can easily handle the overall software product by which earlier traditional workloads on him became very less. **Automation** came in picture by which customer can take more relaxation by putting his workload on the automatic planners.

3) Team Building

Many departments can handle different units of each different phase. So software project is able to achieve its goals by using different ‘thinking’ - ‘planning’ and most importantly ‘cooperating’.

4) Reduce in Size of Software 0 Product

As the new technologies arrived in market various compressed techniques are there in the market. Even the size of the product is much lesser which can be compatible for the customer’s point of view.

3.2 OLDWAY: BEGINNING OF THE SOFTWARE

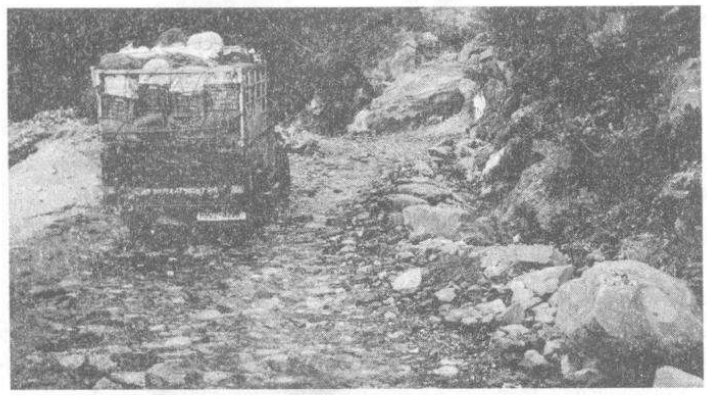


Fig. 3.2.1 : Old way for travelling

See the above picture in which a tempo travels on the road which is full of rocks and water. Tempo is heavily occupied with some containers and driver has overburden to drive the tempo till his destination.

If we compare the above example with software, it follows conventional Software project Matrix (SPM), and it includes the **Waterfall models** and **Software Management** performance which having various difficulties towards the achieve software goals.

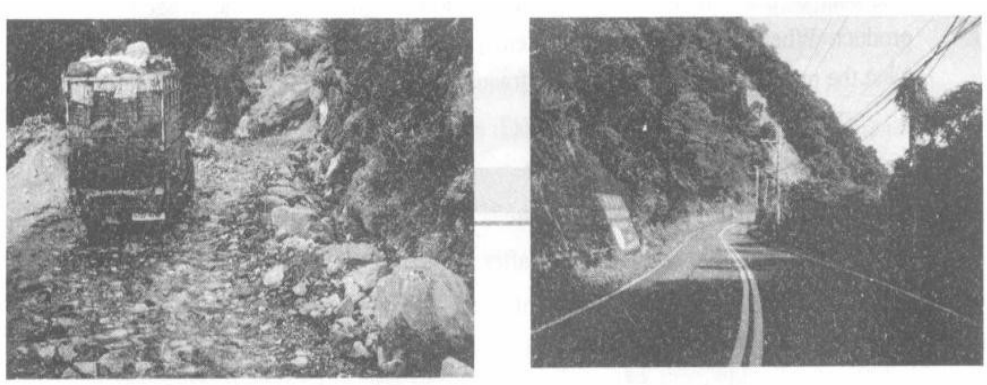


Fig. 3.2.2 : Comparison between old way and new way

As differentiation in above both figures, we make some points as

- Old roads are fullo frocks whilene ware smoot plane.
- Old roads are having water and hence sleepy are a while new roads having provision of the water circulation.
- New roads having in dicators to destination which cannot in older roads.
- Various provisions such as street lights, road divider are available with the new roads.
- Obviously, driver chooses the new way fortravelling.

3.3 THE PRINCIPLES OF CONVENTIONAL SOFTWARE ENGINEERING

Following are the principles of the conventional software engineering, which are applicable to software development life cycle phases which encompasses of the software.

1. Focus onQuality:

Qualitative software always achieve the goods goal son the customer side. The teams involvedin the designing the software are always have some goals. Team is made up of several members, which can have different quality aspects. The software can be made qualitative by making use of the qualities of each of those members in parallel way. If the lack of skilled members is used in the making of the software, then it might degrade the software quality. In that case hiring the skilled quality team

members are essential. Involving customer is another necessary, since finally he is user of the software. What is right and what is wrong is considered by his point of viewisveryes sential. Like wise, the team can inspect the project the goes in smooth handling or not? The prototype used for that are place better or not? Project has simplified design or not? These are some questions which gives the high-quality software.

2. RoleoftheCustomer:

End user of the software is always the customer who handle and usage the software product. When the customer can interact earlier with the software product, he judges what the problems before and after software product. He thinks on points;

- Time duration are reduced or not? Cost usages are reduced or not?
- How automation comes in picture?
- What kind of relaxation he has after software is in his hands? Which situation is better, current or earlier?
- Is the workload being really minimize? And so on ....

Answers of the all above questions are put by customer if and only if when he plays the software earlier.

1. **Quality Software:** Various parameters are there by which software maintains its quality. For example, we had taken in above point, which is involvement of customer in the quality consideration of the software. Other parameters are simple design, taking time to time inspection and appoint quality personals.

2. **Finding the Problem**: As the old way is fully depends upon the requirement and the model (water fall model) which we use din the old way is also called as 'Requirement-Driven Model'.But also defining the problem before the collecting and writing the requirement is very essential since most cases solution as per requirements are there but might be that solution can be in longer way. Since whatever alternative solutions are the re that are also comes into the picture. And this is done only when'exact problem is front of the software team'.

3. **Model Designing**: When architecture can design the building, before to that he makes plan and afterword he follows that plan for make successful construction. The same process here in the software modeling that means project has some rules and regulation that are binds with some corporate culture. Ready to accept the riskfactors.

4. **Solution with alternatives way:**When wediscuss about the ways there are lots of ways from source to destination and traveler always thinks that which way is fruitful for him. Here the meaning of the fruitful is that safety, short distance, less costly etc. Likewise in case of the software designing, it cannot bind with state forward way it has some alternative. That entire alternative must be proven at the time of writingalgorithms.

5. **Various Languages for different functionality:** There are so many solutions available today which converts the complex and difficult task to easier way. Various different principles are used to solve such complex problem. But user has satisfied is the lastmotive.

6. **Closerto real world problems:**Whenever all the real problems are satisfied then and then onlyreal-world problems are solved. This is done by minimizing intellectual distance.

7. **Right best than Rapid:** Rapid solution is always happy to customer. He thinks how his work rapidly goes on. But parallel to that it is also check that the job done is right way ornot.

8. **CodeChecking**: Software testing is more concern about the users at is faction More sophisticated way is that inspecting the overall design and codes for finding the errors withinit.

9. **Managementv/sTechnology:** Aswecomp are management and technology, technology requires the lots of resources then and then only it goes smoothly, best technology cannot compensate for poor management. But the quality of good manager is handling to poor management also in efficient way for producing the best results. Best manager handles his overall team to motivate the job. In other words, the best management is always better than best technology.

10. **Human resources are Success key:** The skilled work is not judged based on the proper and sufficient tools, technology desire language and the process which is used. It is based on the skillful person who knows the exact solution for the problem. This is not happened with the unskilled person or we can say that wrong person even if we provide him best technology, proper tool etc.

11. **Curiosity handling:** Whatever happening is might be correct, the answer may be appropriate, but if we cannot handle the processing very careful as per the environment which is not beneficial.

12. **Responsibility Handling:** As leader taking initiative in the managerial work is the most important task for the Management, like that at the time of the software designing, coding engineer must be taking initiative in taking the responsibility.

13. **Know the Customer Satisfaction:** The final product is handling by the customer. He is the end user of the product. What is actual his need, is very essential fact in case of the designing that product. When customer makes confirm with that product then and then only product designer has made success. In this case at the initial stage, it is important to that customer understand the customer priorities. The late delivery of product is might be good thing but the earlier delivery of product with wrong or less functionality is always wrong.

14. **Innovations are always welcome:** As per user requirements product are done, is ok with the user satisfaction. But word happy is more

fruitful as compare with word satisfied. Whatever more functionality with new innovations is provided to the customer he is wants that much performance via the product.

15. **Change becomesslowly:** The new innovations are not directly accepted by the customer. If the product is new brand with variety of innovation the customer has thought he is in risk to accept that product. He doesn't know about that product, whether that productis really good or not. So, there must be follow some basic facts which are customer previously known. Afterword new applications or algorithms may be produce with that only. Since new things may not be easy at the firsttime.

16. **Design for change:** As we renew our house, when there is function in house. As we feed of that renovation like that the architectures, components and specification techniques must be change within the software.

17. **Essentiality of Documentation:** Suppose one of the civil engineers complete the work of the bridge on the other road and after finishing the work he said that "I done bridge, now I see by paperwork that bridge has really good condition with that much columns". This is funny statement (Too much risky) since paperwork as drawing, checking is essential before to building that bridge. The height of that bridge might be passing down the truck.

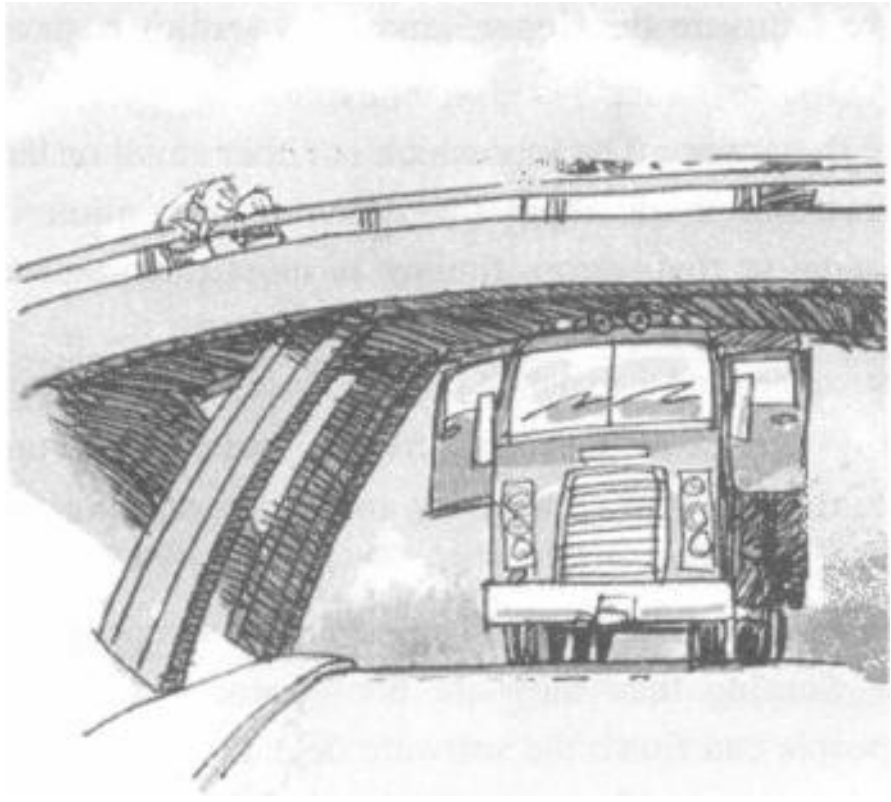


Fig. 3.3.1: Problematic bridge for large vehicles

Same to that, designing completed without documentation is not the proper work at all.

18. **Tools for its practical**: Byusing software project customer has comfort in his work is a fact. Like if we are use Microsoft Power Point for mock presentation then it is reliable, we will use word for typing the book.

19. **Ticks-less always better:** At the time of self-learning tricky programming is very essential for logic building. Some of the programmers are love to make tricky programming. But at the customer point of view, tricks are not better even if it is better for programmer or designer to do smart logical programming with less trickycodes.

20. **Encapsulation:** Encapsulation is the wrapping the several materials in single unit. By using that information to be hidden in simple way.

21. **Coupling and Cohesion:** What kind of maintain ability and a daptability, in heritsin software measured by the coupling and cohesion.

22. **McCabe complexity measure:** Even thought here are many number of metrics available to track and report the inherent complexity of software, but none of them is as intuitive and easy to use as Tom McCabe's. There fore, use of McCabe complex ityme as ure is insisted.

23. **Test by Others:** Asweknow customer is enduser of the software, heputshisre quirements in front of the software engineer. Software engineer afterword makes a plan for designs and developing. Once software completes then it is essential to test it to various different reasons. If we design and develop the software then we cannot test its own since software developers cannot test their own developed software. The reason is end user can handle software, whatever problems he faces earlier is minimize or not, is cannot judge by the software developer, since that testing is must done by the other person.

24. **Findout Cause of the error:** The bug which is either small or large, removing that is very essential. When we think about the thought 'prevention is better than cure', software logic is same to that, errors finding is most important task since, when we analyze them and prevent them it is cost effective method. When error is detected then and then only it is fixed, since this is analyzing method.

25. **Software's entropyincreases**: Any software system that under goes continuous changes in it will surely grow in complexity and becomes more and more unorganized and abundant.

26. **No relation between People and Time:** If we consider the people, who are engaging in the completion of working then they are not

depends on the time parameter. For example, if three people can finish the software development in one month, that cannot indicate that, they have ready 12 projects in one year. In other words, we say that, time and people are notinterchangeable.

27. **Expertexcellence**: We can produce good jobs (products) from our team. What ever final success is totally depending on our expertise team. Since here when we have more expectation from our employee then and then only our employee makes betterwork.

3.4 PRINCIPLES OF MODERN SOFTWARE MANAGEMENT

As per Davis format there are 10 principles of the modem management. These principles are placed in priority-wise as below;

1. **Architecture First Approach:** The design architecture primarily and the lifecycle plan. Afterword resources should be finalized for full scaledevelopment.

2. **Builditerative lifecycle process**: In the latest software industry it is not possible that fixed whole problem, as per that problem generate designs whole, make entire software and finally test it that means whole in proper sequence. In the iterative process problems can be rectify previously and give proper solution for that. Most of the risks are known earlier, which is useful for increasing predictability and use to avoid expense of repetitive work.

3. **Component Based Development**: Ascustom development is possible rat her than human generated source code then it is possible to move on line of code generation to component-based code generation. Since component consists of executable format predefine source code.

4. **Change Management Environment**: Changeine very factisessential, it is giving the dynamiciterative model. If we use different - different peoples or teams for the same job then that is workflow follows the change management environment.

5. **Round Trip Engineering:** Automation in every sector of software is very essential since without to that automation bookkeeping, changing, designing, documentation, coding and finally testing is difficult. In iterative process change liberty is essential. These are the top 5 principles in software management. Following figure depict it in brief idea.

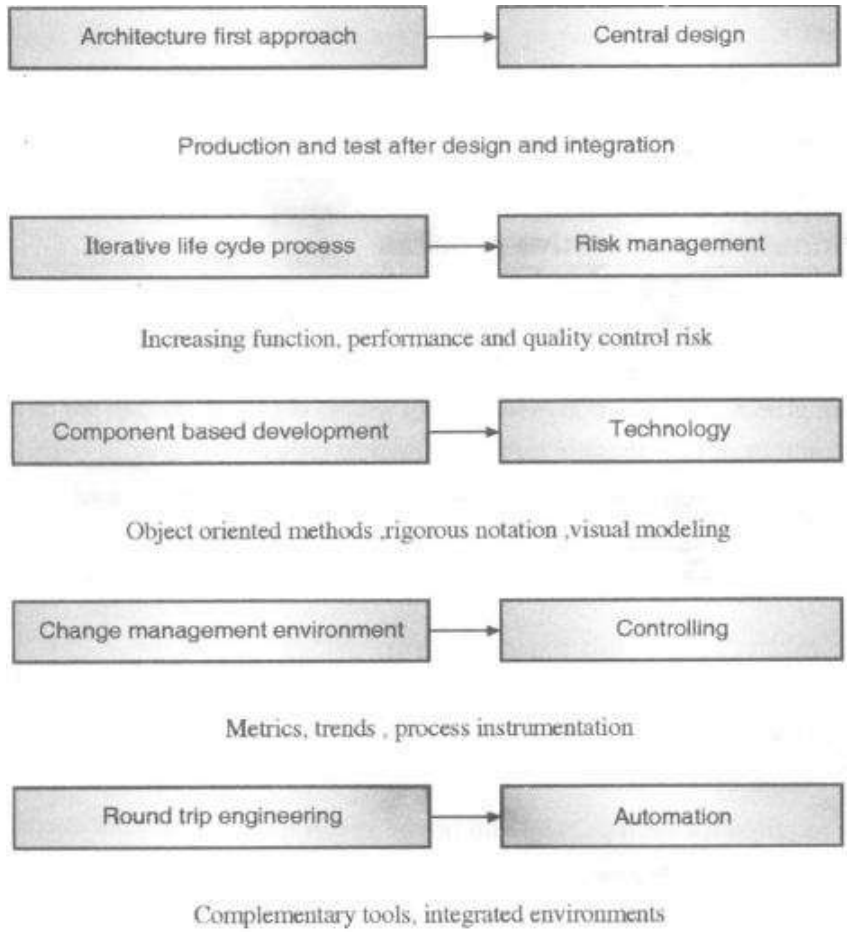


Fig. 3.4.1: Principles of software management

(i) Model Based notation

Develop design artifacts using model-based notations. This makes the reviewing easy compared to the inspection process.

(ii) Objective Quality Control

The quality can be controlled by assessing the progress. Once the assessment is over, it is integrated with the development process.

(iii) Demonstration based Approach

This approach is used to assess intermediate artifacts. This approach is generally applied to early prototypes, baseline architectures, and early releases.

(iv) Evolvinglevelsofdetails

Using these evolving levels of details, the intermediate releases are planned so as to allow early and continuous releases. And these releases are accompanied with corresponding use-cases and scenarios.

(v) ConfigurableProcess

A configurable process is adopted that is economical and is scalable across a wide range of projects.

3.5 TRANSITIONING TO AN ITERATIVEPROCESS

First of all, we have a question i.e., why waterfall model (which is traditional model in the software engineering) is no longer used? The answer is that, in waterfall model, each phase depends on the previous phase. It indicates that, the output of one phase is input for the next consistent phase. The latest modern processes depend upon the initial version of the system. E.g., Spiral, Increments, Generations, Release.

Drawbacks of the Waterfall Model are as follows:

1. Real products rarely follow this sequentialflow.
2. It is very difficult for customer to state all the requirements in onetime.
3. Many projects face this requirement uncertainty at beginning itself and so it is very difficult to design nextphases.
4. Time span required for each phase could not bespecified.
5. Naturally project requires moretime.
6. Project becomes lengthy also.
7. Customer should have patience.

Iterative Process:

- It can also be called as Incremental Process Model and the basic idea is that the software is developed in increments, where each increment adds some functional capability to the system until the full system isimplemented.
- In iterative enhancement, extensions and design modifications in the project can be made at each step.

Need of Iterative Process:

- The iterative life cycle model removes the limitations of the waterfall model and tries to combine the benefits of both prototyping and the waterfallmodel.
- The iterative process models combine the elements of waterfall models applied in iterativefashion.

1. It adapts all the phases of waterfallmodel.
2. It accepts linearprocess.

3. It tries to solve one by one problems of the customer, or it accomplishes one by one requirement of the user, which is called as incrementaliterations.
4. Generally, first increment is nothing but core product of thecustomer.
5. As time grows, it is incremented from one requirement to next requirement or simultaneously it performs both the requirements (i.e., old requirement as well as starts new requirements)

Advantages of Iterative Process:

1. Allows early development of initial versions.
2. Risk areas are addressed early in project life cycle phases.
3. Several iterations are developed.

Examples: Incremental model, Rapid Application Development model (RAD), Spiral model

1. It can result in better testing, since testing each increment is likely to be easierthan testing entire system like in the waterfall model.
2. Asinpro to typing, the increment provides feedback to the client, which is useful for determining the final requirements of the system.

Review Questions

Q. 1 Discuss the parameter which are used for economic development.

Q. 2 How conventional software model focuses on Quality?

Q. 3 Explain Role of customer in conventional software engineering.

Q. 4 Explain S principles of modern software management?

Q. 5 What are drawbacks of waterfall model?

Q. 6 Why iterative process is needful? What are its advantages.

4

LIFE CYCLE PHASES

Unit Structure

Syllabus:

Engineering and Production Stages, Inception, Elaboration, Construction, transition Phases

4.1 SOFTWARE PROCESS

A software process identifies a set of activities that are applicable to the development of any software project, regardless of their size or complexity. A software process is a collection of work activities, actions and tasks that are to be performed when some software project is to be developed.

Software process can be categorized into:

1. Generic Process model - represents a framework activity populated by a set of software engineering activities.

4. Personal and Team Process models - This model helps in creating a software that best fits either the personal needs of the user or that meets the broader needs of a team.

3. Prescriptive Process models - provides an ordered structure and an effective roadmap to software engineering work.

4.2 GENERIC PROCESS MODEL

This model defines a set of umbrella activities which are also a must for any software engineering process as shown in the below figure.

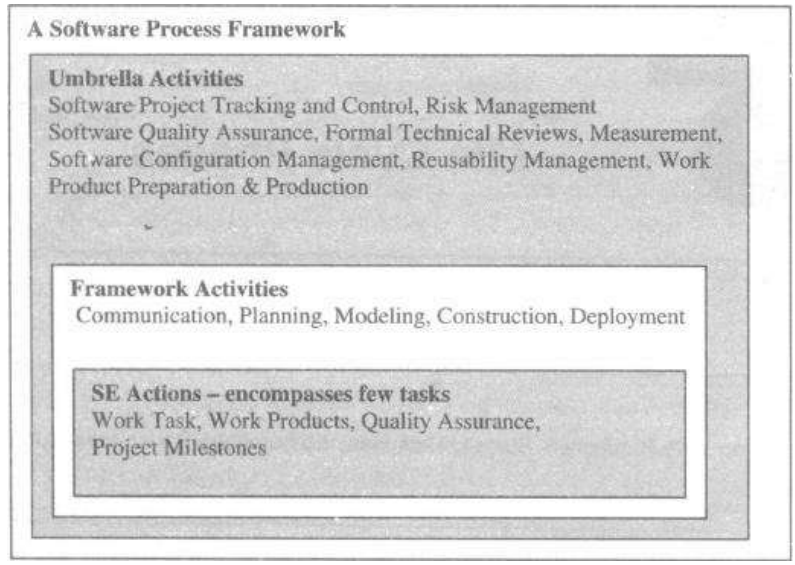


Fig 4.2.1: A Software Process

Again, each framework activity contains a set of software engineering activities which is a collection of tasks that develops a major software product.

4.2.1 Framework Activities

First, we take a look at the generic process framework activities that are must for the development of any software project:

- **Communication:** The main cause of communication is requirement gathering. This activity establishes sufficient interaction or collaboration between the developer and the customer for gathering the requirements and knowing the expectations of thecustomer.

Example: we can explain the **work task** regarding the **communication activity** of a simple projectas listed below:

- Making the list of the end-users, software engineers and support people for theproject.
- Inviting all of them for an informalmeeting,
- Ask each end-user to make a list of features and functionsrequired,
- Discuss these requirements and prepare a finalist,
- Arrange the requirements according to their priority,
- Note the areas of uncertainty.

- **Planning:** This activity defines the software development process to be conducted. It describsall the needed technical tasks, possible risks, the resources that are required, the work product to be produced and the schedule to work out the whole process. This activity plans the work, identifies the resources, tasks and sets the schedule.

- **Modeling**: This activity creates a model (blueprint) which clearly describes the software requirements and the design that will achieve these requirements. This is helpful to both customer and the developer respectively, to understand what he wants from the software and how he can develop it. Modeling is composed of two main activities-analysis (requirements gathering, elaboration, negotiation, specification and validation) and design (data design, interface design and each module leveldesign).

- **Construction:** This activity includes code generation either manually or using automated tools and then testing the code to correct the errors if any.

- **Deployment:** The software (as a complete product or in a partial stage) is delivered to the customer who then checks the product and provides feedback on evaluation. The framework activities are applied on every project but the degree of tasks depend on the:

o Type of the project

o Characteristics of the project

o Agreement of the project team on common views.

4.2.2 Process Iteration and Activities

The above framework activities discussed in sec 4.2.1 occur in an organized pattern with respect to sequence and time. This work flow pattern of the activities is termed as 'Process Flow'

1. **Linear Process Flow:** Executes the five framework activities in a sequence starting with 'communication'andendingwith'deployment'.

Fig. 4.2.2: Linear Process Flow

2. **Iterative Process Flow:** Repeats one or more of the five framework activities before proceeding to thenext.

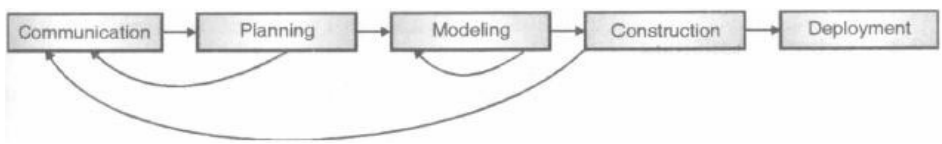


Fig. 4.2.3: Iterative Process Flow

3. **Evolutionary process Flow**: Executes the five frame work activitiesina "circular/cyclic"manner.

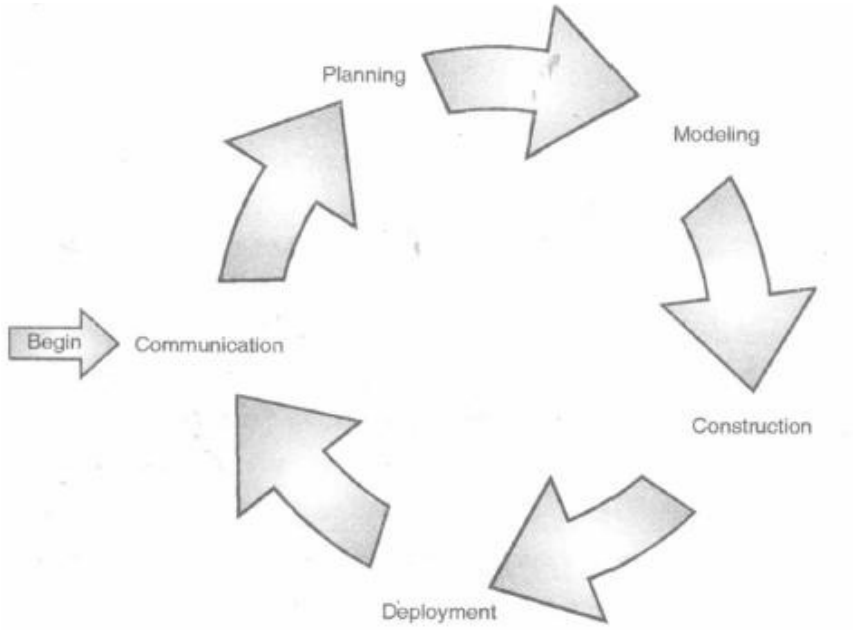


Fig. 4.2.4: Evolutionary Process Flow

4. **Parallel Process Flow:** At a time, executes one or more activities i.e. one or more of the five framework activities are executed in parallel with the other. Say, modelling of one module is executed parallel to the construction of another module.

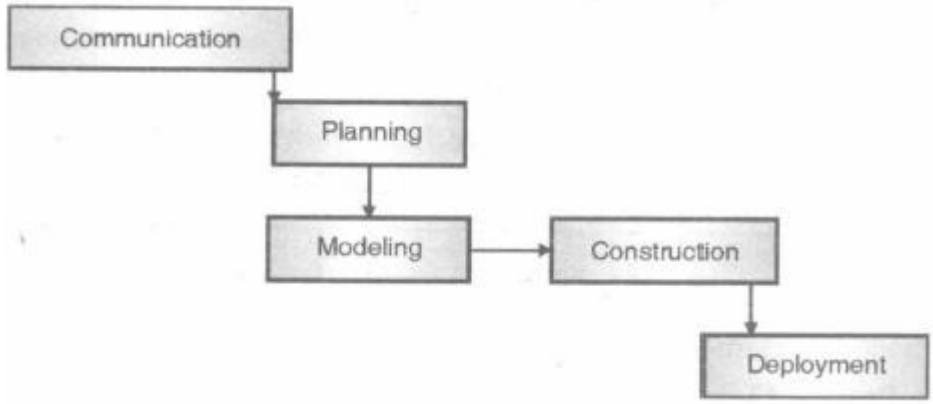


Fig. 4.2.5: Parallel Process Flow

4.2.3 Umbrella Activities

Now, we look at the **Umbrella activities** that are applicable throughout the software process:

- **Software Project Tracking and Control**: Software teamassesses the progress of the project plantime to time and takes necessary action to maintain the schedule. Thus, software team tracks and controls the projectschedule.
- **Risk Management**: Software team assesses the risks that may affect the out come of the projectorsay the quality of the product.
- **Software Quality Assurance**: Software team defines and conducts the activities needed to preserve the quality of the software product.
- **Formal Technical Reviews**: Software team assesses the technical efforts to find and remove the errors before they are forwarded to the nextaction.
- **Measurement**: Just the coincidence is the four P'sof Software Engineering: Project (the taskat hand), Process (the manner it is done), Product (the object produced) and People (by whom it is done). Software team collects all the project, process and product measures so that it can be used in combination with all other framework and umbrella activities.
- **Software Configuration Management**: Software team takes essential steps to manage theaf fects of changes made throughout the softwareprocess.
- **Reusability Management**: Software team defines criteria for work product reuse and establishes mechanisms to achieve reusable components.
- **Work Product Preparation and Production**: Encompasses the activities required to create work products such as models, documents, logs, and forms and lists.

4.3 PERSONAL AND TEAM PROCESSMODELS

- The whole process of developing aproduct through out the project is handled by the people working on it.
- The software process can either be **personal** or **team**built.
- Some small hardware or software products can be developed by individuals (personals), but the scale and complexity of modem systems is such, and the demand for short schedules is so great, that it is no longer possible for one person to do most engineering jobs. Systems development is a team activity, and the efficiency of the team largely decides the quality of theengineering.

4.3.1 Personal Software Process(PSP)

- The PSP provide sengineers with adisciplined personal frame work for doing software work.
- This personal frame work of PSP model consists of five mainactivities:

1. Planning:

This activity isolates and defines the requirements and based on these, develops the size and resource estimates and also identifies the probable defects. All these metrics are recorded on worksheets, first they are analyzed and then finally the development tasks are identified and the project schedule is created.

2. Highleveldesign:

This activity identifies the external specifications needed to construct each component and accordingly the design is created. And when there is uncertainly, the prototypes are built.

3. Highleveldesignview:

This activity performs the verification methods to find and remove the errors in the coding or design.

4. Development:

Every module (component) design is reviewed in detail. Accordingly, the corresponding code is also generated, reviewed, compiled and tested so as to achieve 'zero-defect' product.

5. Postmortem:

All the four activities described above are controlled, tracked and recorded time to time properly on the worksheets. Using the information collected from the above activities, the effectiveness and the quality of the software development process is determined. This information helps insuggesting any changes in the process, if required, so as to improve the software quality.

The PSP helps software engineers to:

- manage the quality of their project
- make commitments they can meet
- improve their estimating and planning skills
- reduce the defects in their work and ultimately in their product. This is achieved through a rigorous assessment of all activities done by the software engineer.

- The goal of PSP is to help developers produce quality and zero-defect products on time. For example; Motorola division in Florida achieved zero defects in over 18 projects through implementing PSP technique.
- The PSP is a prerequisite for an organization i.e. planning to introduceTSP.
- The PSP can be applied to many parts of the software development process such as:
    - small-program development
    - requirement definition
    - document writing
    - systems tests
    - systems maintenance
    - enhancement of large software systems

4.3.2 Team Software Process(TSP)

The common definition for a team:

- A team consists of at least two people.
- The members are working toward a common goal.
- Each person has a specific assigned role which avoids conflict between the team members, avoids duplicate work and time wastage.
- Completion of the mission requires some form of dependency among the group members. Each team member depends to some degree on the performance of the other members. Interdependence improves individual performance because the members can help and support each other.
- The goal of TSP is to build "self-directed" project team that organizes itself to produce high quality software. The objectives of TSP are to:
- Build self-directed teams that plan and track their work, establish goals and own their processes and plans. These can be pure software teams or integrated product teams of 3 to about 20 engineers.
- Provide a simple process framework based on the PSP.
- Show managers how to coach and motivate their teams to sustain peak performance. Use modest, well-defined problems.
- Develop products in several cycles.
- Establish standard measures for quality and performance. Provide detailed role definitions.
- Use role and team evaluations. Require process discipline.

- Provide guidance on teamwork problems.
- First version of the TSP process was developed in 1996 by Watts Humphrey. His objective was to provide an operational process to help engineers consistently do quality work.

TSP helps the engineers to:

- ensure quality software products
- create secure software products
- improve process management in an organization

Each project is launched using a sequence of tasks that enables the team to establish a solid basis for starting the project. The TSP launch process includes below tasks:

- establish product and project goals
- define and assigning team roles
- assess risks
- develop the quality plan and set quality targets
- plan for needed support facilities
- produce an overall development strategy
- make a development plan for the entire project
- make detailed plans for each team member for the next phase
- merge the individual plans into a team plan
- rebalance the team workload to achieve a minimum overall schedule
- Assess project risks and assign tracking responsibility for each key risk.
- After the launch, the TSP provides a defined process framework for managing, tracking and reporting the team's progress.

4.4 PRESCRIPTIVE PROCESSMODELS

Definition: Prescriptive models define a discrete set of activities and actions to accomplish all tasks of the software with milestones, which is used to develop the software. These Process models may not be perfect but they give very good guidance in software development process.

Uses

It is used by

1. Software engineer.
2. Manager.
3. All employees who play important role to develop the software.

Importance

This model is important because,

1. It provides stability
2. It provides control for well organization activity.
3. It is also referred as rigorous model.

Steps

This model takes following steps:

1. This process guides software team.
2. It generates frame work activities to organize into a process flow.
3. Process flow may be linear or incremental or evolutionary.
4. The terminology of each model is different.

Work product

The work product is the programs, documents and data that produce a sequence of activities as well as task defined by the system.

Ensure

This mechanism determines the maturity of the software using quality, timeliness and long-term validity of the product.

The best indicators are the users who use the product and judge the efficiency.

The prescriptive model also called as a conventional process model. In short...

1. It describes a unique set of frame work activities.
2. It should populate framework activities to set a software engineering action.
3. These actions are used to create work product to accomplish to meet development goal.
4. It finds out- the nature of project, whether is suitable for the people using it, whether itis suitable for the environment, where it is implemented.
5. Framework activities are communication, planning, modeling, construction, and deployment.
6. It prescribes a set of process elements, framework activities, software engineering actions tasks, work products quality assurance, change control mechanism of each project.

Examples: The various prescriptive (conventional process) models are waterfall model, incremental model, rapid application development model, prototyping model and spiral model.

4.5 LIFE CYCLE PHASES

- Mostly, all software development process overemphasizes either on: i) research and development or on ii) production.
- But, characteristic of a successful software development process is achieved by a proper separation between i) "research and development" ii) activities and "production"activities.
- Projects developed under the conventional process have a very precise project milestone when there is a transition from a research step to a production step. The initial phases focus on achieving functionality and the later phases focus on achieving a product that can be delivered to the customer, with special attention on the product ‟robustness, performance, and finish.
- Similarly, the modern software development process must be defined to support the following in-order to achieve successful software development process:

o Designing the plans, requirements, and architecture, together with well-defined clear and precise points,

o Managing the risks and measuring the objectives to achieve quality.

- Developing the system capabilities by increasing the system functionalities.

4.5.1 Engineering and ProductionStages

To achieve higher returns on investment, there is a need to adopt such a software manufacturing process which is driven by technological improvements in process automation and component- based development.

The two particular stages of the life cycle are:

1. The **engineering stages** are less predictable and have smaller teams involved in design and synthesis activities. It focuses on risk reduction, prototyping, establishing architectural baseline, analysis, design, and planning.
2. The **production stages** are more predictable and have larger teams involved in construction, testing, and deployment activities.

Table 4.5.1: Two Stages of the Life Cycle - Engineering and Production

Life-Cycle Aspect	Focus of Engineering Stage	Focus of Production Stage
Risk reduction	Schedule, technical feasibility	Cost
Products	Architecture baseline	Product release baselines
Activities	Analysis, design, planning	Implementation, testing
Assessment	Demonstration, inspection, analysis	Testing
Economics	Resolving diseconomies of scale	Exploiting economies of scale
Management	Planning	Operations

Engineering stage is carried out in two distinct phases - inception and elaboration Similarly, Production stage is also carried out in two distinct phases - construction and transition.

All these four phases of the life-cycle process are loosely mapped into a conceptual framework of the spiral model as shown below;

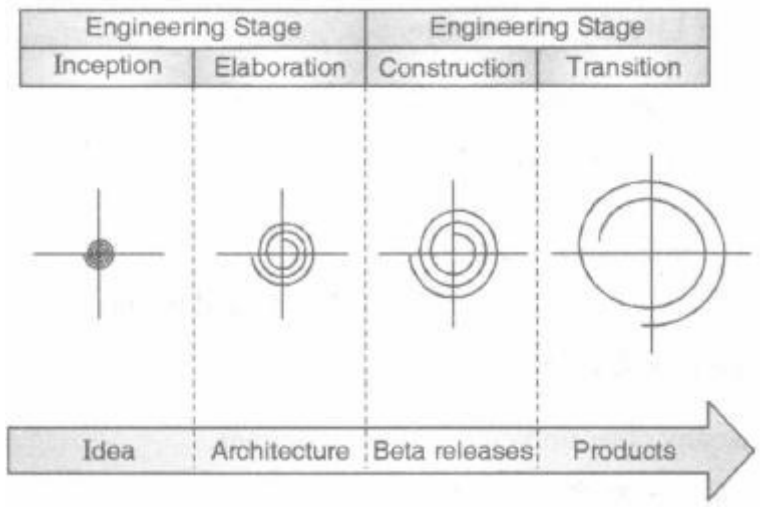


Fig. 4.5.1: Four Phases of a Life Cycle Process

These life cycle phases are very widely practiced and adopted in the industries.

4.5.2 Features of Life Cycle Phases:

- The most important feature/idea is Iterative Development. Iterative Development is sequentially expanding and refining a system through multiple iterations, using feedback and adaptation.
- Each phase has iterations, each having the purpose of producing an executable piece of software. The duration of iteration may vary from phase tophase.

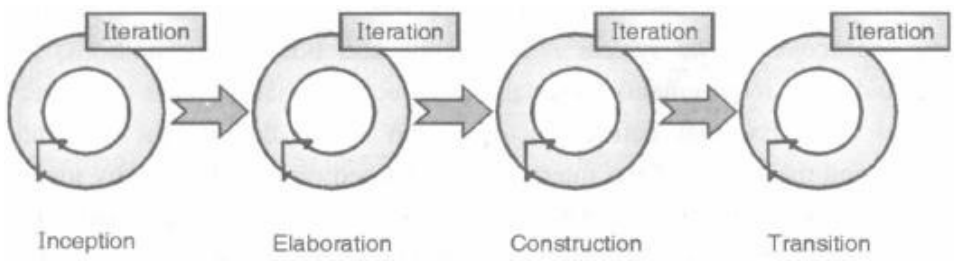


Fig. 4.5.2: Four phases of Iterations

- It is a lightweight process addressing the needs of small projects -to more comprehensive process addressing the needs of large projects.
- It helps in early and continuous documentation of the most urgent and the most probable risks by proper planning and keeping a follow up. This helps in mitigating the risks at early phases of software development.
- It also uses visualization methods such as an UML to build models so as to understand the complexity of the system.
- Italsousesuse-casesastestcaseswhichallowsend-userdocumentationandhelpsindesigning.

4.5.3 Advantages of Life Cycle Phases:

It emphasizes on addressing very early high risks areas.

It does not assume a fixed set of firm requirements at the inception of the project, but allows refining the requirements as the project evolves.

It does not put a strong focus on documents.

The main focus remains the software product itself, and its quality.

4.5.4 Drawbacks of Life Cycle Phases:

It fails to provide any clear implementation guidelines, leaves the tailoring entirely to the user.

4.6 INCEPTION

Inception means start i.e.; this is the point where the project is proposed.

Goal of Inception:

- To achieve concurrency among the stakeholders of the system on the objectives decided for the project.

Objectives of Inception Phase: It constitutes of Business modelling i.e.

- **Define the problem:** The objectives of the project are stated, so that the needs (requirements) of every stakeholder are considered.
- **Define the scope of the system:** The Scope and boundary conditions, acceptance criteria and some requirements are established such as what is and is not expected to be in the product. External entities (actors) with which the system will interact are identified and the nature of the interaction is defined on a high-level by identifying all use cases. It also includes identifying the business case i.e. identifying the success criteria, risk assessments and estimation of the resources needed and a phase plan showing dates of major milestones.
- Separate the vital use cases of the system and the main scenarios of the operation that may drive the major design trade-offs.
- Demonstrate at least one candidate-architecture against some of the primary scenarios.
- Estimate the cost and schedule needed for the entire project.
- Estimate the potential risks i.e. the sources that may cause unpredictable problems.
- Initiating the Project: And then, you can start working on the project.

Activities in Inception Phase:

- To define the project scope: The scope of the project is defined in the SRS (Software Requirement Specification). This also defines the problem and derives the acceptance criteria for the proposed product.
- To develop the architecture: The SRS must reveal the feasibility of at least one candidate- architecture and also must define an initial baseline of decisions so that the required expenditures (cost), schedule needed for development and delivery, and required resources can be estimated.
- To plan and prepare a business case : This involves the evaluation of the alternatives proposed for risk management, staffing, iteration plans, cost,/and schedule/profitability.

Evaluation Criteria:

- Do all stakeholders agree on the defined scope and cost and schedule estimates?
- Are requirements precise, clear and understood as defined in the use cases?
- Are the cost and schedule estimates, priorities, risks, and development processes realistic?

- Does the depth of an architecture prototype define the preceding criteria? i.e., does it provide a vehicle for understanding the scope and assessing the efforts of the development group in solving particular technical problem.
- Does actual resource expenditure match approximately with planned expenditures?

Feasibility Study in Inception Phase:

- A new project is started when a new business need is identified or a new service is discovered. Stake holders of the business (business managers, marketing people, and product managers) define a business case for the idea, try to identify the breadth and depth of the market, do a rough feasibility analysis, and identify the project scope. All this information is enough to discuss with the software engineer to start a project.
- The feasibility study made by stakeholders can be as below which decides whether or not the proposed system is useful. The study checks:

o Will the system add to organizational benefits?

o Can the system be engineered using current technology and within budget?

o Can the system be integrated with other systems that are used?

Interviewing the Customers and Users in Inception Phase:

At project Inception time, software engineers ask some questions to the people in the organization based on the information collected:

- What if the system wasn't implemented?
- What are current process problems?
- How will the proposed system help?
- What will be the integration problems?
- Is new technology needed?
- What facilities must be supported by the proposed system?

The need of these questions is to find:

o The effectiveness of collaboration between the customer and developer,

o Basic understanding of the problem,

o Who will use the solution?

o The desired nature of thesolution.

Inception Outcome:

- Vision document
- Initial use-case study (10%-20%complete)
- Initial business case
- Initial risk assessment
- Project plan
- Stakeholders decide whether to commence a full scale project ornot.

4.7 ELABORATION

- Elaboration means refinement (careful development). This phase is the end of "engineering" stage.
- In this phase, an executable architecture prototype is developed in one or more iterations depending upon the scope, size, &risk.
- Elaboration is about creating an analysis model that defines the informational, functional and behavioral aspects of theproblem.
- The main task is to describe the problem in a way that establishes a firm base for designing a model.
- Elaboration focuses on expanding the information (i.e. obtained from inception and elicitation) and then developing a refined technical model of software functions, features and constraints (i.e. restrictions or limitations). This process is composed of various modelling and refinement tasks.
- Different models may be produced during this activity depending on the relationships and collaboration between the various business domain entities.
- From the designed model, it would be easy to judge if the efficiency of workflow of the system is as it has been imagined.

Goal of Elaboration Phase:

- This phase gives you a mile wide and inch deep view i.e., little bit deeper view of the system.
- Detailed analysis of the problem resulting in the definition of an architectural foundation for the project. It constitutes of requirements, analysis and design phases.
- This phase ensures that:
  - the architecture, the requirements, and the plans are defined and there is no change now;
  - the risks are traced and fixed;

- the schedule for completion of development is predicted within an acceptance criterion.

Objectives:

- baselining an architecture as rapidly as practical
- baselining a vision
- baselining a sound plan for the construction phase
- Representing that the baseline architecture supports the vision at a reasonable cost in a reasonable time

Activities in Elaboration Phase:

- Detailing the vision.
- Detailing the process and infrastructure.
- Detailing the architecture and defining the required components.
- Eliminate the highest risk elements of the project.

Evaluation Criteria:

- Is the defined visions table?
- Is the defined architectures table?
- Are the major risk elements eliminated from the process?
- Is the construction phase plan reliable and are the required estimations done?
- Do all stakeholders agree that the current vision can be achieved by following the current plan for developing the complete system in context of the current architecture?
- Does actual resource expenditure match approximately with planned expenditures.

Elaboration Outcome:

- Use-case model will be 80%complete.
- Additional requirements capturing the non-functional requirements and requirements not associated with a specific use-case are identified.
- Description of the Software Architecture.
- An executable architectural prototype is developed.
- A revised risk list and revised business case is developed.
- A development plan of the whole project. This defines the iterations and evaluation criteria needed for each iteration and also the process i.e. to beused.

4.8 CONSTRUCTION

Construction means to build i.e., it is a manufacturing process. Aim of Construction Phase: It constitutes of implementation i.e. detailed design and construction of source code.

Objectives of Construction Phase:

- To minimize the software development cost by reducing the required number of resources and avoiding the unnecessaryrework.
- To achieve enough software quality as fast aspossible.
- To keep track of program versions (such as the alpha, beta, and other test releases) as fast as possible.

Activities in Construction phase:

- Emphasizes on managing resources and controlling operations to optimize costs, schedules and quality. This phase is broken into severaliterations.
- In this phase, all the remaining components and the remaining application features are integrated into one application and then all the features are thoroughly tested after the integration.
- Assessing the product releases against acceptance criteria of thevision

Construction Outcome:

- An executable product that is ready to put in the hands of the endusers.
- The software product integrated on the adequate platform.
- A usermanual.
- Description of the currentrelease.
- This is considered as a betarelease.

Evaluation Criteria:

- Is the product baseline stable enough so as to be deployed on the userside?
- Are the stakeholders ready for transition of the product on to the userside?
- Does actual resource expenditure match approximately with plannedexpenditures?

4.9 TRANSITION

Transition means delivery. The transition phase is the phase where the product is put in the hands of its end users. This phase is entered when the product is mature enough to be deployed on the enduser's site.

Goal of Transition Phase:

It constitutes of deployment i.e., delivery of the system to the user community. It involves issues of marketing, packaging, installing, configuring, supporting the usercommunity, making corrections, etc.

Activities in Transition Phase:

- Deliver the software product to the usercommunity.
- Issue newreleases
- Correcting problems (ifany)
- Finish the features that werepostponed
- Perform beta-testing to validate new system against userexpectations
- The system might run in parallel with a legacy system that it isreplacing
- Training of usermaintainers
- Roll-out the product to marketing, distribution, and salesteam

Objectives:

- Achieving user self-supportability
- Achieving stakeholders‟ agreement that deployment is complete andconsistent.

The transition phase ends when the deployment of the product achieves the complete positive agreement of the customers and users.

Evaluation Criteria:

- Is the customer and user satisfied?
- Are actual resource expenditures versus planned expenditures acceptable?
- Does actual resource expenditure match approximately with planned expenditures?

Transition Outcome:

- Matured enough product that can be deployable on user‟ssite.

4.9.1 Six Best Practices

Life Cycle Stages is built on the "Six Best Practices" :

1. Develop iteratively:

- Software must be developed in small increments and short iterations.
- An iterative process breaks a development cycle into a sequence o f 4 phases each of which includes a series ofiterations.

2. Manage requirements:

- It allows accommodating requirement changes in system development strategy.
- Those requirements that change over time and those requirements that have greater impact on project goals are identified.
- It is a continuous process to identify requirements.
- Managing requirements include:

o Elicit, organize (according to the priority), and document the required functionalities and constraints,

o Evaluate the impact of change sand

o Track and document the decisions.

3. Use component architecture:

- The process focuses on early development and design of independent executable modules, prior to committing for full-scale development.
- Components that are most likely to change and components that can be re-used are identified and built.

4. Model visually:

- Models must be built using visualization methods like that of the UML, to understand the complexity of the system.
- This helps you to understand the different aspects of your software and see how the different elements of the system communicate with each other.
- Maintains uniformity between design and its implementation.
- Promotes unambiguous communication between developer and end-user.

5. Verify quality:

- Quality of the software is maintained by its frequent testing.

- Testing is done to remove defects at early stages, thus reducing the cost at later stages. In particular, high risk areas are tested more thoroughly.
- The software released at the end of every iteration is tested and verified.
- Test cases are created based on use cases (and its scenario).
- Decisions are made on real test results.

6. Control changes:

- Any changes to requirements must be managed and their effect on software should be tracked.
- All change control goes through the convener of the CCB (Change Control Board).
- Members of CCB can be representatives from different areas, say: test designer, project manager, system analyst or stakeholders.

Review Questions

Q.1 Define the software process.

Q.2 List out the Software Process Framework Activities.

Q.3 List out the Software Process Umbrella Activities.

Q. 4 List out the different life cycle stages and what happens in those stages.

Q. 5 Describe the four phase in the Life cycle

Q.6 Explain generic view of process.

Q. 7 Explain PSP and TSP.

Q.8 State the six best practices followed in the life cycle stages.

5

ARTIFACTS OF THE PROCESS

Unit Structure

5.1 The Artifactsets

5.2 ManagementArtifacts

5.3 Engineering Artifacts

5.4 Pragmati Cartifacts

Syllabus:

Artifacts of the process: The artifact sets, Management artifacts, Engineering artifacts, programmatic artifacts.

5.5 THE ARTIFACTSETS

To manage the complete development process of a software system, there is a need to collect and organize the distinct sets of information. These organized distinct sets are known as artifact sets. Artifacts involve the cohesive information that is generally developed and reviewed as a single entity.

The information used in our Life-cycle model that we have studied till now can be organized in to sets called as software artifacts. It can be organized into five distinct artifact sets that are partitioned as follows:

1. **Management set** (ad hoc textualformats),
2. **Requirements set** (organized text and models describing theproblem),
3. **Design set** (models of the solutionspace),
4. **Implementation set** (programming and associated source files),and
5. **Deployment set** (machine-process able languages and associatedfiles).

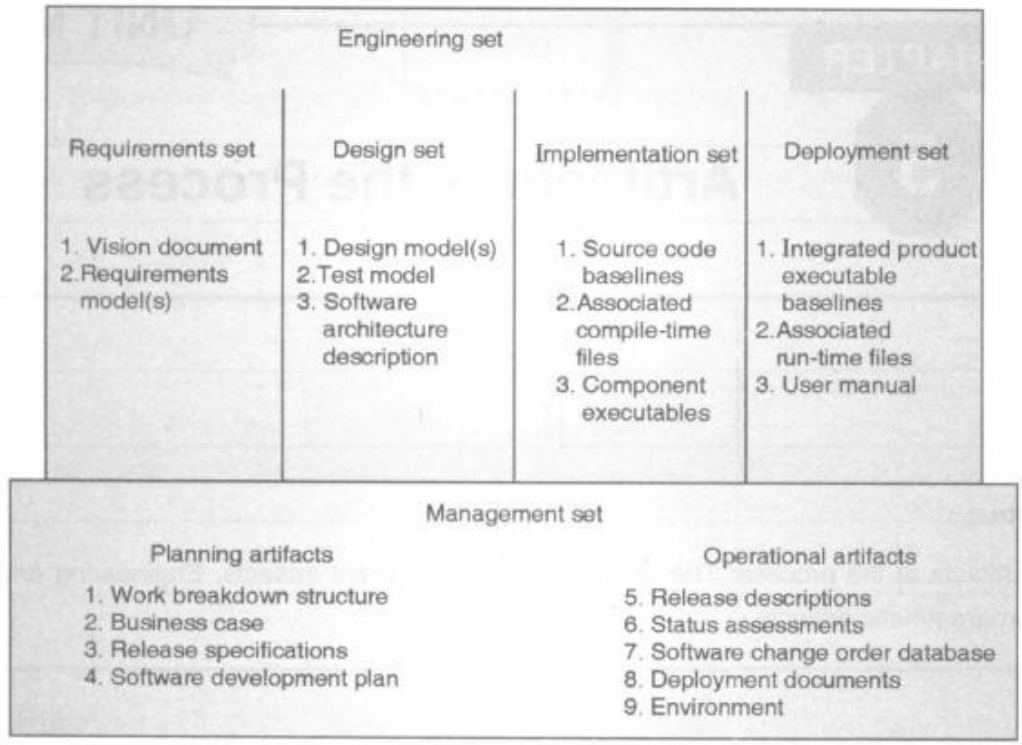


Fig. 5.1.1: Five Distinct Artifact Sets

5.5.1 Management Set

- The„managementset"involvestheartifacts(informationset)associatedwith processplanning andexecution.
- These artifacts use ad hoc notations, text, and graphics, or any other type of representations that are required to attain the "contracts" such as that of:

o Among the project development team (such as the project managers, architects, developers, testers, marketers, andadministrators),

O Among the stake holders (such as financing authority, users, software project manager, organization manager, regulatory agency), and

o Between the project development team and stakeholders.

- Artifactsdescribedinthis„managementset"canalsoincludetheworkbreakd ownstructurefortracking :

o The activities andschedule

The financial mechanisms such as expenditures and profit expectations, The release specifications such as project scope, plan and objectives

The software development plan such as tracking of project process at any instance,

- The status assessments that may include the snapshots of project progress at anyinstance.
- The software change orders i.e. tracking the versions of changed software modules.
- The deployment documents such as the cutover plan and the trainingcourse
- The hardware and software environment that include software tools,process automation, and documentation.

- Artifacts of the „Management set" are evaluated, assessed,analyzedand measuredbasedon the:
- Stakeholderreview
- Changes analyzed between the current version of the artifact corresponding to its previous versions
- Major milestone of the balance described among all artifacts and specially the accuracy of the business case and visionartifacts.

5.5.2 Engineering Set

The „engineering set" includes four types of sets :

1. Requirements set,
2. Design set,
3. Implementation setand
4. Deployment set.

1. Requirements Set:

"Requirement set" artifacts are evaluated, assessed, analyzed and measured based on the :

- Consistency of the release specifications of the "managementset"
- Consistency between the vision and the requirementsmodels
- Consistency, completeness and the semantic balance between information in the "design", "implementation," and „deployment sets" derived from the mappings against these sets.
- Changes analyzed between the current version of the artifact corresponding to its previous versions (scrap, rework, and defectlimitations)
- Review of other qualityfactors

2. Design Set:

- The models in the „design set" are engineered using the UML notations. These design models are essential for achieving the solution.
- The "designset" consists of varied level so fabstraction that represent the components of the product solution such as the „Class diagram" represents the components identities, attributes, static relationships, dynamic interactions andetc.
- "Design set" artifacts are evaluated,assessed,analyzed and measured based onthe: The internal consistency and quality of the designmodel

o The consistency with the requirementsmodels

o Translation of artifacts from design set into the implementation and deploymentsets

o The consistency, completeness and the semantic balance between information in thesesets

o Changes analyzed between the current version of the artifacts in design model corresponding to its previous versions (scrap, rework, and defectlimitations)

o Review of other qualityfactors

3. ImplementationSet:

- The implementationset contains the source code i.e. the programming languagenotations that describe the implementation pattern of the components such as their interface or the dependency relationships andetc.
- "Implementation set" artifacts are written in human-readable formatswhich are evaluated, assessed, analyzed and measured based on the:

o Consistency with the designmodels

o Translation of artifacts from implementation set into the deployment set notations such as compilation and linking so as to evaluate the consistency and completeness among artifactsets,

o Executable files assessed against the relevant evaluation criteria. This assessment is done through inspection, analysis, demonstration, ortesting,

o Execution of test cases that compare expected results with the actualoutcome.

o changes analyzed between the current version of the artifacts in design model corresponding to its previous versions (scrap, rework, and defect eliminations)

o Review of other qualityfactors.

4. Deployment Set:

- The “deployment set” contains:

o Userdeliverables

o Machine languagenotations

o Executable softwareproduct

o Buildscripts

o Installation scriptsand

o Executable target specific data that is required for using the product in its targetenvironment.

- Deployment sets are evaluated, assessed, analyzed and measured based onthe:

o Tests conducted against the user requirements and quality attributes so as to evaluate the consistency, completeness and the semantic balance between the artifacts in the twosets,

o Tests conducted on the partitioning, replication, and allocation strategies described in the mapping components of the „Implementation setagainst the physical resourcesof the„deployment system" such as that of theplatformtype, number, and network topology,

o changes analyzed between the current version of the artifacts in design model corresponding to its previous versions (scrap, rework, and defect eliminations)

o Review of other qualityfactors

Each artifact set described above has a prime development focus matching with any one of the phases of the life cycle.

- Requirements set is the focus of the inceptionphase;
- Design set is the focus of the elaboration phase;
- Implementation set is the focus of the construction phase;and
- Deployment set is the focus of the transitionphase.
- The other set i.e., the „Management set' takes a check on the balance roles. This set also evolves across the life cycle but at a fairly constantlevel.

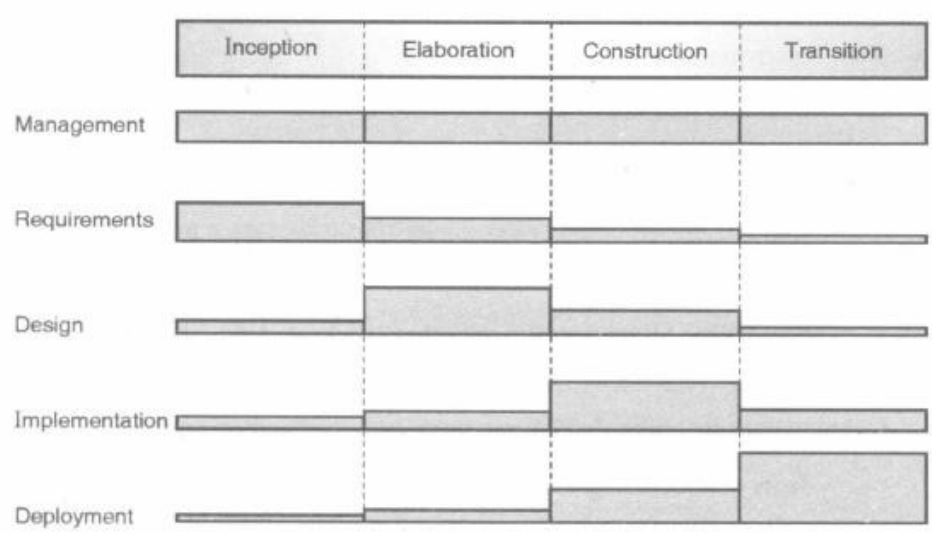


Fig. 5.1.2

Mostly, almost all the software development tools map closely to one of the five artifact sets.

2. **Management set:** scheduling, workflow, error tracking, version control, documenting, deriving spreadsheets, resource management and presentation tools
2. **Requirements set:** requirements management tools
3. **Design set:** design modeling tools
4. **Implementation set:** compilers, debuggers, code analyzers, test coverage analysis tools, and test management tools
5. **Deployment set:** test coverage analysis and test automation tools, network management tools, commercial components (like the OS, GUI, RDBMS, Networks, Middleware), and the installation tools.

Advantages of Deployment Set over other sets (Deployment set v/s other sets):

This differentiation is important because the structure of information delivered to the user is very different from the structure of the source r code information.

- The implementation set consists of the source code whereas the deployment set consists of executable code.
- The quality achieved in the deployment set is really not that attained in the design and implementation sets because the deployment set consists of:

o Dynamically reconfigurable parameters include any type of run-time parameters, buffer sizes, color palettes, number of servers, number of simultaneous clients, and databasefiles.

- o Impact of compiler and linker optimizations includes space optimization versus speed optimization.
- o Performance issues in case of any type of allocation strategies such as centralized v/s distributed, primary v/s shadow threads, dynamic load balancing, hot backup v/s checkpoint/rollback)
- o Virtual machine constraints such as file descriptors, garbage collection, heap size, maximum record size, and disk filerotations,
- o Issues in Process-level concurrency such as the deadlock and raceconditions,
- o Impact of varied Operating systems or any other platform differences on the performance or behavior of the softwareproduct.

5.5.3 Artifact Evolution over the Life Cycle Phases

Each phase of development life cycle focuses on a particular artifact set. But at the end of each phase, the overall system state will have covered all the artifact sets as shown in the below Fig.

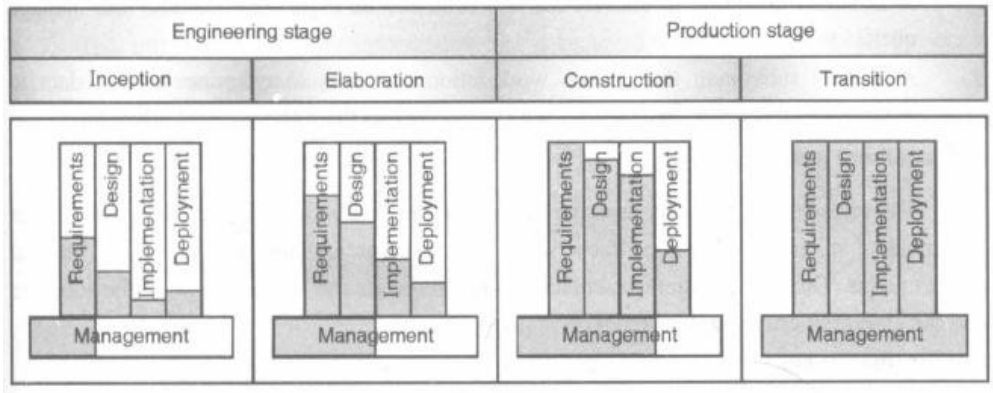


Fig. 5.1.3

- The **inception** phase mostly focusesontheuser' srequirements with a secondary preference oninitial deployment view.
- The **elaboration phase** goes in greater depth in requirements, much more depth in the design set, and further works on implementation and deploymentissues.
- The **construction** phase mostly focuses on design andimplementation.
- The **transition** phase focuses on achieving the consistency and completeness of the deployment set in context of othersets.

5.5.4 Test Artifacts

- The test artifacts are developed simultaneously along with the product since from the inception phase through the deployment phase. That

means, testing is a complete-lifecycle activity that is conducted ineachande very phase of the life cycle. That means, testingisstartedearly"inthe development life cycle and it is NOT a late life-cycleactivity".

- The test artifacts are communicated and developed within the same artifact sets as that of the developed product.
- The test artifacts are implemented in programmable and repeatableformats.
- The test artifacts are documented in a similar manner as any software product isdocumented.
- The test artifact developers/engineers use same tools, techniques, and training as that of the software developers/engineers use while developing theproduct.
- „Test artifact sets" are veryproject-specific.

Example :

Consider a software system which performs seismic data processing for the purpose of oil exploration. This project has three fundamental subcomponents:

1. A sensor to capture raw seismic data in realtime
2. A technical operation to convert the raw data into an organized database and manage queries to thisdatabase
3. A display subsystem that allows workstation operators to examine seismic data in human- readable form. Such a software system derives the following testartifacts:

Management set:

The release specifications describe the objectives, evaluation criteria, and results of an intermediate milestone. These artifacts work as test plans and test results among internal project teams. The changes (defects, enhancements, requirements" ambiguities) in the software and the entry/exit criteria associated with it are again tested.

Requirements set:

The use cases associated with the system involves the operational concepts of the system, acceptance test cases, expected functionalities of the system and its quality attributes. The requirement set is also called as „Test Artifact"because it is the base for all the assessment activities throughout the lifecycle.

Design set:

It includes the test models for non-deliverable components which are used to test the product. These components include design set artifacts that are a seismic event simulation for attaining realistic sensor data, a „virtual operator" that will support the unattended and afterhours test cases. It also includes the instrumentation suites needed for early demonstration of resource usage, the response time, test case drivers for the whole system and for the standalone components.

Implementation set:

This artifact set includes the source code representations useful for testing the components and also the test drivers provide the equivalent test procedures and test scripts. This source code may also include human-readable data files representing explicit test source files. The output files of these test drivers provide the equivalent test reports.

Deployment set:

This artifact set provides the executable versions of test components, test drivers, and datafiles.

5.6 ManagementArtifacts

- The management set consists of various artifacts that describe the intermediate results and the associated information that is required:
- To document the product/process legacy, To maintain the product,
- To improve the product, and To improve the process.
- Business Case Artifact:
- This artifact gives all the information that is required to decide whether the project is worth investing in ornot.
- It describes the expected cost, expected technical and management plans, and backup data necessary to face the risks and realism of theplans.
- The main objective is to transform the vision into economic terms which can help the organization to make an accurate assessment of ROI (Return onInvestment).
- The financial forecasts are updated step by step with more accurate forecasts as the life cycle progresses.

- **(I) Context (domain, market, scope)**
- **(II) Technical approach**
  - (a) Feature set achievement plan
  - (b) Quality achievement plan
  - (c) Engineering trade-off and technical risks
- **(III) Management approach**
  - (a) Schedule and schedule risk assessment
  - (b) Objective measures of success
- **(IV) Evolutionary appendixes**
  - (a) Financial forecast
    - (1) Cost forecast
    - (2) Revenue estimate
    - (3) Bases of estimates

Fig. 5.2.1: Outline of a Business Case Software Development Plan Artifact:

The software development plan (SDP) gives in detail description of the processframework. Two main objectives of a SDP are:

1. **Periodic updatingand**
2. **Understanding and acceptance by managers andpractitioners.**

- **(I) Context (scope, objectives)**
- **(II) Software development process**
  - (a) Project primitives
    - (1) Life-cycle phases
    - (2) Artifacts
    - (3) Workflows
    - (4) Checkpoints
  - (b) Major milestone scope and content
  - (c) Process improvement procedures
- **(III) Software engineering environment**
  - (a) Process automation (hardware and software resource configuration)
  - (b) Resource allocation procedures (sharing across organizations, security access)
- **(IV) Software change management**
  - (a) Configuration control board plan and procedures
  - (b) Software change order definitions and procedures
  - (c) Configuration baseline definitions and procedures
- **(v) Software assessment**
  - (a) Metrics collection and reporting procedures.
  - (b) Risk management procedures (risk identification, tracking, and resolution)
  - (c) Status assessment plant
  - (d) Acceptance test plan
- **(vi) Standards and procedures**
  - (a) Standards and procedures for technical artifacts
- **(vii) Evolutionary appendixes**
  - (a) Minor milestone scope and content
  - (b) Human resource (organization, staffing plan, training plan)

Fig. 5.2.2 : Outline of a Software Development Plan

Work Breakdown Structure :

Work breakdown structure (WBS) is a base for budgeting and cost estimation used to monitor and control the project's financial performance. The project manager must have an insight into project costs and expenditure. The WBS is a serious project planning constraint.

Software Change Order Database :

- Managing and tracking the changes is one of the important activities in an iterative development process.
- As the iterative development process provides a „greater change freedom", a project can be iterated again and again to achieve more and more productivity.
- This flexibility of making greater changes in the project ultimately increases the number of iterations and thus increases the content and also the quality of the software product and that too within a given schedule.
- The flexibility of making changes is achieved in practice through automation and using the today's iterative development environments which involve a phase for „change management". Because without automation, the organizational processes that depend on manual change management techniques face lot of major inefficiencies.

Release Specifications :

- This artifact includes the scope, plan, and objective evaluation criteria for each release and all these details are derived from the vision statement as well as from many other sources such as from the analysis, risk management concerns, architectural considerations, implementation constraints, and quality thresholds.
- All these artifacts of the Release specification evolve and get updated along with the process, thus, achieving greater fidelity and maturity in understanding the requirements.

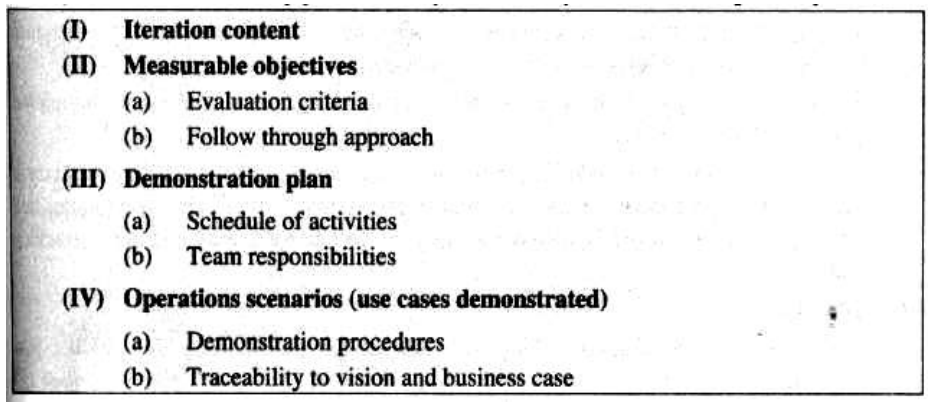


Fig. 5.2.3 : Outline of a Release Specification Release Descriptions :

- It describes the results of each release that includes the performance details against each of the evaluation criteria in the corresponding release specification.
- Release descriptions also describe the evaluation criteria for the configuration baseline and also provide substantiation i.e. a thorough demonstration, testing, inspection, and analysis of each criterion that has been addressed asrequired.

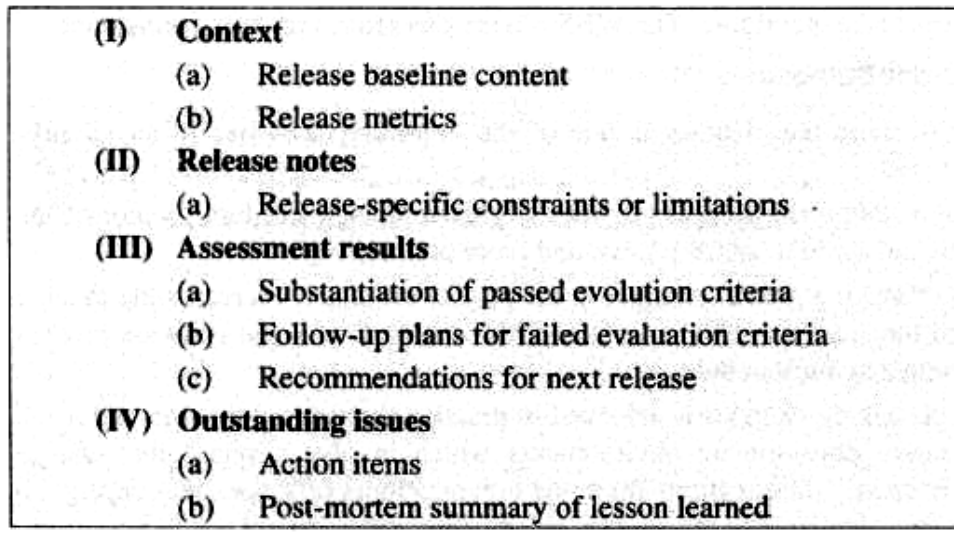


Fig. 5.2.4 : Outline of a Release Description

Status Assessments :

- These artifacts provide periodic snapshots of the project status that includes the project manager's risk assessment, and the quality and management indicators.
- It also includes a review of resources, staffing, financial data (cost and revenue), top ten risks, technical progress, major milestone plans and results, and scope of the project.

Environment :

- Mostly, all the modem approaches define the development and maintenance environment as a first-class artifact of the process.
- A development environment is said to be robust and integrated if it supports automation of the development process.
- Such an environment includes requirements management, visual modeling, document automation, programming tools, automated regression testing, and continuous and integrated change and configuration management, and also a feature and defect tracking tool.

Deployment:

- This artifact includes several document subsets for transitioning the product into operational status.

- If the system is delivered to a separate maintenance organization, then in such cases, the deployment artifacts must include the computer system operations" manuals, software installation manuals, plans and procedures, site surveys, andetc.
- If the software product is used for commercial purpose, the deployment artifacts must include marketing plans, sales rollout kits, and training courses.

Management of Artifact Sequences :

Each phase of the life cycle produces new artifacts and also updates the previously developed artifacts to incorporate the changes done and describes the further depth and breadth of the solution.

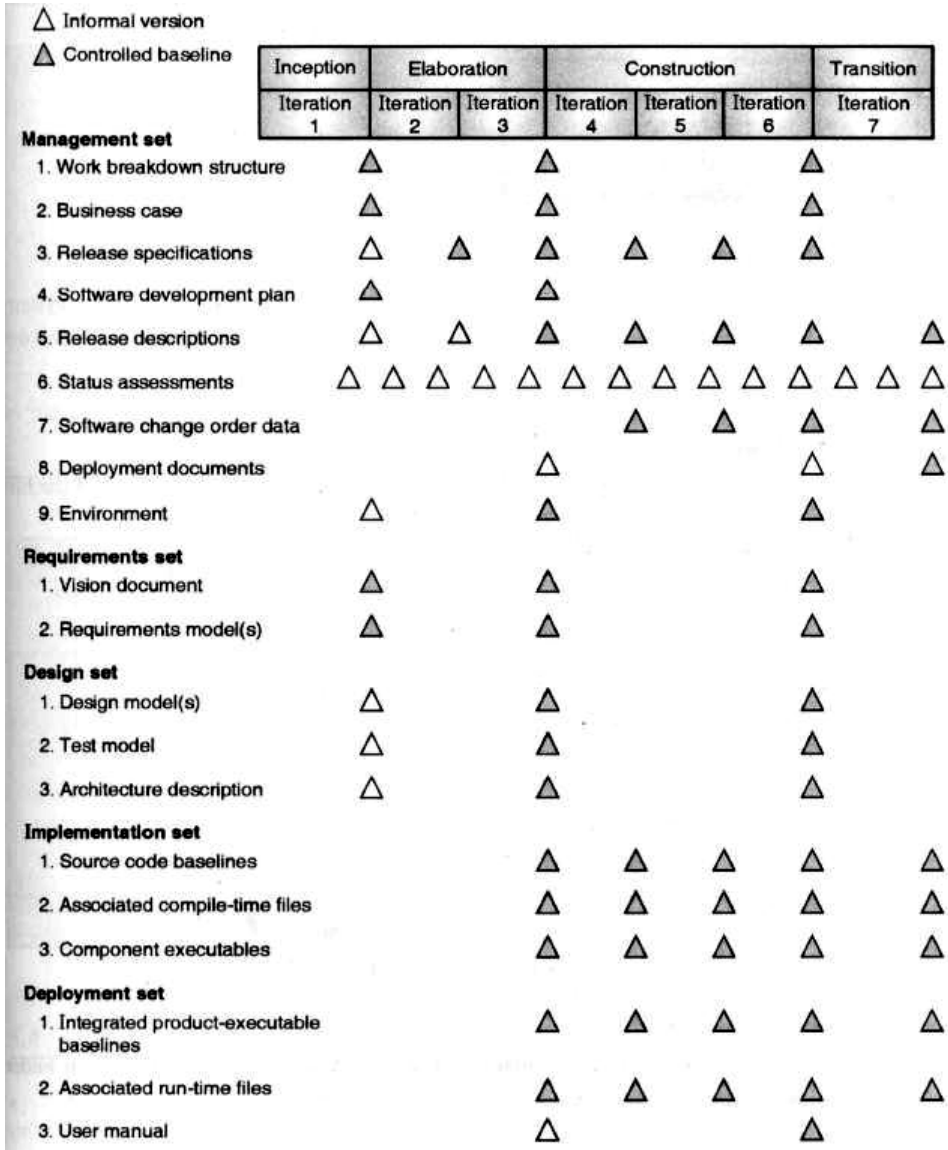


Fig. 5.2.5 : Artifact Sequences across a typical life cycle

5.7 ENGINEERINGARTIFACTS

This set of artifacts is derived from the thorough engineering notations such as of UML notations, programming languages, or executable machine codes. This Engineering artifact set can be categorized into three kinds which are specially intended to give a more general review.

These are :

1. Vision document
2. Architectural description
3. Software user manual

1. Vision Document:

• This document provides a complete vision about the proposed software system that is under development and. It is the base that supports the contract between the funding authority and the development organization.

• This document involves changes progressively as understanding evolves out of the requirements, architecture, plans, and technology. A good vision document should change slowly at each and every phase of the lifecycle.

(I) Feature set description
- (a) Precedence and priority

(II) Quality attributes and ranges

(III) Required constraints
- (a) External interfaces

(IV) Evolutionary appendixes
- (a) Use cases
  - (1) Primary scenarios
  - (2) Acceptance criteria and tolerances
- (b) Desired freedoms (potential change scenarios)

Fig. 5.3.1 : Outline of a vision Document

2. ArchitectureDescription

• This artifact provides an organized architectural view of the software that is under development.

• These descriptions are derived from the design model which includes the views of the design, implementation, and deployment sets that

helps in understanding how the operational concept of the requirements set are derived.

- The depth and breadth of the architecture description varies from project to project based upon various factors.

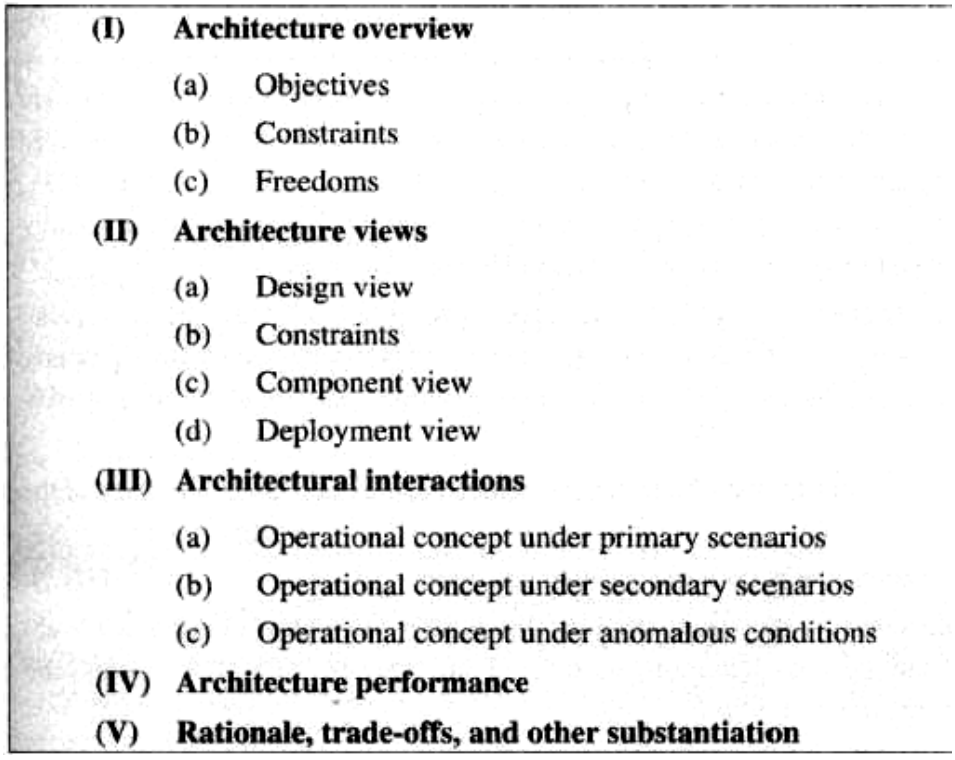


Fig. 5.3.2 : Outline of a Architecture Description

3. Software UserManual

- The manual provides the user with the references that are required to support the delivered software.
- The user manual includes the installation procedures, usage guidance, operational constraints, and a user interface details.
- This manual must be developed in early phases of the life cycle since it is a required mechanism for communicating and stabilizing the user requirements.
- The user manual is written by the members of the test team who understand the user's perspective more clearly than the development teamdocs.

5.8 PRAGMATICARTIFACTS

- People are interested in reviewing the Artifacts but they don't understand the language of the artifact because they do not have any knowledge of the engineering language in which the artifact is written and they don’t even care to learn it. It is very common among the software managers, quality assurance specialists, or an auditing

authority who say "I do know much about the UML notations and nor I'm going to learn UML, but I can review the design models of this software, so give me the software description in the form of flowcharts or text so that I can understand."

- People are interested in reviewing the artifacts but don't have any access to the tools because it is very of ten seen that the development organization are not fully tooledandthes take holders (other than the development organization) rarely have any capability to review the engineering artifacts on-line. Ultimately, the development organizations are forced to exchange the paper documents instead of on-line documents. Standardized formats constructed using UML notations, spread-sheets, Visual Basic, C++, visualization tools, and using the web designing are rapidly becoming economically feasible for all stakeholders to exchange information electronically.

- Human-readable engineering artifacts use rigorous notations that are complete, consistent, and used in a self-documenting manner. It is constructed using properly spelled English words so as to allow easy identification, easy readability and easy understanding. Acronyms and abbreviations must be used only where they are acceptable in place of the component'susage.

- A good documentation is self-defining and that getsused.

- Paper is tangible whereas, electronic artifacts are so easy to change. On-line and Web-based artifacts can be changed more easily and are viewed with more skepticism because of their inherent volatility.

Review Questions

Q . 1 Give an overview of the artifact set.

Q . 2 Write short notes on :

(a) Management artifacts

(b) Engineering artifacts

(c) Pragmatic artifacts

Q. 3 Write short notes on Test Artifacts

Q. 4 List out the different types of management artifacts and give their brief outline.

Q. 5 List out the different types of engineering artifacts and give their brief outline.

6

MODEL BASED SOFTWARE ARCHITECTURE

Unit Structure

6.1 Introduction

6.2 Architecture: Managementperspective

6.3 Architecture: Technical Perspective

Syllabus :

A Management perspective and Technical perspective.

6.1 INTRODUCTION

Definition of Software Architecture: If we consider the complex software system which shows the overall problem by central design is known as Software architecture. It has many dimensions in case of complexities. We can measure or prove that by any logical relation of physics or mathematics.

It is note that without depends on any prove theory software architecture based on the experimentation. We can say this is fundamental reason of the transitioning to an iterative process. Old conventional software process produces the less powerful systems. Architecture comes First are includes

- Simpler
- Requires informal representations
- Requires single computers
- Requires single program system
- Various mapped objects
- Objects implementation

Model: Independent abstraction of a system is known as **model.**

View : Model which abstract specific, relevant perspective is known as **view.**

6.2 ARCHITECTURE: MANAGEMENTPERSPECTIVE

Architecture is most critical and technical product of the software project, which includes parameters infrastructure, control and data interface. All these parameters coordinates each other and makes huge system. If there

are lots of languages available in the communication media and management team members having different literacy then lots of communication problems are generated which are never solved. If the team members have good in inter project communication then and then software architecture must be accurate and exact.

There are three different aspect of Management Perspective;

1. Architecture

- It is design of the software system
- Intangible concept
- All types of engineering which is useful to fulfill bill of material is part of it.
- Problems of whatever purchase and sale of the components are solved since custom components are elaborate. Cost of unit component and it's as sembling and construction cost is to be determined.

2. Baseline of Architecture

- It is tangible artifacts
- They are satisfying the all stakeholders
- All stakeholders are the primary vision which includes the function and quality
- The vision set the business parameters which includes o Cost

o Profits

o Time

o Technology

o Peoples

3. Description of Architecture

- Well manner subset of information
- It includes the text and graphics which gives the complete information about the model
- It communicates about the intangible information for tangible artifacts.

If system changes then the architecture should be change. The number of views and its level should be change. For example of raw boat and jet boat, both are boats but is run through human interface and another is automatic.

Fig. 6.2.1: Architecture of raw boat and jet boat Importance of Software Architecture

1. Get stable software architecture, which is fixes the milestone. By using those milestones problems like purchasing and selling are resolved.
2. Gets basis of the trades off balancing between problem and its solution space.
3. Encapsulates the communications between individual, various teams, stakeholders and organizations.
4. Project unsuccessful or failure reasons : Dull Architecture and immature process
5. Requirements are understandable in the mature process and architecture should be previously demonstrated.
6. Two important facts 1) Architecture Development and 2) Process Definition

6.3 ARCHITECTURE: TECHNICAL PERSPECTIVE

We look towards Software via various different aspects which are stated below;

- Structure of software system
- Behavior
- Guides lines patterns about elements
- Collaboration
- Compositions

The all over information given in design model and architecture view is abstraction of design model. Real system involves the four views of the design model;

Table 6.3.1: Design model"s four view

Design	Express structure and function of the design model. Important for every system
Process	Express concurrency and control thread between other views Add as per complexity
Components	Express implementation set structure Add as per complexity
Deployment	Express structure of deployment set Add as per complexity

Following figure illustrates artifacts of the design set which includes above various views and architecture description (captures electronically and maintains on printed format). Engineering models and views are defined with collection of the Unified Modelling Language diagrams such as use case diagram which describes how system critical.

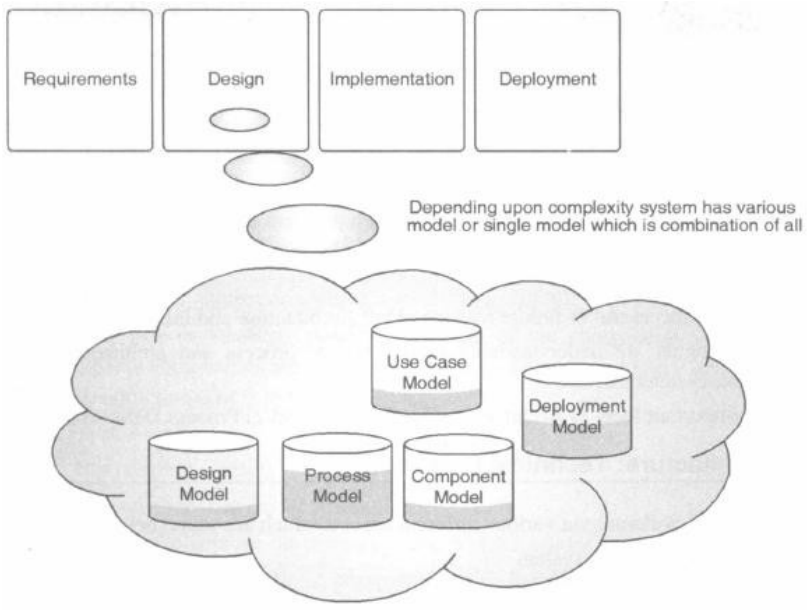


Fig. 6.3.1 : Architecture, organized and abstracted view into design model

Review Questions

Q. 1 What is Software architecture ?

Q. 2 Explain three aspects of management perspective.

Q. 3 Why software architecture is important ?

Q. 4 Explain Software architecture in technical perspective.

❖❖❖❖

Unit III

7

PROCESS WORKFLOWS

Unit structure

7.0 OBJECTIVES

After reviewing this unit, you will be able to summarize the project management workflow with new techniques along with the objectives to be achieved during the process.

7.1 SOFTWARE PROCESS WORKFLOWS

Before moving on to software process workflows in software project management, let's first clarify these two terms: workflow and software process workflows.

1. **Workflow -**
   In general, workflow refers to the series of sequential activities that are performed to achieve a certain goal. Each step of the workflow is defined by three parameters, namely input, transformation, and output. In the workflow process, a series of actions are performed to achieve a business result.

2. **Software Process Workflows -**

   The software process is the set of related activities that are carried out to obtain a software product as a result and there the workflows of the software process guide software developments in a linear fashion by performing a series of sequential activities.

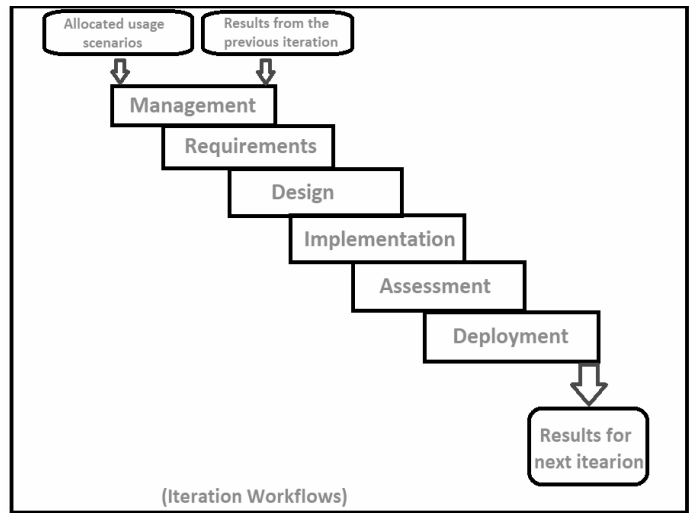


Fig7.1 - Iteration workflow

There are 7 main software process workflows in software project management:

1. Management workflow: Some of the essential steps to control the process are performed in the management workflow. Artifacts include the software development plan (SDP), business case, vision, etc., ensuring mutually beneficial conditions for stakeholders in terms of software project development, execution and implementation.

2. Environment workflow: Automate the process to coordinate and integrate tools and people with the process through the workflow, which in terms of reducing human errors and enabling faster development with faster resource allocation and problem response. Evolution of the maintenance environment to maintain and update the software.

3. Requirements workflow: Analyze the problem space to identify / understand problems and find a solution. Evolution of requirements artifacts, such as use cases, requirements, and design documents / specifications that help describe the function, architecture and design of the software.

4. Design workflow: Software modeling is done to express software design, where software modeling will take care of the entire software design. Evolution of architectural and design artefacts.

5. Implementation workflow - In this workflow, the implementation of designs and architectures is done by programming the components. Artifacts are developed along with this plant and distribution.

6. Evaluation workflow: Evaluate the trends of the process. Product quality assessment is performed here by analyzing product quality attributes and managing product defects.

7. Implementation workflow: In this workflow, the process of delivering the end products to the user is performed or the application / software product is prepared to run and operate in a specific environment.

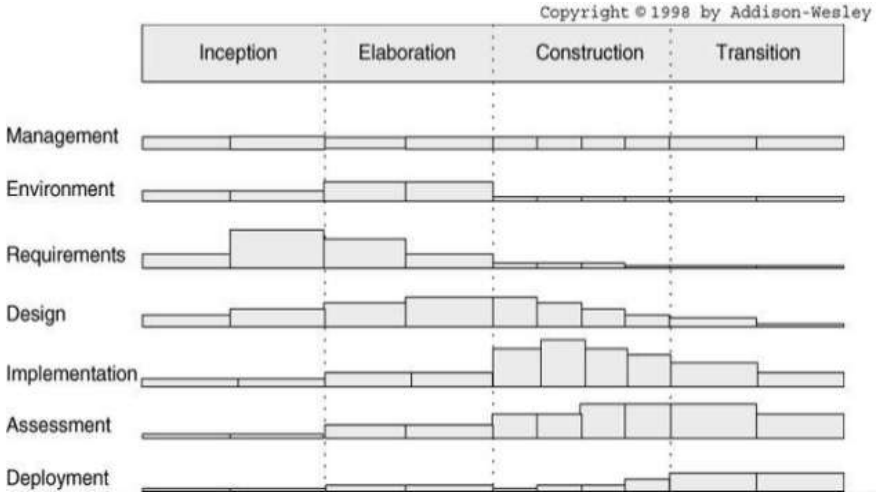


Fig 7.2 Levels of activity in the phases of the life cycle

7.2 KEY PRINCIPLES OF MODERN SOFTWARE ENGINEERING

There are some modern principles for software development. By following these modern principles we can develop effective software that meets all customer needs. To develop the right software, you need to follow the following 10 software development principles:

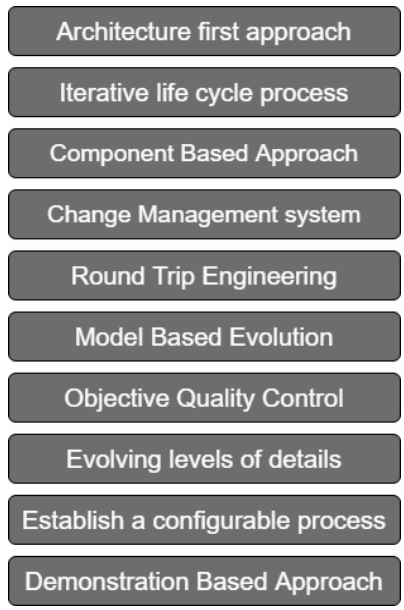


Fig 7.2 Approaches to software development

These are explained below.

1. **First approach to architecture:** In this approach, the main goal is to build a solid architecture for our software. All ambiguities and flaws are spotted during the very mundane phase. Furthermore, we can make all decisions regarding software design that will improve the productivity of our software.

2. **Life cycle iterative process:** In an iterative life cycle process, we repeat the process over and over to eliminate risk factors. In an iterative lifecycle, we have mainly four steps: requirements gathering, design, implementation, and testing. All of these steps are repeated over and over until the risk factor is mitigated. The iterative life cycle process is important to alleviate the risk at an early stage by repeating the above-mentioned steps over and over again.

3. **Component-based approach:** The component-based approach is a widely used and successful approach where we reuse previously defined functions for software development. We reuse the part of the code in the form of components. Component-based user interface development optimizes the design process and requirements and is therefore one of the important principles of modern software.

4. **Change management system:** Change management is the process responsible for managing all changes. The main goal of change management is to improve the quality of the software by making the necessary changes. All implemented changes are then tested and certified.

5. **Round trip engineering:** In roundtrip design, code generation and reverse engineering occur simultaneously in a dynamic environment. Both components are integrated so that developers can easily work on both. In round-trip engineering, the main character is the automatic updating of artifacts.

6. **Model-based evolution:** Model-based evolution is an important principle of software development. A model-based approach supports the evolution of graphics and textual notions.

7. **Objective quality control:** The goal of quality control is to improve the quality of our software. It includes a quality management plan, quality metrics, quality checklist, quality baseline, and quality improvement measures.

8. **Evolution of the levels of detail:** Schedule interim releases in use case groups with changing levels of detail. We need to plan for an incremental deployment where we have an evolving level of use cases, architecture and details.

9. **Establish a configurable process:** Establish a configurable process that is economically scalable. A single process is not suitable for all development, so we need to use a configurable process that can handle multiple applications.

10. **Proof-based approach:** In this approach, we mainly focus on demonstration. It helps to increase the productivity and quality of our software by providing a clear description of the problem domain, the approaches used and the solution.

7.3 ITERATION WORKFLOW

1. **Planning of the iterations** it is generally a process to be adapted as the project develops through changes to plans. Plans are simply modified based on feedback from the monitoring process, some changes in project assumptions, risks, and changes in scope, budget or schedule. It is very important to include the team in the planning process. Basically, planning is generally concerned with explaining and defining the actual sequence of interim results. It is an event where each of the team members identifies what part of the team's pending work they can commit to doing during a subsequent iteration.

2. Iteration planning is generally the process of discussing and planning the next cycle, stage, or iteration of a software application under development. An evolutionarily developed plan is very essential because there are always adjustments to developed content and planning as a first assumption that simply evolves under well understood project circumstances.

To achieve economies of scale and greater returns on investment, we need to move to a software production process driven by technology improvements in process automation and component-based development. Two stages of the life cycle are:

1. The design phase, led by less predictable but smaller teams that conduct design and synthesis activities

2. The production phase, led by more predictable but larger teams conducting construction, testing and distribution.

The transition between engineering and manufacturing is a crucial event for the various stakeholders. The production plan has been agreed and the understanding of the problem and solution is good enough to allow all interested parties to make a firm commitment to move production forward. The engineering phase is divided into two distinct phases, start up and development, and the production phase in construction and transition. These four stages of the life cycle process are loosely mapped to the conceptual structure of the spiral model:

1. **INITIAL PHASE:** The primary objective of the initial phase is to reach agreement among stakeholders on the objectives of the project life cycle.

➢ **MAIN OBJECTIVES**

★Management of software projects

- Establish project software scope and boundary conditions, including an operational concept, acceptance criteria, and a clear understanding of what is and is not intended to be in the product

- Discriminate critical system use cases and key operational scenarios that will lead to major design tradeoffs.

- Demonstrate at least one candidate architecture against some of the main scenarios
- Estimated cost and schedule for the entire project (including detailed estimates for the development phase) Estimation of potential risks (sources of unpredictability)

➢ **ESSENTIAL ACTIVITIES**

- Formulation of the scope of the project. The information repository must be sufficient to define the problem space and derive acceptance criteria for the final product.
- Synthesizing the architecture. Sufficient information repository is created to demonstrate the feasibility of at least one candidate architecture and an initial baseline of manufacturing / purchasing decisions so that cost, time and resource estimates can be derived.
- Planning and preparation of a business case. Alternatives are evaluated for risk management, personnel, iteration plans, and cost / planning / profitability trade-offs.

➢ **PRIMARY EVALUATION CRITERIA**

- Do all stakeholders agree with the scope definition, cost estimates and timelines?
- Are the requirements understood, how does it demonstrate the fidelity of critical use cases?
- Are the cost estimates and timelines, priorities, risks and development processes credible?
- Does the depth and breadth of an architectural prototype demonstrate the above criteria? (The main value of prototyping the candidate architecture is to provide a vehicle for understanding the scope and assessing the credibility of the development group in solving the particular technical problem.)
- Are actual resource expenditures acceptable compared to planned expenditures?

2. **PREPARATION PHASE:** At the end of this phase, the "engineering" is considered complete. The development phase activities should ensure that the architecture, requirements and plans are sufficiently stable and that the risks are sufficiently mitigated so that the cost and timing for completion of the development can be predicted within an acceptable range. During the build phase, an executable architecture prototype is created in one or more iterations, depending on scope, size, and risk.

➢ **MAIN OBJECTIVES**

- Base the architecture as fast as possible (create a configuration-managed snapshot where all changes are simplified, tracked and maintained)
- Basing the vision
- Establish a high-fidelity plan for the construction phase Demonstrate that the core architecture will support the vision at a reasonable cost in a reasonable time

➢ **ESSENTIAL ACTIVITIES**

- Create the vision.
- Management of software projects
- Process and infrastructure development.
- Prepare the architecture and select the components.

➢ **PRIMARY EVALUATION CRITERIA**

- Is the vision stable?
- Is the architecture stable?
- Does the executable demonstration show that major risk elements have been credibly addressed and resolved?
- Is the construction phase plan sufficiently faithful and supported by a credible estimate basis?
- Do all stakeholders agree that the current vision can be fulfilled if the current development plan of the whole system is executed in the context of the current architecture?
- Are actual resource expenditures acceptable compared to planned expenditures?

3. **CONSTRUCTION PHASE:** During the construction phase, all remaining components and functionality of the application are integrated into the application and all functionality is thoroughly tested. The newly developed software integrates where needed. The construction phase represents a production process, in which emphasis is placed on resource management and control of operations to optimize costs, time and quality.

➢ **MAIN OBJECTIVES**

- Minimize development costs by optimizing resources and avoiding waste and unnecessary rework
- Get the right quality as fast as possible

- Get useful versions (alpha, beta and other trial versions) as fast as possible

➢ **ESSENTIAL ACTIVITIES**

- Management, control and optimization of resource processes
- Complete the development and testing of components according to the evaluation criteria.
- Evaluation of the product launch against the vision acceptance criteria.

➢ **PRIMARY EVALUATION CRITERIA**

- Is this product baseline mature enough to be deployed to the user community? (Existing defects are not an obstacle to achieving the purpose of the next version.)
- Is the baseline for this product stable enough to be distributed to the user community? (The pending changes are not an obstacle to achieving the purpose of the next version.)
- Are stakeholders ready to move to the user community?
- Are actual resource expenditures acceptable compared to planned expenditures?

4. **TRANSITION PHASE:** The transition phase is initiated when a baseline is mature enough to be implemented in the end user's domain. Typically this requires that a usable subset of the system with acceptable quality levels and user documentation has been achieved for the switch to the user to produce positive results. This phase could include managing the software project one of the following activities:

   1. Beta testing to validate the new system against user expectations
   2. Beta testing and parallel operation in relation to a legacy system it is replacing.
   3. Conversion of operational databases
   4. Training of users and maintainers The transition phase ends when the baseline of the implementation has reached the complete vision.

➢ **MAIN OBJECTIVES**

- Achieve user self-sufficiency
- Reach stakeholder agreement that implementation baselines are complete and consistent with vision assessment criteria.
- Get the baselines of the final product in the fastest and cheapest way that is practical.

➢ **ESSENTIAL ACTIVITIES**

- Synchronization and integration of concurrent build increments across consistent distribution baselines
- Implementation specific engineering (cutting, commercial packaging and manufacturing, development of sales implementation kits, training of staff in the field)
- Evaluation of implementation baselines against the comprehensive view and acceptance criteria in the requirements set.

➢ **EVALUATION CRITERIA**

- Is the user satisfied?
- Are actual resource expenditures acceptable compared to planned expenditures?

7.4 ARTIFACT SET

The artifact is highly associated and related to specific development methods or processes. The methods or processes can be project plans, business cases, or risk assessments. In general, various collections and collections of detailed information are organized and incorporated into artifact sets. A set generally represents a complete aspect of the system. This is done simply to develop and establish a complete software system in a manageable way.

Software lifecycle artifacts are generally organized and divided into two sets, namely the management set and the engineering set. These sets are further subdivided or partitioned based on the underlying language of the set. These artifact sets are shown below in the diagram:

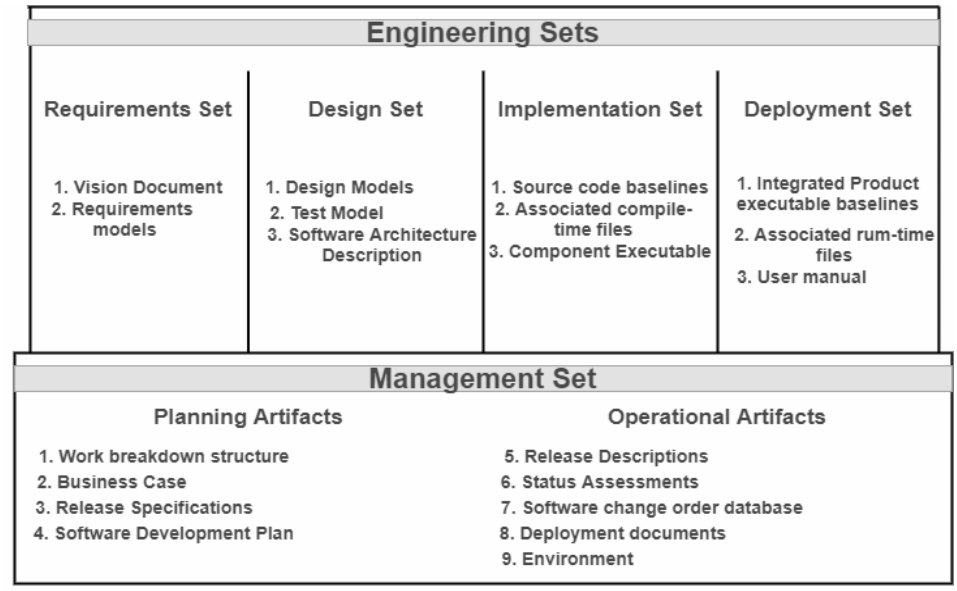


Fig 7.3 Overview of artifact sets

1. Engineering set:

In this set, the main mechanism or method for forming an idea regarding the evolutionary quality of these artifact sets in the transition of information from one set to another. This set is divided into four separate sets which include the requirement set, the design set, the implementation set, and the implementation set.

1. **Requirement set -** This set is the primary design context used simply to evaluate the other three artifact sets in the design set and the basis for test cases. The artifacts in this set are evaluated, verified, and measured using a combination of the following:

- Analysis of the consistency between vision models and current needs.
- Analysis of consistency with the complementary specifications of the management system.
- Consistency analysis between requirements models.

2. **Design Set -** Tools used in visual modeling tools. To design the design model, Unified Modeling Language (UML) notations are used. This set simply contains many different levels of abstractions. The design model typically includes all structural and behavioral data or information to determine the BOM. These assembly artifacts mainly include test models, design models, software architecture descriptions.

3. **Distribution set -** The tools used are debuggers, compilers, code analyzers, test management tools. This set generally contains source code such as the component implementation, its form, interfaces, and executables required for independent component testing.

4. **Development set -** The tools used are network administration tools, test coverage and test automation tools, etc. To simply use the final result or product in the environment where it should be used, this set usually contains executable software, build scripts, ML annotations, installation scripts.

2. Management set:

This set generally acquires artifacts that are simply associated with planning and execution or the running process. These artifacts generally use ad hoc notation. It also includes text, graphics or any representation required or necessary to simply acquire the "contracts" between all project staff (such as project developers, project management, etc.), between different stakeholders (such as the user , the project manager, etc.)), and also between stakeholders and project staff.

This set includes various artifacts such as work breakdown structure, business case, software development plan, distribution, environment. The artifacts in this set are evaluated, verified, and measured using a combination of the following:

- Review of relevant stakeholders.
- Analyze the alterations or changes between the current version of the artifact and previous versions.
- Demonstrations of major milestones relating to the balance between all artifacts and, in particular or specifically, the accuracy of the business case and vision artifacts.

7.6 SUMMARY

1. Most process descriptions use sequences of activities in representation format.
2. Sequence-oriented process descriptions are easy to understand, represent, plan and execute.
3. Software projects are not sequential activities. Software projects include many teams progressing on many artifacts that need to be synchronized, contrasted, homogenized, merged and integrated.
4. The complexity of management arises from the distributed nature of the software process and subordinate workflows.
5. Sequentially from the conventional process model, the main problem arises as the details of each step must be completed and frozen before the next one begins.
6. In this process, important engineering decisions would be slowed down or stopped altogether. The intention here is to explicitly recognize the continuum of activities at all stages.

7.7 BIBLIOGRAPHY

1. Project Management Basic Manual: Mantel Jr., Meredith, Shafer, Sutton with Gopalan (Wiley India Edition)
2. Project Management - TYBSCIT -Semester 6 - Divya Shetty - Sheth Publications.
3. https://www.geeksforgeeks.org/

7.8 EXERCISES AT THE END OF THE CHAPTER

1. Explain the software process workflow with the diagram.
2. What are the key principles of modern software engineering?
3. Explain iteration workflows with diagrams.
4. Explain the artifact sets with their phases.

8

PROCESS CONTROL POINTS

Unit structure

8.0 OBJECTIVES

This chapter will help us understand the process checkpoints required during the planning period. We will also focus on major milestones, minor milestones and periodic status assessment.

8.1 INTRODUCTION

In software development, all system-level events take place at the end of each development phase. These checkpoints provide visibility into milestones in the lifecycle, as well as problems and problems throughout the system. These checkpoints generally provide the following:

- Simply synchronize management and engineering perspectives.
- It also verifies that all the steps have been fulfilled or not.
- It provides a basis for analysis and evaluation in order to determine if a project is progressing as planned and also to make the correct correction and action as required.
- It also identifies essential risks, problems or problems and intolerable conditions.
- Throughout the entire life cycle, it performs a comprehensive assessment.

In general, three sequences of project checkpoints are used to synchronize stakeholder expectations throughout the lifecycle.

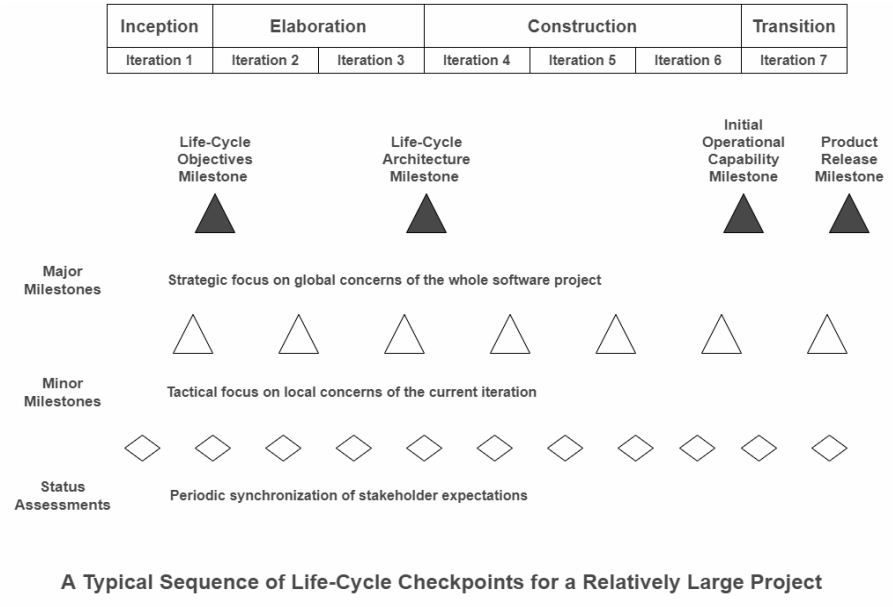


Figure 8.4

These three types of joint management reviews are described below:

1. **Important milestones -** Major milestones are system-wide events that occur at the end of each development stage. These milestones can be used in various process models, including conventional waterfall models. They typically help provide visibility into system-wide problems. They also help synchronize management and engineering perspectives. It helps to verify that the goal or objective of each stage has been successfully achieved or not. They are used to obtain the competition of all those who are interested in the current state of the project. These milestones are very essential to confirm and ensure that understanding of requirements, life cycle plans and product form, function and quality are at balanced levels of detail.

2. **Minor milestones -** Minor milestones are also called micro milestones. They are simply monitoring points that project managers generally use to track activities every day. Minor milestones are sprint-focused events that are held to review the data or content of a sprint in detail and also to authorize work that has been continued. They typically divide the time spent between major stages into short time intervals. This is done to give us confidence in achieving important milestones. Early iterations focus simply on analysis and design, while subsequent iterations focus more on completeness, consistency, usability, and change management.

3. **State Ratings -** State assessments generally provide mechanisms for addressing, communicating, and resolving problems or problems related to project, technical, and management risks. Its main goal is to ensure that all expectations of all parties are synchronized and

consistent. They are carried out to direct and verify the progress and quality indicators, guaranteeing continuous attention to the dynamics of the project. It also maintains communication between all interested parties. It also provides the management with frequent and regular information on the progress made.

8.2 PLANNING AND ANALYSIS OF CONTROL POINTS

For full benefit and impact, checkpoints should be identified based on specific project phases and if necessary to ensure timely advancement of project objectives and results. Checkpoints should be structured to answer one main question: are you ready for the next stage? If the answer is yes, the project continues. If the answer is no, other actions must be taken, to include progress with corrective / compensatory actions, suspension of the project or total cancellation.

ILLUSTRATION OF THE CONTROL POINTS MANAGEMENT PROCESS

To illustrate the use of control points, we can use a project structure organized in five (5) phases, as follows:

- Phase 1: requirements. Define the technical and commercial requirements of the project.
- Phase 2: planning. Design the technical results.
- Phase 3: development. Develop and test the technical solution.
- Phase 4: implementation. Implement and support the implementation of technical results.
- Step 5: closing. To transfer the project and the final results from the project state to the operational state.

Continuing with this illustration, the following checkpoint "decision tree" marks the way through the progression process.

➔ Establishment of checkpoints for "Phase 1: Requirements"

- Have all the "requirements" activities been completed?
- Are there any open problems?
- How will these problems be solved?
- Are the established requirements sufficient to move on to the next stage?
- If not, requirements issues must be resolved, mitigated, or eliminated before you can progress.

➔ Establish checkpoints for "Phase 2: Design"

- Have all the "design" activities been completed?
- Does the design meet the established requirements?
- Are there any open design problems?
- How will these problems be solved?
- Does the layout work as expected?
- Is your project ready to move on to the next stage? If not, design issues must be solved, mitigated, or eliminated before you can progress.

➔ Establishment of checkpoints for "Phase 3: Development"

- Have all "development and test" activities been completed?
- Is the system working as expected?
- Are there any open development problems?
- How will these problems be solved?
- Is the system ready to go to the next step?
- If not, development problems must be solved, mitigated or eliminated before progress can be made.

➔ Establishment of checkpoints for "Phase 4: Implementation"

- Have all "distribution" activities been completed?
- Are there any open problems?
- How will these problems be solved?
- Is the project ready to move on to the next stage? If not, deployment issues must be resolved, mitigated, or eliminated before progress can be made.

➔ Establishment of checkpoints for "Phase 5: Closure"

- Have all "arrest and transition" activities been completed?
- Are there any open problems?
- How will these problems be solved?
- Have all necessary "close and accept" approvals been obtained?
- Has the review of lessons learned been completed?
- Can the project be closed? If not, closure issues must be resolved, mitigated, or eliminated before the project can be closed.

Checkpoints can present difficult options. Each checkpoint analysis requires an objective review of the design to date. This can be a difficult task for the project manager and the team that has invested so much in each project. Sometimes the checkpoints are not passed and unpopular actions must be taken, up to and including the cancellation of the project. But, when the feasibility of the project is in doubt, it is better to abandon than proceed with an impractical initiative. Eventually, checkpoints can provide a much-needed safety net to avoid wasting time and resources.

8.3 INTRODUCTION TO MILESTONES

1. A milestone is an indicator in a project that signifies a change or stage of development. Milestones are powerful components in project management because they show milestones and map forward movement in your project. Project plan.

2. Milestones act as signals throughout the project, helping you stay on the right track. Without keeping track of project milestones, you are just monitoring activities and not necessarily following the right path in your project.

3. Milestones can do more than just show progress - they can help you communicate what's happening with your project. TeamGantt features project milestones in its free project management software, so it syncs seamlessly with all moving parts of the Gantt chart.

➔ Is it a task or a milestone?

You're not building a rocket here, you're building a Project planand the components are not that complex. That said, distinguishing between tasks and milestones can be difficult on larger projects, or if the project you're managing is simply not (yet) within the scope of your experience.

If you've ever been confused about what is (or isn't) a milestone in your project plan, ask yourself these questions:

1. Is it an activity or a final result?

2. Will this affect the final deadline?

3. Is it an important moment in the project that will indicate progress forward?

4. Does this need to be reviewed by interested parties?

5. Is it an event that affects the project?

Essentially, you want to make events more important than your project's milestones so they can be easily viewed and mapped by the project team. Milestones have additional importance to the activities in a plan so that the project manager can keep track of the activities while the team and stakeholders focus on progress forward.

➔ *Examples of project management milestones*

Milestones simplify project monitoring by identifying important events, dates, decisions and results. Below are some examples of project milestones that you can include in your plan:

- Start and end dates of the project phases
- Key deliveries
- Customer and stakeholder approvals
- Important meetings and presentations
- Key dates or outages that may affect planning

Let's delve a little deeper and explore 3 specific examples of how using project milestones can benefit your projects.

Check the deadlines

- No plan is complete without a list of deadlines! The best way to make them visible is to use the technique of milestones and deliverables of project management. What does this mean? Make the final results of the project milestones!
- Why do this? Well, it's no secret that not everyone wants to go through their beautiful project plan carefully to find key dates. Most people, including teammates, want a high-level view of key dates and events. Milestones are great for this purpose because they are specially mentioned, usually with a diamond, in project plans.
- While you should list the activities and efforts that lead to a project milestone, be sure to present the milestone at the end of those activities to indicate a delivery, or even a presentation, of the end result.

Highlight important dates

- Are there days from now until the end of your project that could affect your project in any way? Maybe your team needs to be out of the office for mandatory training. Maybe you are expected to attend a board meeting.
- It is important to keep all these important events in mind when planning a project because they could affect yours the timeline of the project. So why not include them as project milestones so you can keep track of them all in one place?
- In this example, the team's off-site strategic meeting has been added to the project plan as a milestone so that work can be planned around it.

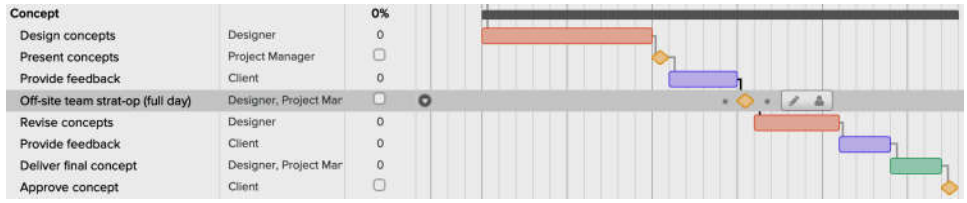


Figure 8.5

Identify potential project bottlenecks

- Many projects rely on the work produced by external teams or partners to move forward. If you're not tracking these external factors somewhere, there's a big chance you'll forget to track.
- This is why it is important to list these deliverables as project milestones if you are working on a project that depends on someone or something outside of your project. Here's an example of what customer approval might look like.

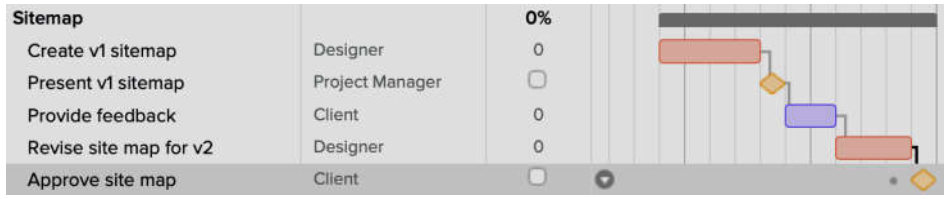


Figure 8.6

➔ *Why are milestones important?*

Milestones are critical to successful project management for the following reasons:

1. **Help Track Deadlines:** Defining central milestones in the planning phase of a project will help project managers stay up to date on all associated deadlines.
2. **Identify potential bottlenecks:** Many projects rely on work produced by external teams or partners. If these external factors are not tracked, delays and compression are likely to occur.
3. **Easily pinpoint critical dates:** Using milestones makes it easier to see the big picture and easily pinpoint important dates and events. You or your entire team may need to be off-site for a mandatory project-related training session.
4. **Increase Project Visibility -** Visibility can make things easier when it comes to project management. Everyone can see where a project is and what remains to be done.
5. **Allocation of Time and Resources:** Time and resources are critical to completing all successful projects. Using milestones helps managers distribute resources effectively so projects are delivered on time and within budget.

6. **Vendor payments often rely on milestone completion -** track and schedule payments to key vendors with milestone completion.
7. **Stakeholder participation varies according to the milestones;** Stakeholders tend to become more involved as a milestone approaches. Use milestones to plan when stakeholders should approach the project.
8. **Accountability:** Project teams need to see what they are responsible for. Milestones help everyone be accountable for the role they play.
9. **Demonstrate "success measures"** - it's great to be able to measure your success! Completing and passing all major milestones is a visible and successful way to demonstrate the overall success of a project.

3.4 IMPORTANT MILESTONES

The four main milestones occur at the transition points between life cycle stages. They can be used in many different process models, including the conventional waterfall model. In an iterative model, major milestones are used to gain competition between all stakeholders on the current state of the project. Different stakeholders have very different concerns:

- **Customers:** planning and budget forecasts, feasibility, risk assessment, understanding of requirements, progress, compatibility of the product line
- **Users:** consistency with requirements and usage scenarios, ability to adapt to growth, quality attributes
- **System architects and engineers:** compatibility of the product line, changes to requirements, analysis of trade-offs, integrity and consistency, balance between risk, quality and usability
- **Developers:** Sufficient details of requirements and descriptions of use scenarios. framework for the selection or development of components, resolution of development risks, compatibility of the product line, adequacy of the development environment
- **Maintainers:** adequacy of artifacts and product documentation, comprehensibility, interoperability with existing systems, adequacy of the maintenance environment
- **Others:** possibly many other stakeholder perspectives such as regulatory agencies, independent verification and validation contractors, venture capital investors, subcontractors, partner contractors, and sales and marketing teams

Table 8-1 summarizes the balance of information in the main stages.

Milestone	Plans	Requirements	Product
Fundamental objectives of the life cycle	1. Definition of stakeholder responsibilities 2.Low-fidelity life cycle plan 3.High fidelity creation phase plan	Basic vision, including growth vectors, quality attributes and priorities 2.User case model	1. Demonstration of at least one feasible architecture 2. Initial design model
Architecture milestone of the life cycle	1.Hi-fi construction stage plan 2. Low-fidelity transition plan	1.Stable vision and use case model 2. Evaluation criteria for the build version, initial operational capacity 3. Draft user manual	1.Stable design assembly 2. make exchanges of purchase / reuse 3 Prototypes of critical components
Milestone of initial operational capability	1.Hi-Fidelity transition plan	1. Acceptance criteria for product launch 2. Release the user manual	1.Stable distribution set 2.Critical characteristics and key capabilities 3. Objective information on product qualities
Milestones of product launch	1 Next Generation Product Plan	1. End of user manual	1.Stable distribution set 2.Full features 3.quality of complaints

➔ **Milestone of life cycle goals**

The milestone of the life cycle objectives occurs at the end of the initial phase. The goal is to present a recommendation to all interested parties on how to proceed with the development, including a plan, estimated cost and

time, expected benefits and cost savings. A successfully completed lifecycle goal milestone will entail authorizing all stakeholders to proceed to the development phase.

➔ **Architecture milestone of the life cycle**

The lifecycle architecture milestone occurs at the end of the build phase. The main goal is to demonstrate an executable architecture to all interested parties. The reference architecture consists of a human-readable representation (the architecture document) and a set of configuration-driven software components captured in the design artifacts. A successfully completed lifecycle architecture milestone will involve authorizing stakeholders to proceed to the construction phase.

➢ **Engineering certificates available from Lifecycle Architecture Milestone**

1. Requirements

A. Use the appropriate template
B. Vision document (text, use case)
C. Evaluation criteria for processing (text, scenarios)

2. Architecture

A. Tree view (object models)
B. Process view (runtime layout, executable code structure if necessary)
C. Component view (subsystem layout, make / but / reuse component identification)
D. Deployment view (target runtime layout, target executable code structure)
E. Display of use cases (test case structure, expectation of test results)
   1. Draft user manual

3. Source and executable libraries

A. Product components
B. Test the components
C. Environment and components of the instrument

➢ **Predefined agendas for the lifecycle architecture milestone**

❑**Presentation agenda**

I. Scope and objectives

A. Demo overview

II. Requirements assessment

   A. Project vision and use cases
   B. Primary scenarios and evaluation criteria

III. Architecture assessment

A. Progress

1. Baseline architecture metrics (progress to date and baseline to measure the stability, rejection and reworking of the future architecture).

2. Baseline estimate of development metrics (to assess future progress).
3. Baseline estimate of test metrics (to assess future progress of test equipment).

B. Quality

1. Architectural characteristics (synthesis of demonstrative capacity with respect to evaluation criteria)
2. Performance (summary of the ability to demonstrate with respect to evaluation criteria).
3. Architectural Risks Exposed and Resolution Plans.
4. Accessibility and trade-offs between production / purchase / reuse.

IV. Evaluation of the construction phase plan

A. Content of iteration and mapping of use cases

B. Subsequent detailed iteration plan and evaluation criteria

C. Cost of the development / execution phase of the program

D. Resource plan of the construction phase and estimation base

E. Risk management

❑ **Demo agenda**

I. Evaluation criteria

II. Architecture subset summary

III. Demo environment summary

IV. Demo scenarios with scripts

V. Results of the evaluation criteria and monitoring elements

➔ **Milestone of initial operational capability**

The initial operational capability milestone occurs at the end of the construction phase. The objectives are to assess the readiness of the software to initiate the transition to customer / user sites and to authorize the initiation of the acceptance test. Acceptance tests can be run incrementally across multiple iterations or can be completed completely during the transition phase, not necessarily the completion of the build phase.

➔ **Milestone of product launch**

The product launch milestone occurs at the end of the transition phase. The goal is to evaluate the completion of the software and its transition to the support organization, if applicable. The acceptance test results are reviewed and all open problems are resolved. The software quality metrics are reviewed to determine if the quality is sufficient for the transition to the support organization.

8.5 MINOR MILESTONES

For most sprints, which last from one month to six months, only two minor milestones are needed: the sprint preparation review and the sprint assessment review.

➔ **Review of iteration readiness**

This informal milestone is held at the start of each iteration to review the detailed iteration plan and evaluation criteria assigned to this iteration.

➔ **Review of the iteration evaluation.**

This informal milestone is held at the end of each iteration to assess the degree to which the iteration has met its objectives and met its evaluation criteria, to review the results of the iteration, to review the results of the qualification tests (if applicable). the iteration), to determine the amount of rework to perform and to review the impact of the iteration results on the plan for subsequent iterations. The format and content of these minor steps tend to depend heavily on the project and organizational culture.

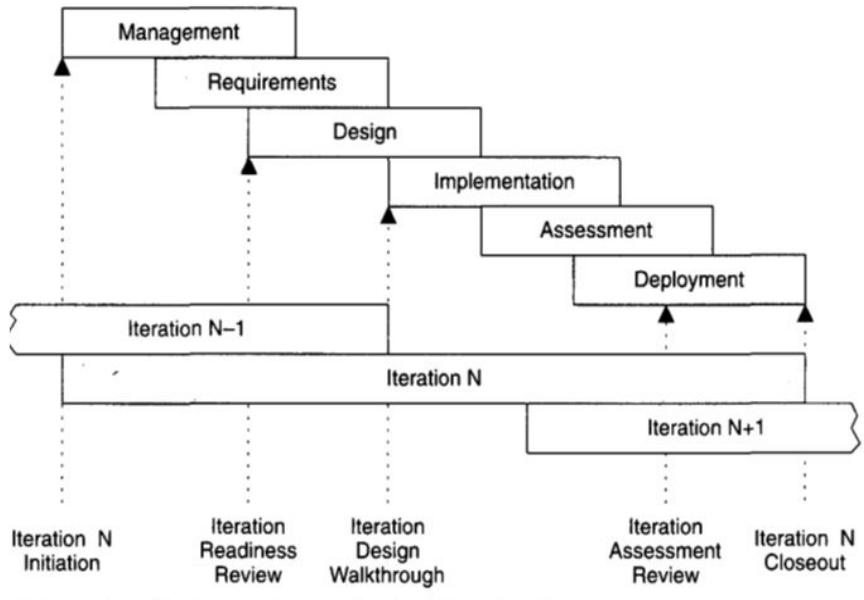


Fig. 8.7 Iteration workflow

➔ ***Why mark a milestone in a project?***

Milestones are more appropriate for those methodologies that want to maintain strict project deadlines. It is a springboard for project planning, helps manage stakeholder expectations and many more. Mainly, the definition of milestones addresses four areas:

❖ **What to complete?**

As a result, small segments of a project are monitored until completion to decide if they comply with the plan and schedule.

❖ **What was completed?**

All milestones that mark the progress of the project are recorded and determine the next stage of a project in project planning that can be started immediately.

❖ **When was it completed?**

Find performance effectiveness by comparing the actual completion date of a milestone with the planned one.

❖ **When will it be finished?**

In case there are variations in the project schedule compared to milestones, setting the milestones will help to estimate if the whole project will be completed on schedule (realistic time frame). You can see if you need to change your plans.

❖ **What are the efforts of your teams?**

Teams A and B are linked because team B cannot start work before team A completes it. So when should Team B be ready to start? Milestone will display each team's efforts. You will know when you have set an important goal for team A.

❖ **What is the sense of urgency?**

When there is a set deadline, each individual will do everything in their power to complete the task by the set date.

Aside from that, project timeline experts say milestones show progress only on the critical path, largely ignoring non-critical project activities.

❖ Task, event or goal?

Distinguishing tasks, events, and milestones can be difficult on larger projects.

- A task is a single item in a task list. It is something that must be achieved in a particular group.
- A milestone is a goal you want to achieve for a to-do list.
- An event is associated only with your calendar, which does not require a responsible deliverable and cannot be marked as "completed".

➔ *Benefits of defining project milestones*

Project Plan Milestones help you track the progress of a project. It keeps a project on track and the whole team on track. There are many benefits to setting project milestones:

❖ Plan your activities

Set up a timeline to complete the key points of a project. Milestones facilitate the planning of project activities. This will ensure that everyone

knows when the deadlines are set for the different aspects of the project. If for some reason it appears that milestones cannot be met, appropriate actions can be taken to get the project back on schedule to keep future milestones on track. More importantly, if you are behind schedule, you can easily accelerate to reach your project milestones.

❖ **Project evaluation on the fly**

As the project progresses, it can be evaluated at each stage. Without setting milestones, project managers generally leave the project evaluation for last. This costs him a lot. Then, with milestones, you can learn which aspects of the project worked well and where things could be improved. Monitoring project progress in real time from the start is important to ensure the successful completion of the project within budget.

❖ **Celebrate success**

As milestones are completed, this can be a celebration for the team and the manager. With a lot going on, the celebration usually gets out of hand. As milestones are set, to keep morale high, celebrate small successes. This will not only recognize the efforts of your team members, but it will also motivate them to work harder to achieve the overall goals of the project.

➔ *What is a milestone in the Gantt chart?*

Gantt charts are widely used in companies to track a project according to project management rules. In general, milestones are specific points in line with the project cycle to monitor project processes. Check off any significant activity in a project. A Gantt chart is a visual display of planned activities over time.

You can easily mark these milestones on Gantt charts;

✓ The start date of the project

✓ What are the tasks of the project

✓ Who is working on each business

✓ Starting and ending an activity

✓Duration of each activity

✓ The completion date of the project

➔ *Artifact test*

Description:-The test refers to the explicit evaluation through the execution of the implementation components set in a controlled scenario with an expected and objective result.

- Whatever document-based approach is applied to software development, it is also followed by people who test software.

- Development teams created requirement documents, high-level design documents, and detailed design documents before creating source files or executable files.
- Similarly, the test teams created system test plan documents, unit test plan documents, and unit test procedure documents before building any test controllers, stubs, or instrumentation.
- This document-based approach caused the same problems for testing and development activities.
- One of the really tasty beliefs of a modern process is to use exactly the same sets, notations and artifacts for the products from the testing activities that are used for product development.
- Test artifacts must be developed concurrently with the product from start to distribution. that is, to test a full life cycle activity, not a late life cycle activity.
- Test artifacts are communicated, designed and developed within the same artifact sets as the developed product.
- Test artifacts are implemented in programmable and repeatable formats like software programs.
- Test artifacts are documented in the same way the product is documented.
- The developers of the test artifacts use the same tools, techniques and training as the software engineers who develop the product.
- Testing is only one aspect of the assessment workflow. Other aspects include inspection, analysis and demonstration.
- The success of a test can be determined by comparing the expected result with the actual result with well-defined mathematical accuracy.

8.6 PERIODIC ASSESSMENT OF THE CONDITION

These are management reviews that are performed at regular intervals (monthly, quarterly) to review the progress and quality of the project and maintain open communication between all stakeholders.

- The main objective of this evaluation is to synchronize the expectations of all stakeholders and also to serve as snapshots of the project. Also provide,

1) A mechanism for openly addressing, communicating and solving project management problems, technical problems and risks.
2) A mechanism to disseminate information on processes, progress, quality trends, practices and experiences to and from all interested parties in an open forum.

3) Objective data derived directly from ongoing activities and from the evolution of product configurations. Iterative planning of the process:

- Like software development, project planning is also an iterative process.
- Like the software, the plan is also immaterial. Plans have an engineering phase, during which the plan is developed, and a production phase, where the plan is executed.

1. Interactive software development activities require ongoing attention to manage risk and assess project status.

2. Periodic health assessments are defined as management reviews carried out at regular intervals to indicate progress and quality indicators, paying constant attention to the dynamics of the project and maintaining communication between all stakeholders.

3. Periodic status reviews are periodic events in which management regularly reviews the progress of a project in order to meet stakeholder expectations.

4. Periodic health assessments are considered one of the critical control points of the project, as they give particular consideration to the gradual development of project priorities.

➔ **Status evaluation features:**

1. The assessment of the state aims at the periodic verification of the expectations of the interested parties.

2. State assessments are carried out by management in order to periodically check the development of a project.

3. Deals with issues related to project status progress or status assessment.

➔ **The need for health assessments in the software lifecycle:**

1. The main objective is to meet the expectations of interested parties in a synchronized and coherent way.

2. Status assessments serve as regular snapshots of a healthy project that includes risk assessment, management indicators, and quality indicators.

3. State assessment documents provide the mechanism to meet everyone's expectations, to communicate and solve project management, technical problems and risks.

4. Provide objective data on ongoing activities to develop product configurations.

5. It also provides a mechanism for the widespread use of processes, progress, quality trends, practices and experience information to all stakeholders on an ongoing basis.

6. Health assessments require the project manager to carry out periodic reviews and collect the data necessary to maintain the good health of the project.

7. Typical health assessments include reviews of resources, personnel, financials, or costs and revenues, top 10 risks, plans for important milestones, product scope, consequences and progress technical such as snapshots of metrics, etc.

➔ ***Engineering artifacts***

Vision document: -The vision document provides a comprehensive overview of the software system under development and supports the contract between the funding authority and the development organization. Whether the project is a massive military standard development (whose vision might be a 300-page system specification) or a small, internally funded commercial product (whose vision might be a two-page white paper), every project needs of a source. stakeholders. The vision of a project must change as the understanding of requirements, architecture, plans and technology evolves. A good vision document should change slowly.

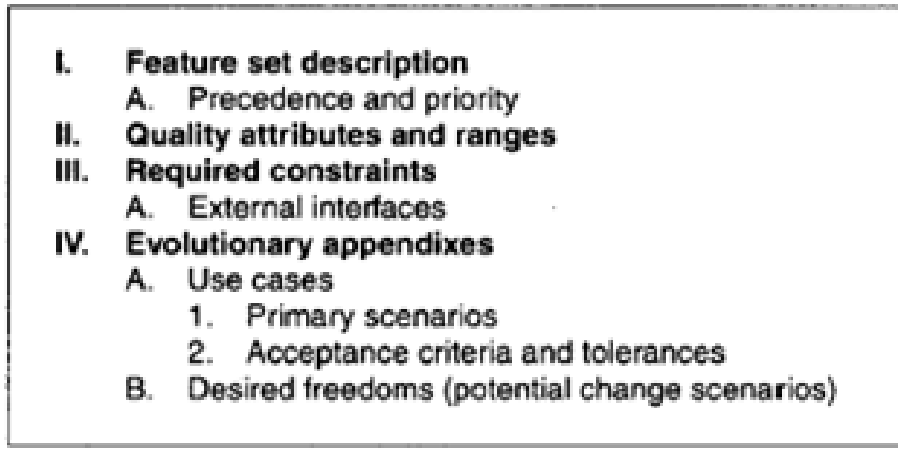


Fig.8.8 Typical online view document

➔ **The description of the architecture: -**The architecture description provides an organized view of the software architecture under development. It is largely drawn from the design model and includes sufficient views of the design, implementation, and deployment sets to understand how the operational concept of the established requirements will be achieved. The breadth of the architectural description will vary from project to project depending on many factors. Architecture can be described using a subset of the design model or as an abstraction of the design model with complementary material or a combination of both.

➔ **Software User Manual: -**The software user manual provides the user with the reference documentation necessary to support the supplied software. Although the content varies widely between application

domains, the user manual should include at a minimum installation procedures, guidance and procedures for use, operational limitations, and a description of the user interface. For software products with a user interface, this manual should be developed early in the life cycle because it is a necessary mechanism to communicate and stabilize an important subset of requirements. The user manual should be written by the test team members, who are more likely to understand the user's point of view than the development team. If the test team is responsible for the manual, it can be generated in parallel with development and can soon evolve as a tangible and relevant insight into the evaluation criteria. it also provides a necessary foundation for test plans and test cases, and for building automated test suites.

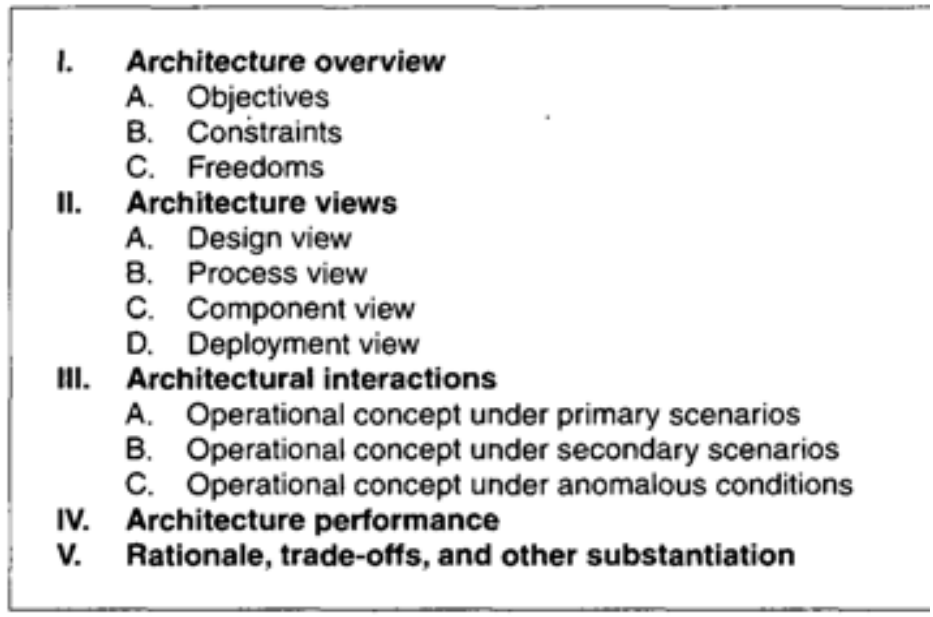


Fig. 8.9 Typical architecture description scheme

THEME	CONTENTS
➔ Personal	➔ Personnel plan vs. actual data ➔ Wear, additions
➔ Financial trends	➔ Spending plan versus actual data for previous, current and subsequent key milestones ➔ Income forecasts
➔ The 10 main risks	➔ Problem solving and criticality plans ➔ Quantification (cost, time, quality) of the exposure
➔ Technical progress	➔ Setting up basic schedules for important milestones

	➔ Software management metrics and indicators ➔ Current trends of change ➔ Test and quality ratings
➔ Results and plan of important goals	➔ Plan, program and take risks for the next major milestone ➔ Pass / Fail results for all acceptance criteria
➔ Total product scope	➔ Total disturbances of size, growth and acceptance criteria

Table 8.2 Default content of health assessment reviews

8.7 SUMMARY

- ➔ Establishing a series of checkpoints during the planning period and treating each checkpoint as if it were an actual delivery reduces the risks of missing the final delivery date, burning people and exceeding budget costs.
- ➔ To achieve this, checkpoints are established at 4-6 month intervals, but at significant points in the plan, where actual progress can be measured.
- ➔ With small projects, these important checkpoints can be established about a quarter, a half and three quarters of the way through to the final delivery date.
- ➔ Three sequences of project checkpoints are used to synchronize stakeholder expectations throughout the lifecycle, which is explained very briefly in this chapter.

8.8 BIBLIOGRAPHY

https://www.geeksforgeeks.org/

https://www.ittoolkit.com/articles/project-checkpoints

https://www.teamgantt.com/blog/the-how-and-why-of-using-milestones-in-your-project-plan

http://www.student.apamaravathi.in/meterials/spm/unit5.pdf

https://www.workflowmax.com/blog/general-project-milestone

http://www.pvpsiddhartha.ac.in/dep_it/lecture%20notes/SPM/unit3.pdf

https://www.proofhub.com/articles/what-is-a-milestone

http://www.faadooengineers.com/online-study/post/cse/software-project-management/163/evolutionary-work-breakdown-structure

http://grabcollectioncrazy.blogspot.com/2019/12/periodic-status-assessments.html

8.9 END OF CHAPTER EXERCISE

1. Explain the life cycle of the checkpoints with the types of joint management reviews.
2. Explain the planning and analysis of checkpoints.
3. What do you mean by Milestones? Why is it necessary?
4. Explain the main milestones.
5. What engineering elements are available in the lifecycle architecture milestone?
6. Explain the minor steps.
7. Explain the artifacts of the test.
8. Explain periodic condition assessment with engineering artifacts.

9

ITERATIVE PROCESS PLANNING

Unit structure

9.0 OBJECTIVES

This chapter will help us understand the process checkpoints required during the planning period. We will also focus on major milestones, minor milestones and periodic status assessment. One of the objectives of the process is to ensure that the expectations of all parties are synchronized and consistent. Periodic health assessment provides a mechanism to manage everyone's expectations throughout the project life cycle.

9.1 INTRODUCTION

These are management reviews that are performed at regular intervals (monthly, quarterly) to review the progress and quality of the project and maintain open communication between all stakeholders.

- The main objective of this evaluation is to synchronize the expectations of all stakeholders and also to serve as snapshots of the project. Also provide,

1) A mechanism for openly addressing, communicating and solving project management problems, technical problems and risks.

2) A mechanism to disseminate information on processes, progress, quality trends, practices and experiences to and from all interested parties in an open forum.

3) Objective data derived directly from ongoing activities and from the evolution of product configurations. Iterative planning of the process:

- Like software development, project planning is also an iterative process.

- Like the software, the plan is also immaterial. Plans have an engineering phase, during which the plan is developed, and a production phase, where the plan is executed.

9.2 WORK DISTRIBUTION STRUCTURES

- The work breakdown structure is the "architecture" of the project plan and also an architecture for the financial plan.
- We say that a project is a success, if we keep a good structure of work distribution and its synchronization with the framework of the process.
- A WBS is simply a hierarchy of elements that breaks down the project plan into distinct work activities and provides:

1) A pictorial description of all significant works.

2) A clear division of tasks for the attribution of responsibilities.

3) A framework for planning, budgeting and spending monitoring.

➔**Conventional WBS Issues:**

Conventional functioning rupture structures commonly suffer from three fundamental failures.

1) Conventional WBSs are prematurely structured around product design: Management

- System requirements and design
- Subsystem 1

➢ Component 11: {Requirements, Design, Code, Test, Documentation ,. . .}

➢Component 1N: {Requirements, design, code, test, documentation ,. . .}

- Subsystem M

➢Component M1: {Requirements, design, code, test, documentation ,. . .}

➢Component MN: {Requirements, Design, Code, Test, Documentation ,. . .}

- Integration and testing: {test planning, preparation of test procedure, tests, test reports}
- Other areas of support: {configuration control, quality assurance, system administration}

- It is a typical CWBS which has been structured mainly around the product architecture subsystem and then broken down into the components of each subsystem.
- Once this structure is incorporated into the WBS and then assigned to responsible managers with expected budgets, programs and results, a concrete planning basis has been established which is difficult and expensive to modify.

2) CWBS are prematurely broken down, planned and budgeted with too little or too much detail.

3) CWBS are project specific and comparisons between projects are often difficult or impossible

Conventional work stoppage structures often suffer from three fundamental flaws.

1. They are prematurely structured around product design.

2. They break down, plan and anticipate prematurely with too much or too little detail.

3. They are project specific and comparisons between projects are often difficult or impossible.

❏ Conventional work breakdown structures are prematurely structured around product design. Figure 3.2 shows a typical conventional WBS that has been structured primarily around the subsystems of its product architecture and then broken down into the components of each subsystem. A WBS is the architecture of the financial plan.

❏ Conventional work stoppage structures are broken down, planned and budgeted prematurely with too much or too little detail. Large software projects tend to be over-planned, while small projects tend to be poorly planned. The basic problem with planning too many details in advance is that the details don't evolve with the plan's loyalty level.

❏ Conventional work-sharing structures are project-specific, and comparisons between projects are often difficult or impossible. Without a standard WBS structure, it is extremely difficult to compare plans, financial data, planning data, organizational efficiencies, cost trends, productivity trends, or quality trends across multiple projects.

9.3 EVOLUTIONARY STRUCTURE OF WORK

❏ An evolving WBS should organize planning elements around the process structure rather than the product structure. The basic recommendation for the WBS is to organize the hierarchy as follows:

❏ The first level elements of the WBS are the workflows (management, environment, requirements, design, implementation, evaluation and distribution).

- ❑ The second level elements are defined for each stage of the life cycle (initiation, elaboration, construction and transition).
- ❑ The third level elements are defined for the focus of the activities that produce the artifacts of each phase.
- ❑ A default WBS consistent with the process framework (phases, workflows, and artifacts) is shown in Figure 10-2. This recommended structure provides an example of how elements of the process structure can be integrated into a plan. It provides a structure for estimating costs and schedules for each item, allocating them in a project organization and keeping track of expenses. The structure shown is simply meant to be a starting point. It has to be adapted to the particularities of a project in many ways.
  - ➔ Ladder. Larger projects will have multiple layers and substructures.
  - ➔ Organizational structure. Projects that include subcontractors or span multiple organizational entities may introduce constraints that require different WBS allocations.
  - ➔ Degree of custom development. Depending on the nature of the project, there can be very different emphases on requirements, design, and implementation workflows.
  - ➔ Business context. Projects that develop commercial products to supply to a large customer base may require much more elaborate substructures for the implementation element.
  - ➔ Previous experience. Very few projects start with a clean slate. Most of them are developed as new generations of a legacy system (with a mature WBS) or in the context of existing organizational standards (with predetermined WBS expectations).

The WBS breaks down the character of the project and assigns it to the life cycle, budget and personnel. Reviewing a WBS provides information on the important attributes, priorities, and structure of the project plan. Another important attribute of a good WBS is that the design fidelity inherent in each element is proportional to the current stage of the life cycle and the status of the project. Figure 10-3 illustrates this idea.

Figure 9-3 Default job breakdown structure

➔A gesture

★Management of the initial phase of AA

★AAA business case development

★AAB Construction Phase Launch Specifications

★WBS specifications for the development phase of AAC

★AAD software development plan

★ Evaluation of the progress and control of the project of the initial phase of AAE

❑AB Management of the development phase

★ ABA Construction Phase Release Specifications

★ Baseline WBS ABB construction phase

★ Evaluation of the progress and control of the project of the development phase of the ABC

❑Management of the CA construction phase

★ Planning the implementation phase of the ACA

★ CBA Baseline WBS implementation phase

★ Evaluation of the state and control of the project of the construction phase of the ACC

❑Management of the transition phase AD

★ Planning for the next generation of ADA

★ Assessment of the status and control of the project of the transition phase of ADB

➔B Environment

★ Specify the BA startup phase environment

★ BB Baseline of the development phase environment

★ Installation and administration of the BBA development environment

★ Integration of BBB development environment and custom tools locksmith

★ BBC SCO database formulation

❑ BC Maintenance of the environment during construction

★ Installation and administration of the BCA development environment

★ BCB SCO database maintenance

❑Maintain the DB transition phase environment

★Maintenance and administration of the BDA development environment

★ BDB SCO database maintenance

★ BDC maintenance environment package and transition

➔Requirements C.

★ Development of requirements for the CA start-up phase.

★ CCA Vision Specifications

★ CAB use case modeling

❑ CB Baseline of the requirements of the development phase

★ Baseline CBA Vision

★ Baseline of the CBB use case model

❑Maintain the requirements of the construction phase CC

❑ Maintaining the requirements of the CC transition phase

➔Design D

❑ Prototyping the DA boot phase architecture

❑ Baseline of the database development phase architecture

★ DBA architecture design modeling

★ Planning and running of the DBB Design demo

★ Description of the DBC software architecture

❑Modeling of the project under construction DC

★ Maintain the DCA architecture design model

★ Modeling the design of DCB components

❑ Maintenance of the DD transition phase project

➔And Implementation

❑ Prototyping of EA Startup components

❑ Implementation of the EB development phase component

★ EBA Critical Component Coding Demo Integration

❑ Implementation of the CE construction phase component

★ RCT initial release component coding and independent testing

★ ECB Alpha component coding and independent testing

★ ECC beta component coding and independent testing

★ ECD component maintenance

❑Evaluation F

★ Evaluation of the initial phase of AF

★ Evaluation of the FB development phase

❖ FBA test modeling

❖ Implementation of the FBB architecture test scenario

❖ FBC demo evaluation and release descriptions

❑ Evaluation of the construction phase of the FC

★ Evaluation of the initial version of the FCA and description of the version

★ FCB Alpha Version Evaluation and Version Description

★ FCC Beta Version Evaluation and Version Description

❑ Evaluation of the transition phase of FD

★ FDA product release assessment and release description

❑Distribution G

★ Planning the implementation of the start-up phase of GA

★ GB Planning the implementation of the elaboration phase

★ Implementation of the compilation phase of the GC

➢ Baseline of the GCA user manual

❑ Implementation of the transition phase GD

★ Transition from the GDA product to the user

Figure 10-3 Evolution of planning fidelity in the WBS over the life cycle

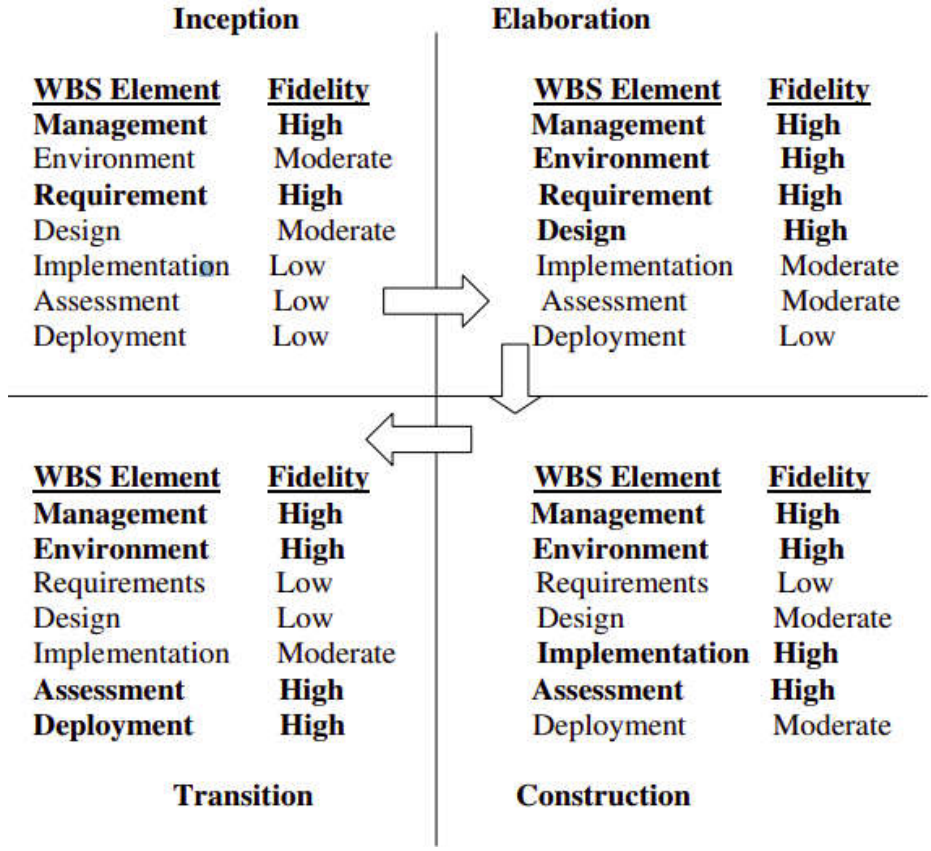


Figure 9.1

9.4 PLANNING GUIDELINES

Software projects cover a wide range of application domains. It is valuable but risky to make specific planning recommendations independent of the project context. Independent project planning advice is also risky. There is a risk that guidelines will be adopted blindly without being adapted to the specific circumstances of the project. Two simple planning guidelines should be considered when starting or evaluating a project plan. The first

guideline, detailed in Table 10-1, prescribes a predetermined allocation of costs between the first level WBS elements. The second guideline, detailed in Table 10-2, prescribes effort allocation and schedule during the life cycle stages.

10-1 Web budgeting defaults

First Level WBS Element	Default Budget
Management	10%
Environment	10%
Requirement	10%
Design	15%
Implementation	25%
Assessment	25%
Deployment	5%
Total	100%

Table 10-2 Default distributions of effort and schedule by phase

Domain	Inception	Elaboration	Construction	Transition
Effort	5%	20%	65%	10%
Schedule	10%	30%	50%	10%

Figure 9.2

9.5 THE COST AND PROGRAM ESTIMATION PROCESS

Project plans must be derived from two perspectives. The first is a forward-looking, top-down approach. It starts with understanding the general requirements and constraints, derives a budget and planning at the macro level, then breaks these elements into lower-level budgets and intermediate milestones. From this point of view, the following planning sequence would occur:

1. The software project manager (and others) develops a characterization of the overall size, process, environment, people and quality required for the project.

2. A macro-level estimate of the total effort and program is developed using a software cost estimation model.

3. The software project manager breaks down the effort estimate into a higher-level WBS using guidelines such as those in Table 10-1.

4. At this point, the subproject managers are responsible for breaking down each of the elements of the WBS into lower tiers using their top tier assignment, staffing profile, and milestone dates as constraints.

The second perspective is a bottom-up approach that looks backwards. We start with the end in mind, look at budgets and programs at the micro level, then add all of these elements to higher-level budgets and milestones. This approach tends to define and populate the WBS from the

lowest levels upwards. From this point of view, the following planning sequence would occur:

1. Lower level WBS elements are worked out in detailed tasks
2. Estimates are combined and integrated into budget and higher level targets.
3. Comparisons are made with top-down budgets and planning milestones.

Milestone planning or budget allocation through top-down estimation tends to exaggerate project management biases and usually results in an overly optimistic plan. Bottom-up estimates often exaggerate the biases of actors and result in an overly pessimistic plan.

These two planning approaches should be used together, in balance, throughout the project life cycle. During the design phase, you will dominate the top-down perspective because there is generally not enough depth of understanding and stability in the detailed task sequences to make credible bottom-up planning. During the production phase, there should be enough prior planning experience and loyalty to master the bottom-up planning perspective. The top-down approach should be well tuned to the specific parameters of the project, so it should be used more as an overall evaluation technique.

Figure 10-4 Planning balance throughout the life cycle

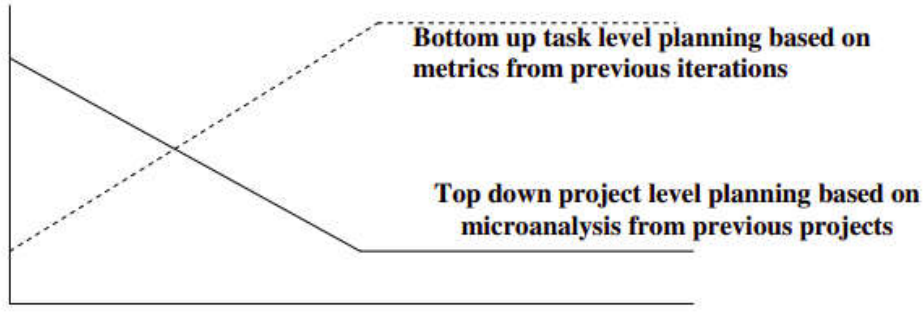


Figure 9.3

Engineering Stage		Production Stage	
Inception	Elaboration	Construction	Transition
Feasibility iteration	Architecture iteration	Usable iteration	Product Releases

Engineering stage planning emphasis	Production stage planning emphasis
Macro level task estimation for production stage artifacts	Micro level task estimation for production stage artifacts
Micro level task estimation for engineering artifacts	Macro level task estimation for maintenance of engineering artifacts
Stakeholder concurrence	Stakeholder concurrence
Coarse grained variance analysis of actual vs planned expenditures	Fine grained variance analysis of actual vs planned expenditures
Tuning the top down project independent planning guidelines into project specific planning guidelines	
WBS definition and elaboration	

Figure 9.4

9.6 THE ITERATION PLANNING PROCESS

Planning is concerned with defining the actual sequence of intermediate results. An evolving build plan is important because there are always adjustments to the content and build schedule as the initial conjectures evolve under well understood project circumstances. Iteration is used to indicate complete synchronization across the entire project, with a well-orchestrated overall assessment of the entire project baseline.

➔ **Boot iterations.**

Early prototyping activities integrate the building blocks of a candidate architecture and provide an executable framework for creating system-critical use cases. This framework includes enough existing components, commercial components and custom prototypes to demonstrate a candidate architecture and sufficient understanding of the requirements to establish a credible software development plan, vision and business case.

➔ **Processing of iterations.**

These iterations result in an architecture, which includes a complete framework and infrastructure for execution. After the architecture iteration is complete, it should be possible to demonstrate some critical use cases:

(1) initialize the architecture,

(2) insert a scenario to guide the worst-case data processing flow through the system (for example, peak transaction throughput or peak load scenario) and

(3) insert a scenario to guide the worst-case control flow through the system (for example, orchestrate fault tolerance use cases).

➔ **Construction iterations.**

Most projects require at least two major build iterations: an alpha version and a beta version.

➔ **Transition iterations.**

Most projects use a single iteration to go from beta to final product.

The general guideline is that most designs will use four to nine iterations. The typical design would have the following six iteration profiles:

★An iteration at the beginning: an architecture prototype

★Two iterations under development: architecture prototype and architecture reference

★Two iterations under construction: alpha and beta versions

★An iteration in transition: the product launch

A very large or unprecedented project with many stakeholders may require an additional initial iteration and two additional iterations under construction, for a total of nine iterations. 10.5

9.6 PRAGMATIC PLANNING

➢ While good planning is more dynamic in an iterative process, doing it accurately is much easier. When performing the N iteration of any phase, the software project manager must monitor and control a plan started in the N - 1 iteration and must plan the N + 1 iteration. tradeoffs on the current iteration plan and the next iteration plan based on the objective results in the current iteration and previous iterations. Aside from bad architectures and misunderstood requirements, improper planning (and consequent mismanagement) is one of the most common reasons for project failures. Conversely, the success of any successful project can be attributed in part to good planning.

➢ A project plan is a definition of how the requirements of the project will be transformed into "a product within the boundaries of the business". It has to be realistic, it has to be current, it has to be a team product, it has to be understood by the stakeholders and it has to be used. Plans aren't just for managers. The more open and visible the planning process and the results, the more ownership there will be among the team members who need to execute it. Wrong and closed floors cause friction. Good open plans can shape cultures and encourage teamwork.

9.7 SUMMARY

Planning must be continuous. Iterative planning is exactly what you think it is, make a plan, create software and then make another plan based on what has been learned.

This plan is a subset of the stories in the release plan that will be built in the next iteration or sprint. There is only one iteration plan

9.8 BIBLIOGRAPHY

1. Project Management Basic Manual: Mantel Jr., Meredith, Shafer, Sutton with Gopalan

   (Wiley India Edition)

2. Project Management - TYBSCIT -Semester 6 - Divya Shetty - Sheth Publications.

3. https://www.geeksforgeeks.org/

9.9 QUESTIONS AT THE END OF THE CHAPTER

1. Define a model-based software architecture?

2. Explain multiple process workflows?

3. Define a typical sequence of lifecycle checkpoints?

4. Explain the overall status of plans, requirements, and products at key milestones.

5. Does it explain the decomposition structures of conventional and evolutionary labor?

6. Briefly explain the balance of planning throughout the life cycle.

Unit IV

10

ORGANIZATION AND RESPONSIBILITY OF THE PROJECT

Unit structure

10.0 OBJECTIVES

1. Identifies the various functions represented in a project.
2. Analyzes and evaluates the influence of the organizational structure on the functions of the project.
3. Design a project flowchart for various project complexity profiles.

10.1 INTRODUCTION

➢ Proper organization of the project team is one of the key constraints for the success of the project. If the project does not have a well organized and productive team, there is a greater chance that this project will fail in the beginning because initially the team cannot get the project done right. Without proper organization of teamwork, team members will fail to perform a variety of specific roles and a variety of individual / group responsibilities. Therefore, when planning a new project, you must first take care of the best organization of the project team through team building activities.

➢ Organizing a project team is a typical activity of a project manager. Proper implementation of this task requires the manager to acquire,

develop and lead a group of people who are expected to complete the project. The organization of the project team is the responsibility of the project manager, who undertakes to build a productive team of professionals to ensure that the project deliverables are produced on time, budget and specifications, and therefore the client will accept those deliverables.

➔ ***What is a project team?***

➢ Before starting to organize a project team, it is essential to understand the definition of a project team. Senior supervisory staff (executives, project managers), as well as group leaders, must clearly understand the definition because such understanding is necessary to establish teamwork, maintain ongoing training, establish productive communications, and support collaboration. Here is the definition of the project team:

➢ A project team is an organized group of people who are involved in carrying out shared / individual project tasks, as well as in achieving shared / individual goals and objectives in order to achieve the project and produce its results. The team is made up of full-time and part-time human resources who should work collaboratively to produce the final results and successfully complete the project.

A group of people becomes a team when all the people in the group are able to meet the following conditions:

- Understand the work to be done as part of the effort.
- Planning to complete the assigned tasks
- Execute tasks within budget, schedule and quality expectations.
- Report issues, changes, risks, and quality issues to the leader
- Report the status of activities
- Be someone who can collaborate with others.

So, when looking for candidates for your project group, first make sure that a candidate is ready to meet all conditions; if not, switch to another candidate. If you understand this, you will have a better chance of finding the best candidates.

➔ ***Three conventional roles***

Each team, regardless of the type, size and nature of the project, has three roles (called "conventional"). These roles are:

- **Head.** A project team leader is a person who provides leadership and guidance to the team and takes responsibility for the results of teamwork. The role of team leader involves developing and encouraging the team through training, leadership, motivation,

recognition, reward, and other activities that encourage or compel team members to perform required activities.

- **Member.** A project team member is a person who is actually involved in carrying out the assigned tasks. Team members have direct access to the project and actively develop its processes. They are subordinate to the group leader.

- **Taxpayer.** A project team contributor is a person or organization who participates in teamwork but is not actually involved in carrying out the tasks or responsibilities of the project team. Contributors help improve the project through valuable suggestions, expert judgment and consultations. They are not responsible for the results of the project. Often, collaborators on the project team have an interest or concern for the project, thus facilitating its successful completion.

When the organization of the project team is adequate, all roles are assigned appropriately. Successful teams often work under the direction and supervision of project managers who oversee the work of the team leader and provide expert advice to team members. In this situation, taxpayers work in collaboration with managers.

10.2 RESPONSIBILITIES AND DUTIES

A team can be responsible for a variety of tasks and responsibilities, depending on the project they are involved in. A good project team organization implies the correct definition of the responsibilities and duties of the team when considering the specific aims and objectives of the project. Below are several common responsibilities and duties of a project team:

- Gain a correct understanding of the quantity and scope of the work assigned
- Following the planned assignments
- Increase the level of detail by task and activity as needed
- Complete assigned tasks within limits of scope, quality, time and cost.
- Inform the leader of any problems that arise.
- Communicate and collaborate proactively with other team members

➔ ***The organization chart***

➢ Typically, all possible roles, tasks, and responsibilities of a team are listed in the project team's organizational chart. You can read the definition below.

- A project team organization chart is a detailed, document-based graphical representation of the team to outline the specific roles, duties and responsibilities of team members and other stakeholders involved in the project and to formally define how exactly they are expected to collaborate with everyone. others during the project implementation process. It is also considered as a mechanism for managing team development processes through the design of training programs based on the group relationships established in the table.

- The team leader typically uses the organization chart to closely follow the processes associated with team management and to record the particular relationships between team members throughout the implementation lifecycle. Team members use the table to explore which roles and responsibilities have been assigned, who will share those roles, and who will manage and direct their efforts.

Here is a small checklist of key activities for creating a project team organization chart:

1. Make a list of the project team. First, you need to list all the people (and their names) who should be the participants on your project team. You can do this after finishing interviews with team candidates.

2. Assign conventional roles. Now you need to think about which individuals will take on which roles. Use the results of your interviews to start with the leaders and then list the members and collaborators.

3. Get the whole team together. Use your team list with details of the roles assigned to your people to form the team. This means that you have to formally form the team.

4. Identify the stakeholders. Your team is formed, now you need to identify the stakeholders or those people / organizations who have a direct interest or are interested in your project. I'm the sponsor and the client. Note that although stakeholders are not team participants, they are added to the project team's organizational plan because they influence the team's decisions.

5. Build the chart. Finally, use all the data to create the chart and show the relationships between the team and its stakeholders. The reports will show who reports to whom and what control mechanism is used to guide teamwork.

   - Project organization is actually a structure that simply facilitates and motivates the coordination and implementation of project activities.

   - Its main goal is simply to create an environment that encourages interactions between team members with a minimum number of interruptions, overlaps and conflicts. The most important decision of a project management team is the shape of the organizational structure that will be necessary and essential for the project. The

organization must evolve with the work breakdown structure (WBS) and life cycle concerns.

➔ **Line of business organizations:**

Below is a diagram showing the roles and responsibilities of a predefined line-of-business organization. Business line organizations must support projects with necessary and essential infrastructures to use a common process. Line of business simply a general term that describes and explains products and services offered simply by a company or manufacturer. Business software lines are generally motivated and supported by return on investment (ROI), new business discrimination, market diversification and profitability.

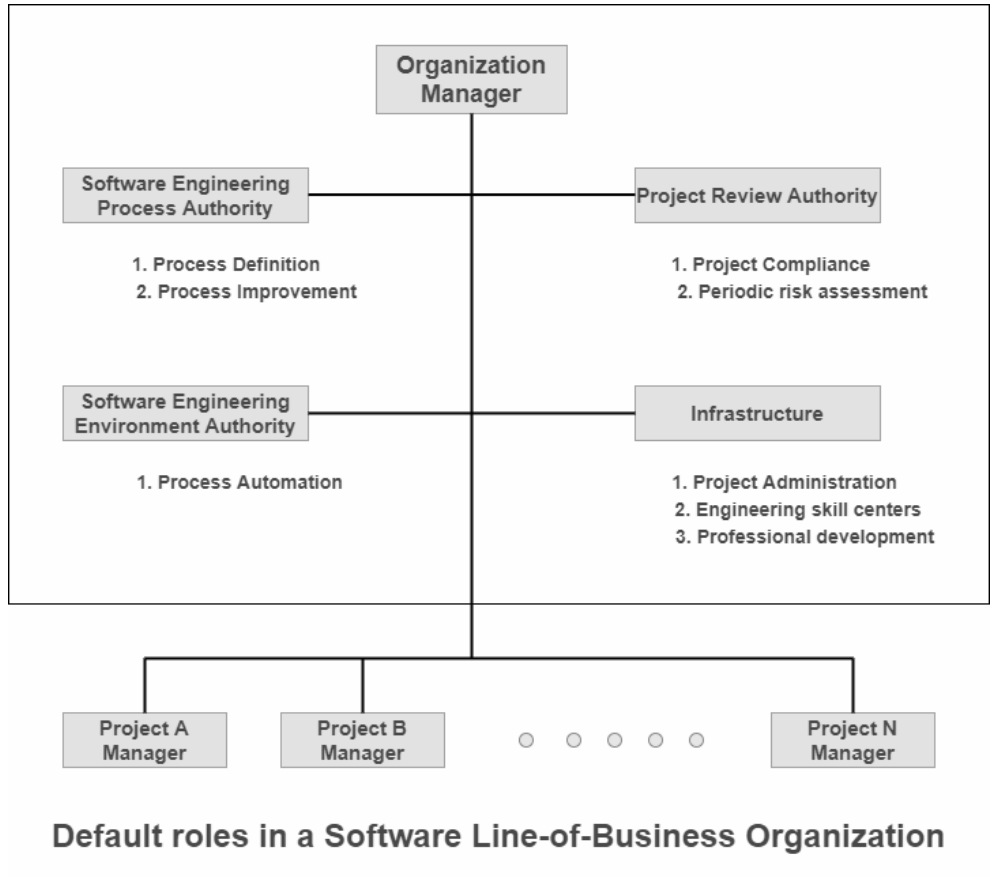


Figure 10.1

10.3 ORGANIZATIONAL RESPONSIBILITY

- They are generally responsible for defining the process, including maintaining the project process.
- They are also responsible for automating processes. This is an organizational role and it is equally important for that process definition role.

- Responsibility for the role of the organization or the role of process automation is assumed and carried out by a single individual or several other teams.

➔ **Various organizing authorities:**

1. **Software Engineering Processes Authority (SEPA) -** It is a team that deals with the exchange of information and the guidance of the project both towards and from the project professionals. Project professionals simply do the work and are generally responsible for one or more activities in the process. SEPA is a very important and essential role or responsibility in any organization.

2. **Project Review Authority (PRA) -** The project review is simply a scheduled status meeting that occurs on a regular basis. It includes the progress, problems and risks of the project. He is responsible for the review of the project. The PRA generally monitors both compliance with contractual obligations and organizational policy obligations of the project.

3. **Environmental Authority for Software Engineering (SEEA) -** SEEA is a very important role and it is very necessary to achieve ROI for a common process. You are simply responsible for supporting and managing a standard environment. For this reason, different tools, techniques and training can be amortized effectively on all types of projects.

4. **Infrastructure -** Organizational infrastructure is generally made up of systems, protocols, and various processes that provide structure to an organization, support human resources, and support the organization in fulfilling its vision, mission, goals, and values. It can range from mundane bureaucracies to entrenched bureaucracies. Various components of the organizational infrastructure are project management, engineering competence centers and professional development.

Below is the diagram showing the roles and responsibilities of a predefined project organization. Project organizations generally need to assign artifacts and responsibilities to the project team simply to ensure and confirm a balance between global (architecture) and local (component) concerns.

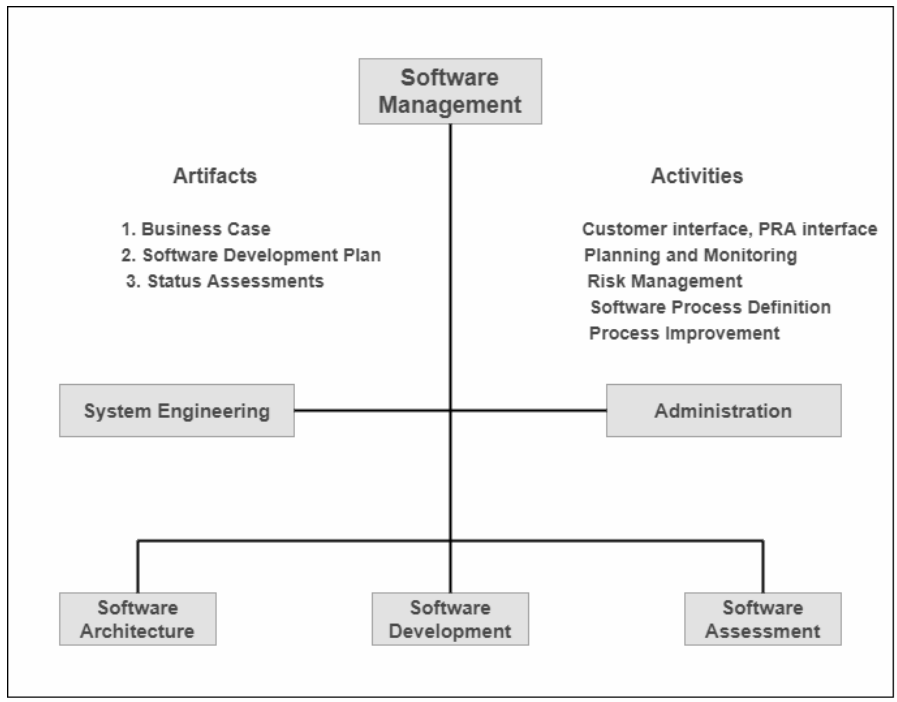


Figure 10.2

★ **Artifacts**

• Description of the architecture

• Launch specifications

★ **Activities**

• Demonstration planning

• Analysis, design

• Architectural prototypes

• Architecture documentation

• Coordination of demonstrations

• Component design

• Production / purchase / reuse analysis

★ **Artifacts**

• Implementation set

• Set of requirements

• Implementation set

★ **Activities**

• Component design

• Implementation of components

• Component testing

• Component maintenance

★ **Artifacts**

• Implementation set

• OCS database

• User manual

• Version descriptions

• Atmosphere

• Implementation documents

★ **Activities**

• Evaluation of the launches

• Testing of use cases / scenarios

• Development of test scenarios

• Change of management

• Transition to the user

• System administration

• Environment configuration

• Maintenance of the environment

• Blacksmith tools

The main features of the default organization are as follows:

• The Project management The team is an active participant, responsible for production and management. Project management is not a spectator sport.

• The architecture team is responsible for actual artifacts and component integration, not just staff functions.

• The development team owns the construction and maintenance of components. The evaluation team is separate from the development. This structure fosters an independent quality perspective and focuses a

team on product testing and evaluation activities along with ongoing development.

- Quality is everyone's job, integrated into all activities and checkpoints. Each team takes responsibility for a different quality perspective.

10.4 ORGANIZATIONAL TEAMS

- **Project Management Team -** He is an active and very enthusiastic participant. They are responsible for the production, development and management of projects.
- **Architecture team -** They are generally responsible for actual artifacts and even component integration. They also discover the risks of product mismatch with stakeholder requirements and simply make sure the solution fits a defined purpose.
- **Development team -** They are responsible for all the work required to produce functioning and validated goods.
- **Evaluation team -**They are responsible for assessing the quality of the final results.

➔ **Software management team**

Most of the projects are overloaded. Schedules, costs, features, and quality expectations are highly correlated and require continuous negotiation between multiple stakeholders who have different goals. The software management team has the burden of offering favorable terms to all interested parties. In this sense, the software project he spends every day worrying about balance. Figure 11-3 shows the focus of the software management team's activities during the project lifecycle.

The software management team is responsible for planning the effort, executing the plan, and adapting the plan to changes in requirements understanding or design. To this end, the team takes over resource management and software management.

➔ **Systems engineering responsibility**

Financial administration. Resource commitments Quality assurance. Staff assignments

- Plans, priorities
- Stakeholder satisfaction
- Definition of the field of application
- Risk management
- Project control

Start	Processing	Building	Transition
Processing phase Formulation of the planning team Baseline of the contract Architecture costs	Construction phase planning Full staff Risk resolution Product acceptance criteria Construction costs	Transition phase planning Optimization of the construction plan Risk management	Customer satisfaction Termination of the contract Sales support Next generation planning

Table 10.1

★ **Artifacts**

• Business case

• Software development plan

• Breakdown of work structure

• State assessments

• Requirements establish the scope of the project and establish operational priorities during the project life cycle. At an abstract level, these activities correspond to the management of the expectations of all stakeholders throughout the life cycle of the project.

The software management team takes care of all aspects of quality. In particular, you are responsible for achieving and maintaining a balance between these aspects so that the overall solution is suitable for all stakeholders and optimal for as many of them as possible.

➔ **Software architecture team**

- The software architecture team is responsible for the architecture. This responsibility includes the engineering required to specify a complete bill of materials for the software and the engineering required to achieve significant trade-offs between manufacturing and purchasing so that all custom components are made to the extent that construction / assembly costs are highly predictable. Figure 11-4 shows the focus of the software architecture team's activities during the project lifecycle.

- For any project, the skill of the software architecture team is critical. It provides the framework for facilitating team communications, for achieving system-wide quality, and for implementing applications. With a good architecture team, an average development team can be successful. If the architecture is weak, even an experienced development team of superstar programmers will likely fail.

- In most projects, the startup and development phases will be dominated by two separate teams: the software management team and the software architecture team. (This distinction can also be blurred,

depending on the scale.) Software development and evaluation teams tend to participate in support roles as they prepare for certificates.

★ **Artifacts**

• Description of the architecture

• Set of requirements

• Launch specifications

★ **Software architecture**

- demo

- Use case modelers

- Design Modeler performance analysts

Life cycle approach

★ **Responsibility**

• Compensation of requirements

• Design of compensations

• Selection of components

• Initial integration

• Technical resolution of risks

Life cycle approach

Start	Processing	Building	Transition
Architecture prototypes Compendiums / purchases Definition of the main scenario Definition of the architectural evaluation criteria	Architecture baseline Main scenario Demonstration Make / Buy Offset Baseline	Maintaining the Architecture Troubleshooting Multiple Components Performance Optimization Quality Improvements	Architecture maintenance Multiple components Troubleshooting Performance tuning Quality improvements

Table 10.2

- When the construction phase begins, the architecture enters maintenance mode and must be supported by a minimum level of effort to ensure the continuity of the engineering heritage.
- To be successful, the architecture team must include a broad enough level of experience, including the following:
- Domain experience to produce an acceptable design view (architecturally significant elements of the design model) and a use case view (architecturally significant elements of the use case model)

- Expertise in software technology to produce an acceptable process view (concurrency and control thread relationships between design models, components and distribution), component view (distribution assembly structure), and distribution view (assembly structure of distribution).
- The architecture team is responsible for system-level quality, which includes attributes such as reliability, performance, and maintainability. These attributes span multiple components and represent the degree of integration of the components to provide an effective solution. In this sense, the architecture team decides how to solve most of the multicomponent design problems.

★ **Software development team**

Figure 11-5 shows the focus of the software development team's activities during the project lifecycle.

Software development

➔ **Artifacts I— Kit of components**

• Implementation set

• Implementation set

Life cycle approach

➔ **Responsibility**

• Component design

• Implementation of components

• Independent component testing

• Component maintenance

• Documentation of components

➔ **Life cycle approach**

Start	**Processing**	**Building**	**Transition**
Support for Make / Buy Offsets prototypes	Critical Component Design Critical Component Implementation and Testing Critical Component Baseline	Component Design Component Implementation Independent Test Component Component Maintenance	Component maintenance component documentation

Table 10.3

The software development team is the most specific group of applications. In general, the software development team includes several sub-groups dedicated to component groups that require a common set of skills. Typical skill sets include the following:

- Business Component: Specialists with detailed knowledge of the business components critical to a system architecture.
- Database: Specialists with experience in organizing, archiving and retrieving data.
- Graphical User Interfaces: Specialists with experience in screen organization, data presentation and user interaction needed to support human input, output and control needs.
- Operating systems and networks: Specialists with experience in running multiple software objects on a network of hardware resources, including all the typical control problems associated with initialization, synchronization, resource sharing, namespace management, reconfiguration, termination and communications between subjects.
- Domain Applications: Specialists with experience in algorithms, application development, or system-specific business rules.

The software development team is responsible for the quality of the individual components, including development, testing and maintenance of all components. Component testing should be built as repeatable, self-documented software that is treated as the source code of another operational component so that it is naturally maintained and available for automated regression testing. The development team decides how to solve local implementation or individual component design problems.

➔ **Software evaluation team**

Figure 11-6 shows the focus of the software evaluation team's activities during the project lifecycle.

There are two reasons for using an independent software evaluation team. The first has to do with ensuring an independent quality perspective. This often debated approach has its advantages (such as ensuring that developer-owned biases don't taint the quality assessment) and its disadvantages (like freeing the software development team from quality ownership, to some extent). A more important reason to use an independent test team is to take advantage of business competition. Programs can be accelerated by developing software and preparing for testing in parallel with development activities. Change management,

★ **Software evaluation**

➔ **Artifacts**

• Implementation Set> SCO Database »User Manual 1 Environment

■ Version 1 specifications Version descriptions

■ Implementation documents

Start Test Change Management Deployment Environment Support

➔ **Life cycle approach**

➔ **Responsibility**

> Project infrastructure

> Independent tests

• Verification of requirements

> Analysis of metrics

> Configuration control

• Change of management

> User distribution

➔ **Life cycle approach**

Start	**Processing**	**Building**	**Transition**
Infrastructure planning Prototyping of primary scenarios	Initial user manual for change management of the infrastructure core architecture release test	Infrastructure Upgrades Startup Test Change Management User Guide Check basic requirements	Infrastructure maintenance Basic startup Change management Implementation for users Check requirements

Table 10.4

FOR modern process You should employ skill-based or use-case-based testing (which can span many components) organized as a sequence of builds and mechanized through two artifacts:

1. Version specification (the plan and evaluation criteria for a version)

2. Description of the publication (the results of a publication)

- Each version may include several (possibly incomplete) components, because integration occurs continuously. The evaluation criteria will document what the customer can expect to see in aimportantmilest one and version descriptions will support the test results. Final iterations will generally be equivalent to acceptance tests and will include levels of detail similar to levels of detail in conventional software test plans, procedures and reports. These artifacts evolve from rather short abstract versions in early iterations to more detailed and rigorous documents, with detailed discussions of integrity and traceability in

later versions. Also for use case testing, test components should be developed in a similar way to component test case development. For example, instead of developing documents on the test procedure,

- Some component tests can be elevated to evaluation criteria and their results are documented in the release descriptions. Many components can only be subjected to informal component testing by the development team, and the results are captured only within test software created by a developer. Formal tests for many components will then be included in the higher-level assessment criteria (typically capacity-oriented or thread-oriented scenarios) and corresponding version descriptions. Not all components are created equal - some require formal component testing to verify requirements, while others are best tested in the context of the capability test.

- The evaluation team is responsible for the quality of the reference versions against the customer's requirements and expectations. Therefore, the evaluation team is responsible for exposing any quality issues that affect customer expectations, regardless of whether these expectations are contained in the requirements or not.

10.5 ROLES AND RESPONSIBILITIES OF THE PROJECT TEAM

Successful projects are often the result of careful planning and the talent and collaboration of members of a project team. Projects can't go on without each of your key team members, but it's not always clear who those members are or what roles they hold. Here, we will describe five roles: project manager, project team member, project sponsor, executive sponsor, and business analyst, and describe their associated tasks.

➔ ***Project Manager***

The project manager plays an important role in the project and is responsible for its successful completion. The manager's task is to ensure that the project is developed within the specified time period and within the established budget, achieving its objectives. Project managers ensure that projects have adequate resources, while managing relationships with contributors and stakeholders.

Tasks of the project manager:

- Develop a project plan
- Manage the results according to the plan.
- Hire project staff
- Lead and manage the project team.
- Determine the methodology used in the project.
- Establish a project schedule and determine each phase

- Assign tasks to project team members
- Provide regular updates to senior management.

➔ *Project team member*

Project team members are the people who are actively working on one or more phases of the project. They can be internal staff or external consultants who work on the project full-time or part-time. The roles of team members can vary according to each project.

Duties of project team members may include:

- Contribute to the overall objectives of the project
- Complete individual deliverables
- Bringing experience
- Collaborate with users to establish and meet business needs
- Document the process

➔***Sponsor of the project***

The sponsor of the project is the promoter and the internal supporter of the project. They are usually members of top management, those who have an interest in the outcome of the project. Project sponsors work closely with the project manager. They legitimize the project objectives and participate in high-level project planning. In addition, they often help resolve conflicts and remove obstacles that occur during the project and sign the necessary approvals to go through each stage.

Duties of the project sponsor:

- Make key business decisions for the project.
- Approve the project budget
- Ensure the availability of resources
- Communicate the objectives of the project to the whole organization.

➔***Executive sponsor***

The executive sponsor is ideally a senior member of management. He or she is the visible champion of the project with the management team and is the final decision maker, with final approval at all stages, final results and scope changes.

The duties of the executive sponsor generally include:

- Take ultimate responsibility for the project.
- Approve all changes to the scope of the project
- Provide additional funding for scope changes
- Approve the results of the project

➔***Business analyst***

The business analyst defines needs and recommends solutions to improve an organization. When part of a project team, they ensure that the project objectives solve existing problems or improve performance and add value to the organization. They can also help maximize the value of project results.

Duties of the business analyst:

- Help define the project
- Collect requirements from business units or users.
- Document the technical and business requirements
- Verify that the project results meet the requirements.
- Test solutions to validate goals

10.6 EVOLUTION OF THE ORGANIZATION

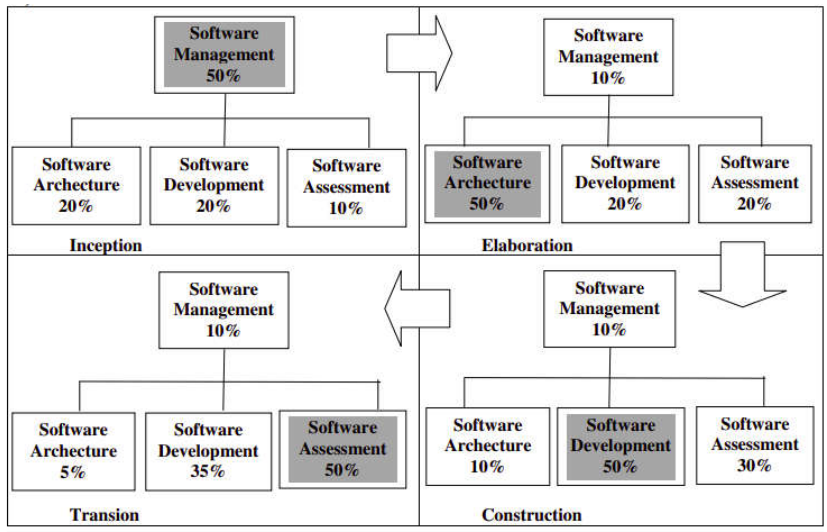


Figure 10.3

Inception: Software management: **50%** Software Archecture: 20% Software development: 20% Software Assessment (measurement/evaluation):10%	**Elaboration:** Software management: 10% Software Archecture: **50%** Software development: 20% Software Assessment (measurement/evaluation):20%
Construction: Software management: 10% Software Archecture: 10% Software development: **50%** Software Assessment (measurement/evaluation):30%	**Transion:** Software management: 10% Software Archecture: 5% Software development: 35% Software Assessment (measurement/evaluation):**50%**

Figure 10.4

10.7 SUMMARY

- The software business lines are driven by return on investment, new business discrimination, market diversification and profitability.
- Project teams are motivated by the cost, planning and quality of specific deliverables.
- Software professionals, in any type of organization, are motivated by professional growth, job satisfaction and the opportunity to make a difference.
- Projects rarely invest in technology or services that do not directly affect costs, planning or the quality of results.
- The organization of a project is temporary.

10.8 BIBLIOGRAPHY

1. https://mymanagementguide.com/basics/project-team-organization-project-team-definition-responssibility-and-roles-and-project-team-organization-chart/
2. https://www.geeksforgeeks.org/project-organizations-and-their-responssibility/
3. https://www.villanovau.com/resources/project-management/project-team-roles-and-responssibility/
4. https://www.gristprojectmanagement.us/software-3/project-organizations.html

10.6 QUESTIONS AT THE END OF THE CHAPTER

1. Explain the work breakdown structure.
2. What are the different roles and responsibilities in a project team?
3. What are the responsibilities of an organization for the project?
4. What equipment is included for the development of the project?
5. Explain the roles and responsibilities of the project team.
6. Explain the evolution of the organization.

11

PROCESS AUTOMATION

Unit structure

11.0 OBJECTIVE

- Process automation is used to describe the automated process through the use of computers and computer software.
- Processes that have been automated require less human intervention and less human time to run.
- The process levels are explained together with their tools.
- The project's environmental articles are briefly discussed along with three distinct states and four major environmental disciplines.

11.1 INTRODUCTION

- Many software development organizations focus on evolving mature processes to improve the predictability of software management and the performance of their software business lines (in terms of product quality, time to market, return on investment and productivity). While process definition and adaptation is necessary, a significant level of process automation is also required modern software development projects to operate profitably.
- Automation needs grow with the scale of commitment. Just as the construction process varies depending on whether you are building a dollhouse, a single family house, or a skyscraper, the software process varies across the spectrum from a single person. spreadsheet work on large-scale, multi-organization catastrophic failure cost applications. Techniques, training, timing, acceptance criteria, and levels of automation differ significantly at opposite ends of the spectrum.

- The majority organizations they are faced with the task of integrating their environment and infrastructure for software development. This process generally results in the selection of more or less incompatible tools that have different information repositories, are provided by different suppliers, work on different platforms, use different jargon and are based on different process assumptions. Integrating such an infrastructure proved to be much more problematic than expected.

Key points

- ❖ The environment must be first process artifact.
- ❖ Process automation, and in particular change management, are critical to an iterative process. If the change is too expensive, the development organization will resist.
- ❖ Round-trip engineering and integrated environments promote the freedom of change and the effective evolution of technical artifacts.
- ❖ Automation of metrics is critical to effective project control.
- ❖ External stakeholders need access to environmental resources to improve interaction with the development team and add value to the process.

- Automating the development process and building an infrastructure to support the various project workflows are important activities in the design phase of the life cycle. They include selecting tools, producing custom tools, and automating the processes needed to execute the development plan with acceptable efficiency. The evolution from the development environment to the maintenance environment is also crucial to any long-term software development project.

- To complicate matters further, it is rare to find interested parties who treat the environment as a first-class artifact required for ongoing product maintenance. The environment provided by process automation is a key tangible artifact for the life cycle cost of the system under development.

- In addition, we have introduced three levels of process. Each level requires some degree of process automation for the corresponding process to run efficiently:

1. Metaprocess: an organization's policies, procedures and practices for managing a software-intensive line of business. Automation support for this layer is called infrastructure. An infrastructure is an inventory of favorite tools, artifact models, microprocess guidelines, macro process guidelines, project performance repository, organizational skill set database, and a library of previous examples of previous project plans and results.

2. Macroprocess: the policies, procedures and practices of a project to produce a complete software product within certain costs, deadlines and quality constraints. Automation support for a design process is called an environment. An environment is a specific collection of tools to produce a specific set of artifacts governed by a specific project plan.

3. Microprocess: the policies, procedures and practices of a project team to obtain a software process artifact. The automation support for generating an artifact is generally called a tool. Typical tools include requirements management, visual modeling, compilers, editors, debuggers, change management, metrics automation, document automation, test automation, cost estimation and workflow automation.

While the primary focus of process automation is the workflow of a project-level environment, the infrastructure context of the project's parent organization and the building blocks of the tool are important prerequisites.

11.2 PROCESS AUTOMATION

Also called business process automation, it is the ability to coordinate and integrate tools, people and processes throughout the workflow. Process automation reduces human error, enables faster response to mission-critical system problems, and enables more efficient resource allocation. Sproutivity can streamline internal business workflows and bridge the gap between action, awareness and knowledge across all departments of the organization. Sproutivity with its experience provides a variety of tools that make your organization 100% proactive. We develop and build systems that can sustain themselves without effort or human resources.

Sproutivity recommends robust applications as efficiently and effortlessly as possible with the following aspects:

- Data Acquisition: List for event collection, correlation and analysis.
- Process Automation: Develop, implement and activate a rules-based intelligent dynamic workflow solution.
- Automation implementation: extension of an existing IT system, suggestion of a process-specific business process automation software, suggestions of an adaptive business process automation solution
- Two-Way Communication - Solution-based knowledge and task distribution, communication and scalability through a variety of interfaces, such as mail, SMS or phone calls.
- Management and control: execution with remote commands, real-time indicators and mobile control.

➔Benefits of Business Process Automation:

- Empower local managers while maintaining corporate oversight
- Gain real-time visibility into financial, human resources and administrative performance.
- Enforce compliance to establish processing standards across multiple locations
- Automatically fill out a complete audit trail
- Automate accountability
- Simplify processing, regardless of the source of the document
- Reduce data entry costs and mitigate entry errors
- Save on paper storage costs and support disaster recovery
- Gives vendors visibility into payment processing

Process automation generally refers to the use of digital technology simply to work and execute one or more processes. This is done to get or complete the workflow or function. For an iterative process, process automation and change management are very critical. Even if the change will be too expensive, development will endure and not allow it. To automate the software development process, several tools are currently available.

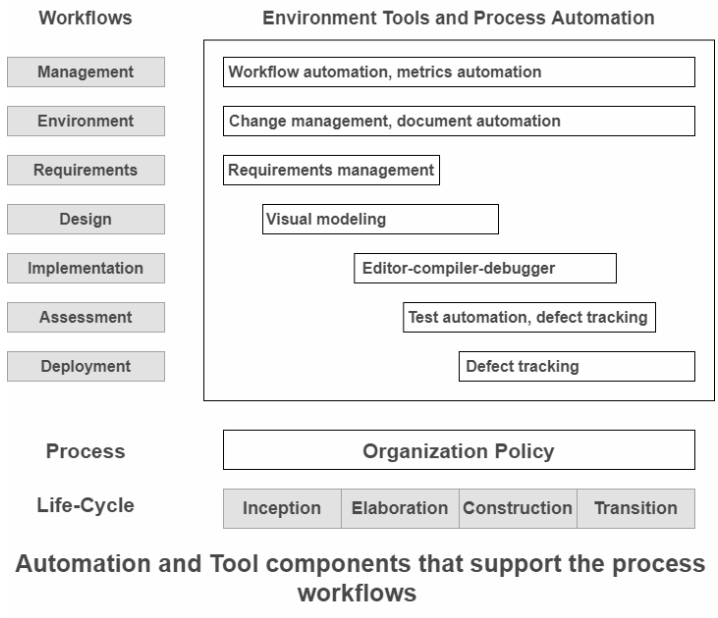


Figure 11.1

In the diagram above, some important tools are included and introduced which are very necessary in the whole software process in general and correlate very well with the process structure. Each of the software

development tools is closely related to one of the process workflows, and each of these workflows has different automation support. Workflow automation generally makes complicated software processes an easy way to manage. Here you will see an environment needed to support the process structure.

Here are some of the concerns associated with each workflow:

1. **Management:**
   Today, several opportunities and possibilities are available for automating project planning and management workflow control activities. To create planning artifacts, several tools are useful, such as software cost estimation tools and Work Breakdown Structure (WBS) tools. Workflow Management Software is an advanced platform that provides flexible tools to improve the way you work efficiently. Therefore, automation support can also improve understanding of metrics.

2. **Atmosphere:**
   Automating the development process and also developing an infrastructure to support different project workflows are very essential activities of the life cycle design phase. An environment that generally provides and provides process automation is a tangible artifact that is generally very critical to the life cycle of a system under development. Even the top-level WBS recognizes an environment as a first-class workflow. Integrating their software development environment and infrastructure is one of the primary tasks of most software organizations.

3. **Requirements:**
   Requirements management is a very systematic approach to identify, document, organize and monitor the evolving requirements of a system. It is also responsible for establishing and maintaining an agreement between the user or customer and the project team on changing system requirements. If a process requires strong traceability between requirements and design, the architecture is likely to evolve to optimize requirements traceability and project integrity. This effect is even more and highly effective and pronounced if process automation tools are used. For effective requirements management,

4. **Design:**
   Workflow design is actually a visual description of each step involved in a workflow from start to finish. It typically presents each activity in sequence and provides complete clarity on how data moves from one activity to another. Workflow design tools simply allow us to graphically represent the different activities involved, as well as represent actors, timelines, data and other aspects crucial to execution. Visual modeling is the primary support required and essential to the design workflow. The visual model is typically used to capture design models, render them in a human readable format, and also translate them into source code.

5. **Implementation:**
   The main goal and purpose of the distribution workflow is initially to write and test the software, relying mainly on the programming environment (editor, compiler, debugger, etc.). On the other hand, it should also include substantial integration along with change management tools, visual modeling tools, and test automation tools. This is simply necessary for the iteration to be productive. It is the main focus of the construction phase. Implementation simply means turning a project template into an executable one.

6. **Evaluation and implementation:**

   Workflow Assessment is the first step in identifying outdated software processes and simply replacing them with the most efficient process. This generally combines expertise in the collection and collection of qualitative and quantitative information, proprietary tools, and much more. You need and need all the tools discussed along with some additional features simply to support test automation and test management. Defect monitoring is also a tool that supports evaluation.

11.3 THE 4 PHASES OF PROJECT MANAGEMENT

1. Initiation and planning

To start outlining a project, you need to identify the scope, deliverables, and stakeholders. When the project is approved, a comprehensive plan is created to fully visualize how the process will run, with associated goals and work.

Project plans require a lot of organization. Some of the processes include:

- Establish a schedule and budget
- Identification of costs, materials and human resources.
- Provide training
- Identification of risks, obstacles and possible bottlenecks

Effective process management can help project managers organize themselves in five ways:

- By visually capturing your processes and attaching supporting documentation, project managers can make the most accurate recent information available in one place that everyone can access.
- Project managers and their resources can obtain information on existing processes related to the new project, thus obtaining an overview of who and what will be affected by the changes.
- By acquiring new processes related to the project, all members of the organization will be kept up to date, with notifications in their control panel that will alert them of any action they need to take.

- Project-related risks can be managed and monitored on the process management platform, giving executives the confidence that potential threats are being monitored.
- Team members who are new to the organization or unsure of what is expected of them can refer to the relevant processes for detailed information they can access with all the necessary details.

2. Execution

Once the project has started, it is up to the project manager to supervise everything. The execution processes for a project manager generally include:

- Organize workflows and activities
- Follow-up resources
- Communicate progress to stakeholders
- Organize regular meetings with stakeholders and workers.
- Troubleshooting for budget or resource issues

To ensure that all aspects of the project are visible, project managers can share their data-based projections of how the project is progressing, within the relevant process maps.

Real-time and auto-generated reports can also be stored within the tool, giving project managers the confidence that teams are viewing the latest version of relevant reports.

3. Monitoring and control

This phase runs alongside the execution and monitors the performance of the project.

To make sure a project runs smoothly, project managers:

- Keep track of the project budget, KPIs and SLAs
- Redeploy resources
- Monitor activities to avoid downtime
- High quality final results monitoring
- Monitor project performance

During this phase of the project, the project manager may need to make changes to accommodate new developments as they occur.

When processes are updated and corresponding documents are centrally stored within a process management platform, executives are confident

that all members of the organization are kept up to date during this transition period.

Project managers can also keep up with people's suggestions for change as teams review existing processes and recommend improvements.

In this way, everyone in the organization actively participates in the success of the project.

4. Closing the project

When a project comes to an end, the project manager and his team will usually assess whether it has been a success. Have you achieved the agreed goals? Were the results of a high standard? Can the project be considered finished?

But project management must also be lasting value, leading organizations to better results that continue to deliver value years after the project is completed.

It is true that analysis is critical for project managers to present to stakeholders and indicate overall post-project success.

To close a project, the project manager:

- Take steps such as communicating with stakeholders about the closure status of the project
- Review the success of the project in relation to the agreed scope
- Analyze team performance
- Evaluate the effectiveness with which the resources have been used.
- Redistribute any unused budgets or resources
- Provide project data and information to interested parties.

Analyzes are critical for managers to present to stakeholders and indicate overall post-project success. Operational metrics will provide a comprehensive overview of project success in terms of productivity, budget and efficiency, and this information can be used to improve and optimize future projects.

11.4 THE PROJECT ENVIRONMENT

Artifacts in the project environment evolve through three distinct states: the prototyping environment, the development environment, and the maintenance environment.

1. The prototyping environment includes an architecture test bed for prototyping design architectures to evaluate trade-offs during the startup and build phases of the life cycle. This informal setup of tools should be able to support the following activities:

- Performance compensation and technical risk analysis
- Conduct / purchase offsets and feasibility studies for commercial products
- Dynamic reconfiguration / fault tolerance compensation
- Analysis of the risks associated with the transition to a large-scale implementation
- Development of test scenarios, tools and adequate instrumentation to analyze requirements.

2. The development environment should include a full set of development tools needed to support the various process workflows and support back and forth engineering to the fullest extent possible.

3. The maintenance environment should normally correspond to a mature version of the development environment. In some cases, the maintenance environment may be a subset of the development environment that is provided as one of the project's end products.

Moving to a mature software process presents new challenges and opportunities for management control of competing businesses and for assessing tangible progress and quality. The project's real-world experience has shown that a highly integrated environment is needed to both facilitate and apply process management control. To this end, there are four important environmental disciplines that are critical to the management context and the success of modern management. Iterative development process:

1. Tools need to be integrated to maintain consistency and traceability. Round-trip engineering is the term used to describe this key requirement for environments that support iterative development.

2. Change management must be automated and applied to manage multiple iterations and allow for the freedom of change. Change is the fundamental primitive of iterative development.

3. Organizational infrastructures make it possible to derive project environments from a common base of processes and tools. A common infrastructure promotes project consistency, reuse of training, reuse of lessons learned, and other strategic improvements to the organization's met process.

4. Expanding automation support for stakeholder environments allows for greater support for without paper more effective information exchange and review of engineering artifacts.

❖ As the software industry moves towards maintaining different sets of information for engineering artifacts, more automation support is needed to ensure an efficient and error-free data transition from one

artifact to another. Round-trip engineering is the environmental support needed to maintain consistency between engineering artifacts.

- Figure 4.2 shows some important transitions between information repositories. The automatic translation of design models into source code (direct and reverse engineering) is quite well established. Automatic translation from design models to process (deployment) models is also simplified with technologies such as ActiveX and CORBA (Common Object Request Broker Architecture).

- Compilers and linkers have long provided the automation of source code into executable code. As architectures begin to use heterogeneous components, platforms and languages, the complexity of creating, controlling, and maintaining large-scale component networks introduces new needs for configuration control and building management automation. However, today's environments do not support automation to the fullest extent possible. For example, the automated construction of test cases from use case and scenario descriptions has yet to evolve to support anything but the most mundane examples, such as unit test scenarios.

- The primary reason for round-trip engineering is to allow the freedom to modify software engineering data sources. This configuration controls all the technical aspects

Automated production -►

★Traceability links

★Advanced engineering (generation of sources from models) Reverse engineering (generation of models from sources)

★Advanced engineering (generation of sources from models) Reverse engineering (generation of models from sources)

Figure 11.2

Portability between platforms and network topologies

Figure 4.2. Round-trip engineering artifacts are critical to maintaining a consistent and error-free representation of the evolving product. However, it is not necessary to have two-way transitions in all cases. For example, although we should be able to create test cases for defined scenarios for a

given logical set of objects, we cannot reverse engineer objects from test cases alone. Likewise, reverse engineering poorly constructed legacy source code into an object-oriented design model can backfire.

Translation from one data source to another may not provide 100% integrity. For example, translating design models into C ++ source code can provide only the structural and declarative aspects of the source code representation. It may be necessary to develop code components that are specific to certain attributes or methods of the object.

11.5 CHANGE OF MANAGEMENT

Managing change is so fundamental processes as a schedule. Tracking changes in technical artifacts is critical to understanding true trends in technical progress and quality trends towards delivering an acceptable final product or provisional version. On conventional processes, basic configuration management techniques for technical artifacts were predominantly a late life cycle activity. in amodern process—Where the artifacts of requirements set, design and implementation are captured in rigorous notation early in the lifecycle and evolve across multiple generations - change management has become critical to all stages and nearly all activities.

➔ **Software Change Orders**

- The atomic unit of work of software that is authorized to create, modify, or obsolete components within a configuration baseline is called a Software Change Order (SCO). Software change orders are a key mechanism for dividing, allocating and scheduling software work against an established software base and for evaluating progress and quality. The SCO example shown in Figure 4.3 is a good starting point for describing a set of change primitives. Shows the level of detail required to achieve the metrics and change management rigor required for amodernsoftware processes. By automating data entry and keeping change logs online, you can also automate the change management bureaucracy associated with metric reporting tasks.

- The level at which a SCO is written is always a problem. What is a discreet change? Is it a change in a program unit or component, file or subsystem? Is it a new feature, a bug fix, or a performance improvement? In most designs, the atomic unit of the OCS tends to be easily accepted. In general, a SCO should be written in a single component so that it can be easily assigned to a single individual. If the resolution requires two people on two different teams, two separate SCOs must be written.

The basic fields of the SCO are title, description, metrics, resolution, evaluation and disposition.

• Qualification. The title is suggested by the creator and finalized once accepted by the configuration control board (CCB). This field must

include a reference to a report on an external software problem if the change was initiated by an external person (such as a user).

- Description. The problem description includes the name of the creator, the date of origin, the SCO identifier assigned by the CCB, and the identifiers of the relevant version of its supporting software. The textual description of the problem should provide as much detail as possible, along with attached code excerpts, showing snapshots, error messages, and any other information that can help isolate the problem or describe the necessary change.

- Metrics. The metrics collected for each SCO are important for planning, scheduling and evaluating quality improvement. The change categories are Type 0 (Critical Error), Type 1 (Error), Type 2 (Enhancement), Type 3 (New Feature), and Type 4 (Other), as described later in this section. After the SCO is accepted, initial estimates are made of the extent of the break and the effort required to resolve the problem.

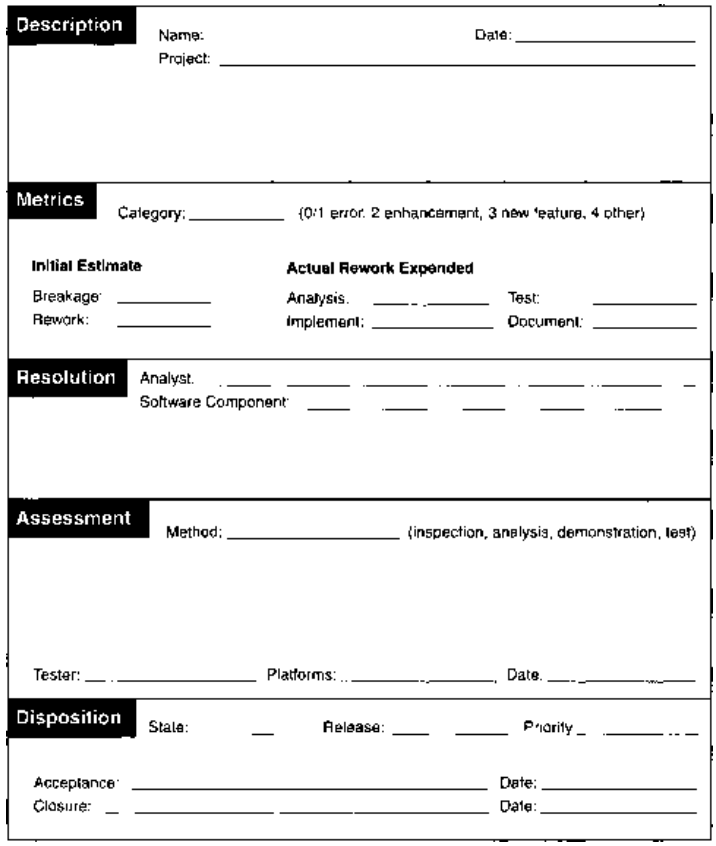


Figure 4.3. The primitive components of a software change order

The break element quantifies the volume of the change and the rework element quantifies the complexity of the change. Once resolved, the actual break is noted and the actual reprocessing effort is further processed. The analysis element identifies the number of staff hours spent understanding the requested change (recreating, isolating and debugging the problem if the change is type 0 or 1; analysis and prototyping of alternative solutions if it is type 2 or 3) . The implementation element identifies the staff hours required to design and implement the resolution. The test element

identifies hours spent testing resolution, and the document element identifies any effort invested in updating other artifacts, such as the manual of the user or version description. Break quantifies the scope of the change and can be defined in SLOC units, function points, files, components or classes. In the case of SLOC, a source file comparison program that quantifies the differences can provide a simple estimate of the break. In general, the accuracy of break numbers is relatively irrelevant. Changes between 0 and 100 lines must be accurate with the nearest 10, changes between 100 and 1000 with the nearest 100, and so on. the accuracy of the break numbers is relatively irrelevant. Changes between 0 and 100 lines must be accurate with the nearest 10, changes between 100 and 1000 with the nearest 100, and so on. the accuracy of the break numbers is relatively irrelevant. Changes between 0 and 100 lines must be accurate with the nearest 10, changes between 100 and 1000 with the nearest 100, and so on.

• Resolution. This field includes the name of the person responsible for implementing the change, the changed components, the actual metrics, and a description of the change. Although the fidelity level of components with which a project tracks change references can be customized, in general, the lowest level of component references should remain roughly at the level assigned to an individual. For example, a "component" assigned to a computer is not a sufficiently detailed reference.

• Evaluation. This field describes the evaluation technique such as inspection, analysis, demonstration or test. Where applicable, you should also refer to all existing test cases and newly run test cases and you should identify all the different test configurations, such as platforms, topologies and compilers.

• Layout. The CCB assigns one of the following statuses to the SCO:

• Proposal: written, awaiting review by the CCB

• Accepted: CCB approved for termination

• Rejected: closed, well founded, as not a problem, duplicated, obsolete change, solved by another SCO

• Archived: accepted but postponed to later publication

• In Progress: actively assigned and resolved by the development organization.

• Under evaluation: resolved by the development organization; be evaluated by a testing organization

• Closed: completely resolved, with the participation of all members of the CCB

The CCB can also assign a priority and release identifier to guide prioritization and organization of concurrent development activities.

11.6 BASIC CONFIGURATION

- A configuration baseline is a named collection of software components and supporting documentation that is subject to change management and is updated, maintained, tested, classified, and out of date as a unit. With complex configuration management systems, there are many desirable project and domain specific standards.

- Generally, there are two types of baselines: external product versions and internal test versions. A configuration baseline is a collection of named components that is treated as a unit. It is formally controlled because it is a packaged exchange between groups. For example, the development organization can publish a configuration baseline to the test organization or even itself. A project can release a configuration baseline to the user community for beta testing.

- In general, three basic emission levels are required for most systems: main, minor and interim. Each level corresponds to a numbered identifier such as NMX, where N is the major version number, M is the minor version number, and X is the provisional version identifier. A major version represents a new generation of the product or project, while a minor version represents the same base product but with improved features, performance, or quality. Major and minor releases are meant to be persistent and supported external product releases for a period of time. A provisional version corresponds to a development configuration that must be transient. The shorter its life cycle, the better. Figure 4.

- Once the software is placed on a controlled baseline, all changes are tracked. A distinction must be made for the cause of a change. The modification categories are as follows:

  ★**Type 0:** critical defects, which are defects that are almost always fixed before any external release. In general, these types of changes represent surprising aspects that impact the usability of the software in its critical use cases.

  ★**Type 1:**an error or defect that does not affect the usefulness of the system or that can be fixed Such errors tend to be related to annoyances in critical use cases or serious defects in secondary use cases that have a low probability of occurring.

  ★**Type 2:** A change that is an improvement rather than a response to a defect. Type 3: A change that is needed to update the environment. Type 4: Changes that are not accepted by the other categories.

  ★**Type 3:** A change required by the environment update.

  ★**Type 4:** Changes that are not accepted by the other categories.

➔ **Change Management - Configuration Dashboard (CCB)**

A CCB is a team of people who act as decision-making authority over the content of the configuration baselines. A CCB includes:

1. Software administrators

2. Software architecture managers

3. Software development managers

4. Software Evaluation Managers

5. Other interested parties who are integral to maintaining the controlled software distribution system?

Infrastructure I: The organization's infrastructure provides the organization's capital assets, including two key artifacts: politics and the environment.

1. Organization policy:

A policy captures the standards for the project's software development processes. The organization's policy is usually packaged as a manual that defines life cycles and process primitives, such as

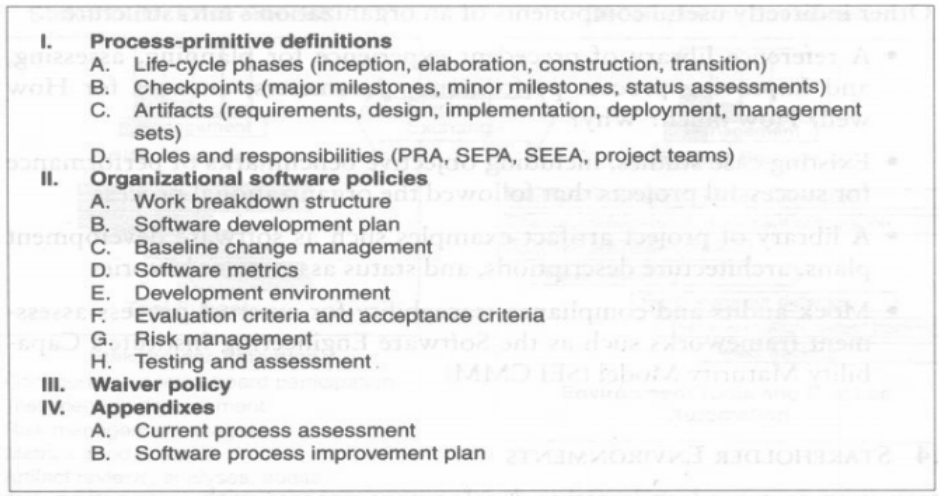
★Important milestones

★Intermediate artifacts

★Engineering repository

★Metric

★Roles and responsibilities



I. **Process-primitive definitions**
   A. Life-cycle phases (inception, elaboration, construction, transition)
   B. Checkpoints (major milestones, minor milestones, status assessments)
   C. Artifacts (requirements, design, implementation, deployment, management sets)
   D. Roles and responsibilities (PRA, SEPA, SEEA, project teams)

II. **Organizational software policies**
   A. Work breakdown structure
   B. Software development plan
   C. Baseline change management
   D. Software metrics
   E. Development environment
   F. Evaluation criteria and acceptance criteria
   G. Risk management
   H. Testing and assessment

III. **Waiver policy**

IV. **Appendixes**
   A. Current process assessment
   B. Software process improvement plan

Figure 11.4

Infrastructure II:

Environment of the organization

The environment that acquires an inventory of tools that are building blocks from which project environments can be configured efficiently and economically

Stakeholder environment

Many large-scale projects include people in external organizations representing other stakeholders involved in the development process which they may include.

★ Contract supervisors of procurement agencies

★ End-user technical support staff

★ External maintenance companies

★ Independent verification and validation contractors

★ Representatives of regulatory agencies and others.

These stakeholder representatives also need to access development resources so they can add value to the overall effort. These interested parties will be accessible via the Internet. An online environment accessible to external stakeholders will allow them to participate in the process as follows. Accept and use executable increments for practical evaluation. Use the same tools, data and online reports used by the development organization to manage and monitor the project Avoid excessive travel, document exchange delays, format translations, shipping costs * and other overhead costs

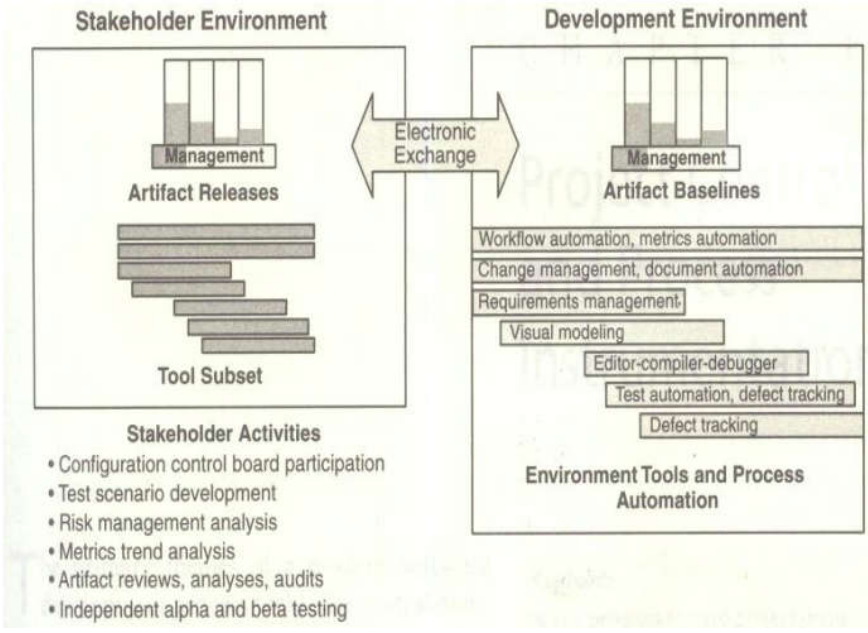


Figure 11.5

11.7 SUMMARY

- It will only exist from the beginning to the end of the project. All project team members come from different parts of the organization.
- Everyone will have a temporary assignment to the project.
- Organizations must evolve to survive. Those who evolve most successfully will have a competitive advantage. Without evolution, the organization will succumb to the wave of creative destruction of the economy.

11.8 BIBLIOGRAPHY

1. https://mymanagementguide.com/basics/project-team-organization-project-team-definition-responssibility-and-roles-and-project-team-organization-chart/
2. https://www.geeksforgeeks.org/project-organizations-and-their-responssibility/
3. https://www.villanovau.com/resources/project-management/project-team-roles-and-responssibility/
4. https://www.gristprojectmanagement.us/software-3/project-organizations.html

11.9 QUESTIONS AT THE END OF THE CHAPTER

1. What are the three different levels of processes in process automation?
2. Explain process automation. Give the benefits.
3. What are the concerns associated with each workflow?
4. Explain the 4 phases of project management.
5. Explain the different forms of the project environment.
6. Explain change management.
7. Explain the baseline of the configuration.

Unit V

12

PROCESS CONTROL AND PROCESS INSTRUMENTATION

Unit Structure

12.0 OBJECTIVES

At the end of this unit, the student will be able to

- Illustrate the need of seven core metrics required for managing project
- Differentiate between the management indicators with quality indicators
- Explain the requirement of life cycle expectations
- Use the pragmatic software metric for developing the project
- Conclude the requirement of metric automation for project management

12.1 INTRODUCTION

1. The main motto of a modern software development process tackle the central management issues of complex software is as follows

1.1 Getting the design right by focusing on the architecture first .

1.2 Managing risk through iterative development.

1.3 Reducing the complexity with component based techniques.

1.4 Making software progress and quality tangible through proper channel of change management.

1.5 Round-trip engineering and integrated environment should be used to automate the overhead and book keeping activities.

1.6 One of the major issues with the conventional software process, is that it is very difficult to manage what cannot be measured objectively.

1.7 Software metrics instrument the activities and products of the of the software development /integration process. Any software process whose metrics are calculated by manual procedures and human-intensive activities will have success for a limited period of time.

1.8 But in a modern development process, the most important software metrics are simple and highly focused on measuring the perspectives of the product and project when they are changing.

2. The progress toward project goals and the quality of software products must be measurable throughout the software development cycle.

3. Metrics values provide an important perspective for managing the process. Metrics trends provide another.

4. The most useful metrics are extracted directly from the evolving artifacts.

5. Objective analysis and automated data collection are crucial to the success of any metrics program. Subjective assessments and manual collection techniques are likely to fail.

6. The quality of software products and the progress made toward project goals must be measurable throughout the software development cycle.

7. The goals of software metrics are to provide the development team and the management team with the following

7.1 An accurate assessment of progress to date.

7.2 Insight in to the quality of the evolving software product.

7.3 A basis for estimating the cost and schedule for completing the product with increasing accuracy over time.

12.2 THE SEVEN CORE METRICS

1. Many different metrics may be of any value in managing a modern process. There are seven core metrics that can be used on all software projects,

2. Three are management indicators and four are quality indicators, whereas management indicators consist of a)Work and Progress(work performed over time),b)Budgeted cost and expenditures(cost incurred over time), c)Staffing and team dynamics(personnel changes over time). Quality Indicators consist of a)Change traffic and stability(change traffic over time),b)Breakage and modularity(average breakage per change over time),c)Rework and adaptability (average rework per change over time),d)Mean time between failures(MTBF) and maturity(defect rate over time).

3. Table 1 given below describes the software metrics. Each metric has two dimensions a static value used as an objective and the dynamic trend used to manage the achievement of that objective.

4. Metrics value provide one dimension of insight, metrics trends provide a more important perspective for managing the process.

5. Iterative development is about managing change, and measuring change is the most important aspect of the metrics program.

6. Absolute values of productivity and quality improvement are secondary issues until the fundamental goal of management has been achieved: predictable cost and schedule performance for a given level of quality.

Sr.No	Metric	Purpose	Objectives
1	Work and Progress	Iteration planning, plan Vs actuals, management indicator	SLOC, function points, object points, scenarios, test cases, SCOs
2	Budgeted cost and expenditures	Financial insight, plan Vs actuals, management indicator	Cost per month, full-time staff per month, percentage of budget expended
3	Staffing and team dynamics	Resource plan vs, actuals, hiring rate, attrition or loss rate	People per month added, people per month leaving
4	Change traffic and stability	Iteration planning, management indicator of schedule convergence	SCOs opened vs SCOs closed by type (0,1,2,3,4) by release / component/subsystem

5	Breakage and modularity	Convergence, software scrap, quality indicator	Reworked SLOC per change, by type (0,1,2,3,4) by release /component subsystem
6	Rework and adaptability	Convergence, software rework, quality	Average hours per change, by type (0,1,2,3,4) by release /component subsystem
7	MTBF(Mean time between failure) and Maturity	Test Coverage/adequacy, robustness for use, quality indicator	Failure counts, test hours until failure by release/component subsystem

7. The seven core metrics can be used in numerous ways to help manage projects and organizations. In an iterative development project or an organization structured around a software line of business, the historical values of previous iterations and projects provide precedent data for planning subsequent iterations and projects.

8. Consequently, when in an organization, the metrics is collected by project manager in order to predict the cost, schedule or quality performance of future work activities.

9. The seven core metrics are based on common sense and field experience with both successful and unsuccessful metrics programs. Their attributes include the following

9.1 They are simple, objective, easy to collect, easy to interpret and hard to misinterpret.

9.2 Collection can be automated and nonintrusive

9.3 They provide for consistent assessments throughout the life cycle and are derived from the evolving product baselines rather than from a subjective assessment.

9.4 They are useful to both management and engineering personnel for communicating progress and quality in a consistent format

.

9.5 Their fidelity improves across the life cycle.

10. Metrics applied to the engineering stage will be far less accurate than those applied to the production stage. Therefore, the prescribed metrics are tailored to the production stage, when the cost risk is high and management value is leveraged.

12.3 MANAGEMENT INDICATORS

1. There are three fundamental sets of management metrics: technical progress, financial status and staffing progress. By examining these perspectives, management can generally assess whether a project is on budget and on schedule.

2. Most managers know their resource expenditures in terms of costs and schedule.

3. The problem is to assess how much technical progress has been made. Conventional projects whose intermediate products were all paper documents relied on subjective assessments of technical progress or measured the number of documents completed.

4. The management indicators recommended here include standard financial status based on an earned value system, objective technical progress metrics tailored to the primary measurement criteria for each major team of the organization and staffing metrics that provide insight in to team dynamics.

1. Work and Progress

1. The various activities of an iterative development project can be measured by defining a planned estimate of the work in an objective measure, then tracking progress against that plan as shown in given figure below.

2. Each major organizational team should have at least one primary progress perspective that is measured against.

3. For the standard teams of project management, the default perspectives of this metric would be as follows

1 Use case is demonstrated by Software architecture teams

2. Software development team: SLOC under baseline change management, SCOs closed

3. Software assessment team: SCOs opened, test hours executed, evaluation criteria met.

4. Software management team: milestones completed.

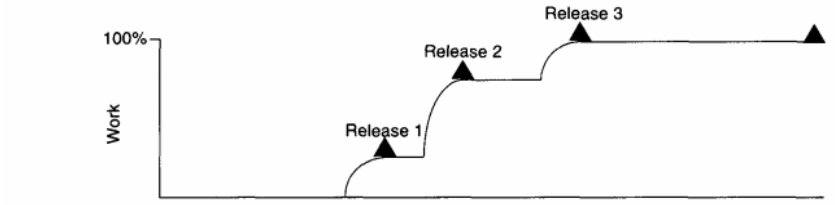


Fig 1 Expected progress for a typical project with three major releases

2. Budgeted Cost and Expenditures

1. To maintain management control, measuring cost expenditures over the project life cycle is always necessary. With an iterative development process, it is important to plan the near term activities in detail and leave the long term activities as rough estimates to be refined as the current iteration is winding down and planning for the next iteration becomes crucial.

2. Tracking financial progress usually takes an organization-specific format. One common approach to financial performance measurement is use of an earned value system, which provides highly detailed cost and schedule insight.

3. Its major weakness for software projects has traditionally been the inability to assess the technical progress accurately.

4. The other core metrics provide a framework for detailed and realistic quantifiable back up data to plan and track against, especially in the production stage of a software project, when the cost and schedule expenditures are highest.

5. Modern software processes are measured for financial performance through an earned value approach.

6. The basic parameters of an earned value system, usually expressed in units of dollars are as follows:

6.1 Expenditure plan- The planned spending profile for a project over its planned schedule. This plan generally tracks the staffing profile for most software projects.

6.2 Actual progress- In a healthy project, the actual progress tracks the technical issues of the project for being complete the project according to the planned project timeline.

6.3 Actual Cost- This calculates the total cost incurred for implementing an healthy project.

6.4 Earned Value: The Value that represents the planned cost of the actual progress.

6.5 Cost Variance- The difference between the actual cost and the earned value of the project. Positive values occur when there is over budget situation and negative values occur when there is under budget situation.

6.6 Scheduled Variance- The difference between the planned schedule cost and the actual schedule cost. Positive values signed to behind-schedule situations and negative values signed to ahead of schedule situations.

7. The main purpose of the other core metrics is to provide management and engineering teams with a more objective approach for assessing actual progress with greater accuracy, but in earned value analysis, the actual progress of project is assessed subjectively.

8. As managers know exactly how much cost is incurred and how much schedule they have utilized for implementing the project in order to accurately measure the budget estimated for the underlying project.

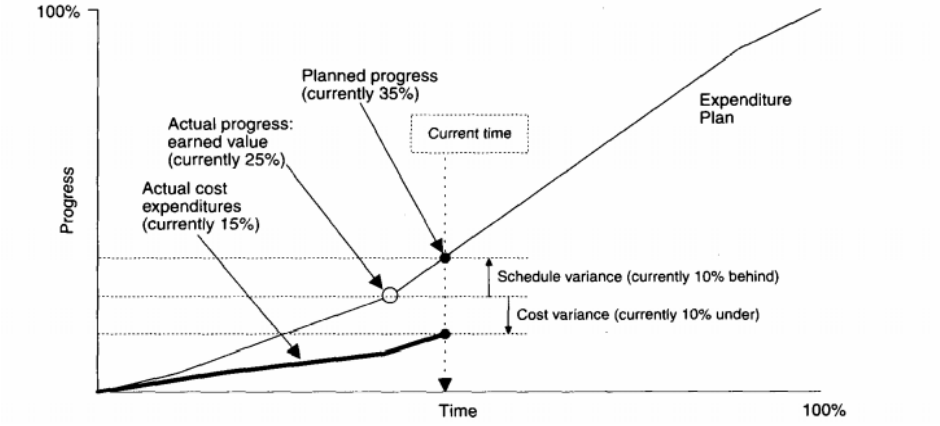


Fig 2 Basic parameter of an earned value system

9. To better understand some of the strengths and weaknesses of an earned value system, can be explained through considering the development of book. Actual progress could easily be tracked by the current state of each chapter, weight-averaged by the number of pages planned for that chapter. Here status of each part progress is given in terms of percentage which in turn is nothing but an earned value of the project.

Sr.No	Earned Value	Status
1	0 to 50%	Content incomplete
2	50%	Author has completed first draft text and art
3	65%	Initial text baseline, initial text editing complete
4	75%	Reviewable baseline, text and art editing complete
5	80%	Updated baseline, cross chapter consistency model
6	90%	Reviewed baseline, author has incorporated external reviewer comments
7	100%	Final edit, editor has completed a final cleanup pass

10. The "percent complete" assessments were assigned subjectively based on experience of writing complex documents. So by taking a weighted average of these percentages listed above in table ,(0+50+65+75+80+90+100)/7=65 %. So overall progress of completing a book development has done only 65% and thus 65% is the earned value of the project.

11. This example provides a good framework for establishing an objective basis, developing a suitable work breakdown structure and planning with appropriate fidelity.

3 Staffing and Team Dynamics

1. An iterative development should start with a small team until the risks in the requirements and architecture have been suitably resolved.

2. Depending on the overlap of iterations and other project-specific circumstances, staffing can vary.

3. For a commercial product development, the sizes of the maintenance and development teams may be the same.

4. When long-lived, continuously improved products are involved, maintenance is just continuous construction of new and better releases.

5. Tracking actual versus planned staffing is a necessary and well-understood management metric. Increase in staff can slow overall project progress as new people consume the productive time of existing people in coming up to speed.

6. Low loss of good people is a sign of success. An increase in unplanned attrition namely, people leaving a project prematurely is one of the most glaring indicators that a project is destined for trouble.

7. The causes of such attrition can vary, but they are usually personnel dissatisfaction with management methods, lack of teamwork or probability of failure in meeting the planned objectives.

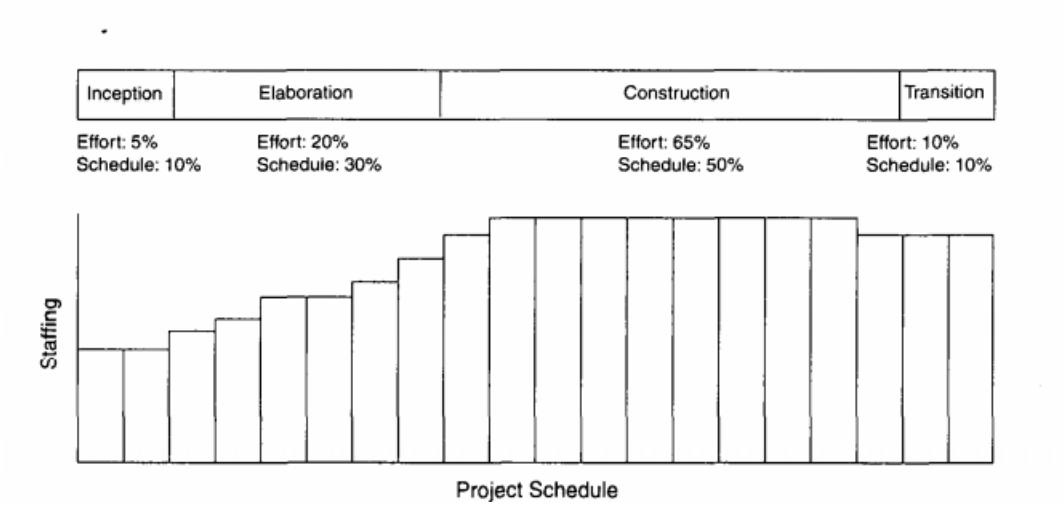


Fig 3 Typical Staffing profile

12.4 QUALITY INDICATORS

The four quality indicators are based primarily on the measurement of software change across evolving baselines of engineering data(such as design models and source code).

1.1 Change traffic and stability

1. Change traffic is defined as the number of software change orders opened and closed over the life cycle. So change traffic is one specific indicator for progress and quality.

2. This metric can be collected by change type, by release, by team, by components, by subsystem and so forth.

3. By coupling with work and progress metrics, it provides insight in to the stability of the software. So stability is defined as the relationship between opened versus closed SCOs.

4. The primary value of this metric and an indicator of how well the process is performing is better provided by the change traffic which keeps tracks on schedule of the project.

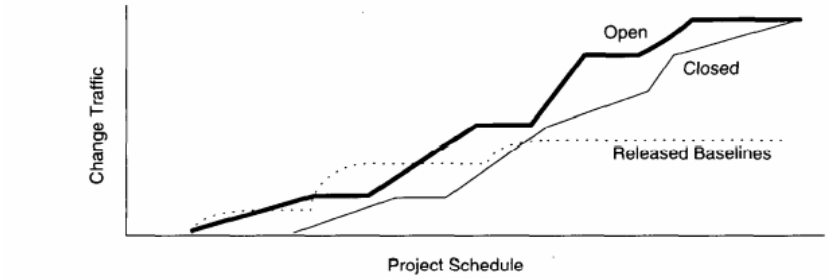


Fig 4 Stability expectation over a healthy project's Life Cycle.

1.2 Breakage and Modularity

1. Breakage is defined as the average extent of change, which is the amount of software baseline that needs rework like function points, components, subsystem, files etc.

2. Whereas Modularity is defined as the average breakage trend over time. For a healthy project, the trend expectation is decreasing or stable as shown in figure below.

3. This indicator provides insight in to the malicious character of software change. In a mature iterative development process, earlier changes are expected to result in more scrap than later changes.

4. Breakage trends that are increasing with time clearly indicate that product maintainability is suspect.

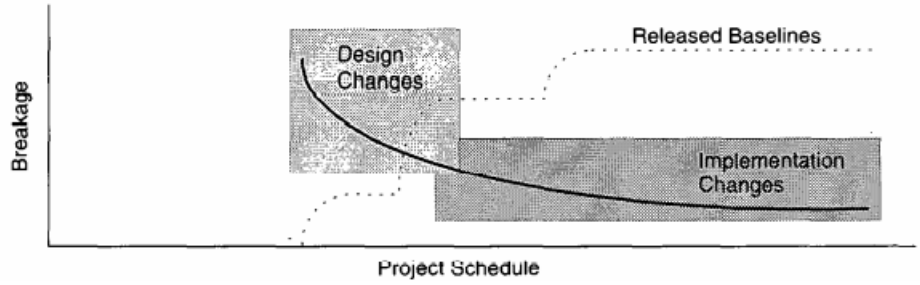


Fig 5 Modularity expectation over a healthy project's Life cycle

1.3 Rework and Adaptability

1. Rework is defined as the average cost of change, which is the effort to analyze, resolve and retest all changes to software baselines.

2. Adaptability is defined as the rework trend over time. For a healthy project, the trend expectation is decreasing or stable as shown in figure below in which X axis belongs to project schedule and Y axis belongs to Rework

3. Not all changes given to project are solved same time, some solved in hours or some solved in weeks. So this metric provides insight in to rework measurement.

4. Architectural changes in the project requires rework at earlier stage of project whereas rework is not required in the implementation phase of the project.

5. Rework trends that are increasing with time clearly indicate that product maintainability is suspect.

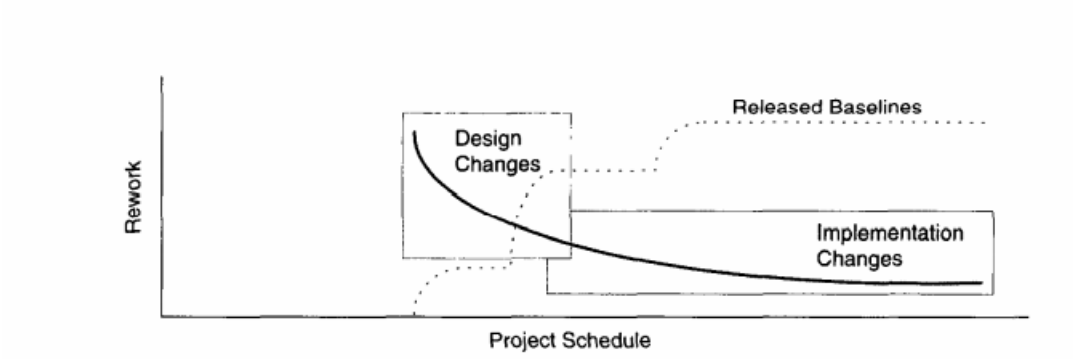


Fig 6 Adaptability expectation over a healthy project's Life cycle

1.4 MTBF (Mean Time Between Failures) is the average usage time between software faults.

2. In rough terms, MTBF is computed by dividing the test hours by the number of type 0 and type 1 SCOs.

3. Whereas maturity is defined as the MTBF trend over time as shown in figure below where X axis belongs to project schedule and Y axis belongs to MTBF.

4. Conventional testing approaches for monolithic software programs focused on achieving complete test coverage of every line of code, branch and so forth.

5. Systems of components are more efficiently tested by using statistical techniques. So maturity metrics measure statistics over usage time of component rather than product coverage.

6. Software errors can be categorized in to two types as Bohr bugs and Heisen bugs respectively, Bohr bugs occur during coding time of the project or when any changes are done to the component independently, where asHeisen bug occurs during the designing of the project hence they are called as design errors.

7. To provide adequate test coverage and resolve the statistically significant Heisen bugs are tested using randomized usage scenarios.

8. Conventional software programs typically contained only Bohr-bugs, whereas in distributed system with numerous interoperating components executing across a network of processors are vulnerable to Heisen-bugs which is very hard to detect and resolve.

9. The better way to mature a software product is to test an randomized usage scenarios early in the life cycle, to optimize coverage across the reliability-critical components.

10. Meaningful insight in to product maturity can be gained by maximizing test time through regression test, stress testing, randomized statistical testing etc. This testing approach at early stage of project life cycle could also be used for monitoring performance improvements and measuring reliability.

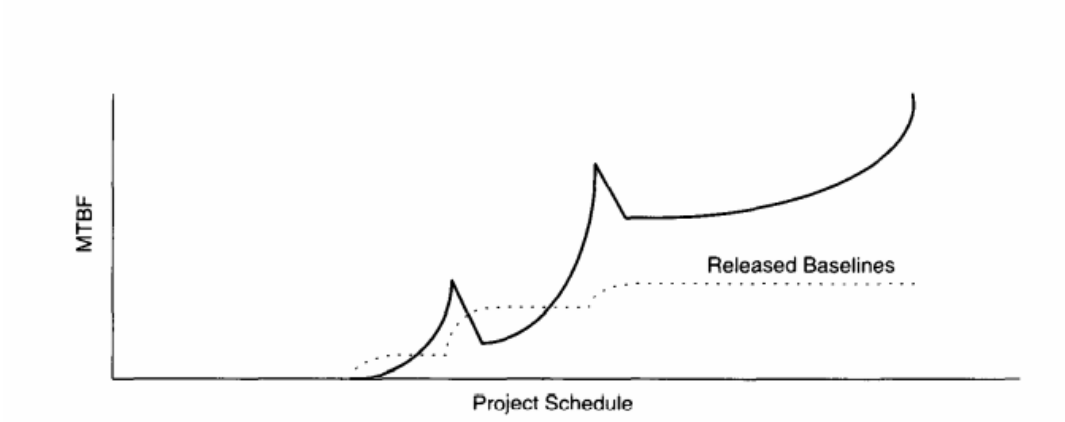


Fig 7 Maturity expectation over a healthy project's lifecycle.

12.5 LIFE CYCLE EXPECTATIONS

1. The quality indicators are derived from the evolving product rather than from the artifacts.

2. They provide insight in to the waste generated by the process. Scrap and rework metrics are a standard measurement perspective of most manufacturing processes.

3. They recognize the inherently dynamic nature of an iterative development process. Rather than focus on the value, they explicitly concentrate on the trends or changes with respect to time.

4. The combination of insight from the current value and the current trend provides tangible indicators for management action.

5. The actual values of these metrics can vary widely across projects, organizations and domains. The relative trends across the project phases, however should follow the general pattern as shown in table given below.

6. A mature development organization should be able to describe metrics targets that are much more definitive and precise for its line of business and specific processes.

Table 1 The default pattern of life-cycle metrics evolution

METRIC	INCEPTION	ELABORATION	CONSTRUCTION	TRANSITION
Progress	5%	25%	90%	100%
Architecture	30%	90%	100%	100%
Applications	<5%	20%	85%	100%
Expenditures	Low	Moderate	High	High
Effort	5%	25%	90%	100%
Schedule	10%	40%	90%	100%
Staffing	Small team	Ramp up	Steady	Varying
Stability	Volatile	Moderate	Moderate	Stable
Architecture	Volatile	Moderate	Stable	Stable
Applications	Volatile	Volatile	Moderate	Stable
Modularity	50%–100%	25%–50%	<25%	5%–10%
Architecture	>50%	>50%	<15%	<5%
Applications	>80%	>80%	<25%	<10%
Adaptability	Varying	Varying	Benign	Benign
Architecture	Varying	Moderate	Benign	Benign
Applications	Varying	Varying	Moderate	Benign
Maturity	Prototype	Fragile	Usable	Robust
Architecture	Prototype	Usable	Robust	Robust
Applications	Prototype	Fragile	Usable	Robust

12.6 PRAGMATIC SOFTWARE METRICS

1. Measuring is useful, but it doesn't do any thing for the decision makers. It only provides data to help them ask the right questions, understand the context and make objective decisions.

2. Because of the highly dynamic nature of software projects, these measures must be available at any time, tailorable to various subsets of the evolving product and along with maintained first and second versions of the product.

3. The basic characteristics of a good metric are as follows

3.1 Meaningful to the customer, manager and performer- If any one of these stakeholder does not see the metric as meaningful, it will not be used. Customers will accept metrics that are demonstrated to be meaningful by the developer.

3.2 Demonstrates quantifiable correlation between process perturbations and business performance- The only real organizational goals and objectives are financial:cost reduction, revenue increase and margin increase.

3.3 Objective and unambiguously defined- Ambiquity is minimized through well understood units of measurement(such as staff-month, function point etc) which are surprisingly hard to define precisely in the software engineering world.

3.4 Displays trends-This an important characteristic. Understanding the change in a metric's value with respect to time, subsequent projects, subsequent-releases and so forth is an extremely important perspective, especially for today's iterative development models.

3.5 Natural by product of the process- The metric does not introduce new artifacts or overhead activities, it is derived directly from the main-stream engineering and management workflows.

3.6 Supported by automation- Experience has demonstrated that the most successful metrics are those that are collected and reported by automated tools, in part because software tools require rigorous definitions of the data they process.

4. Metrics usually display effects, the causes require synthesis of multiple perspectives and reasoning. For eg reasoning is still required to interpret the following situations correctly

4.1 A low number of change requests to a software baseline may mean that the software is mature and error-free or it may mean that the test team is on vacation.

4.2 A software change order that has been open for a long time may mean that the problem was simple to diagnose and the solution required substantial rework, or it may mean that a problem was very time-consuming to diagnose and the solution required a simple change to a single line of code.

4.3 A large increase in personnel in a given month may cause progress to increase proportionally if they are trained who are productive from the outset.

12.7 METRICS AUTOMATION

1. There are many opportunities to automate the project control activities of a software project.

2. For managing against a plan, a software project control panel (SPCP) that maintains an online version of the status of evolving artifacts provides a key advantage.

3. The idea is to provide a display panel that integrates data from multiple sources to show the current status of some aspect of the project.

4. For example, The software project manager would want to see a display with overall project values, a test manager may want to see a display focused on metrics specific to an upcoming beta release and development managers may be interested only in data concerning the subsystems and components for which they are responsible. This panel acts as a dashboard which shows the analysis of a project at one glance.

5. The panel can support standard features such as warning lights, thresholds, variable scales, digital formats and analog formats to present an overview of the current situation.

6. It can also provide extensive capability for detailed situation analysis. This automation support can improve management insight in to progress and quality trends and improve the acceptance of metrics by the engineering team.

7. To implement a complete SPCP, it is compulsory to define and develop the following

7.1 Metrics primitives: Indicators, trends, comparisons and progressions

7.2 A GUI: GUI support for a software project manager role and flexibility to support other roles.

7.3 Metrics collection agents- Data extraction from the environment tools that maintain the engineering notations for the various artifacts sets.

7.4 Metrics data management server- Data management support for populating the metric displays of the GUI and storing the data extracted by the agents.

7.5 Metric definitions- Actual metrics presentations for requirement progress(requirement set artifacts, design progress(design set artifacts), implementation progress(implementation set artifacts), assessment progress(deployment set artifacts) and other progress dimensions (management artifacts).

7.6 Actors- Typically, the monitor and the administrator

8. For every role, there is a specific panel configuration and scope of data presented. Each role performs the same general use cases, but with a different focus.

8.1 Monitor- defines panel layouts from existing mechanisms, graphical objects, and linkages to project data, queries data to be displayed at different levels of abstraction.

8.2 Administrator- Installs the system, defines new mechanisms, graphical objects and linkages handles archiving functions, defining composition

and decomposition structures for displaying multiple levels of abstraction.

9. A panel typically contains a number of graphical object is positioned in a particular geometric layout. A metric shown in a graphical object is labelled with the metric type, the summary level, and the instance name(such as lines of code, subsystem, server1).

10. Metrics can be displayed in two modes 1)Value referring to a given point in time 2)Graph referring to multiple and consecutive points in time.

11. Metrics can be displayed with or without control values. A control value is an existing expectation, either absolute or relative that is used for comparison with a dynamically changing metric. For eg the plan for a given progress metric is a control value for comparing the actuals of that metric.

12. Indicators may display data in formats that are binary(such as black and white), tertiary(such as red, yellow and green), digital (integer or float) or some other enumerated type (for eg Sun-Sat).

13. A trend graph presents values over time and permits upper and lower thresholds to be defined. Crossing a threshold could be linked to an associated indicator to depict a noticeable state change from green to red or vice versa.

14. Metric information can be summarized following a user-defined, linear structure (for eg lines of code can be summarized by unit subsystem and project)

15. Figure given below illustrates a simple example of an SPCP for a project. In this case, the software project manager role has defined a top-level display with four graphical objects

15.1 Project activity status- The graphical object in the upper left provides an overview of the status of the top-level WBS elements. The seven elements could be coded red, yellow and green to reflect the current earned value status. For eg figure shown below are coded with white and shades of gray). Where each color represent different indication of project, green represent ahead of plan, yellow would indicate within 10% of plan, and red would identify elements that have a greater than 10% cost or schedule variance.

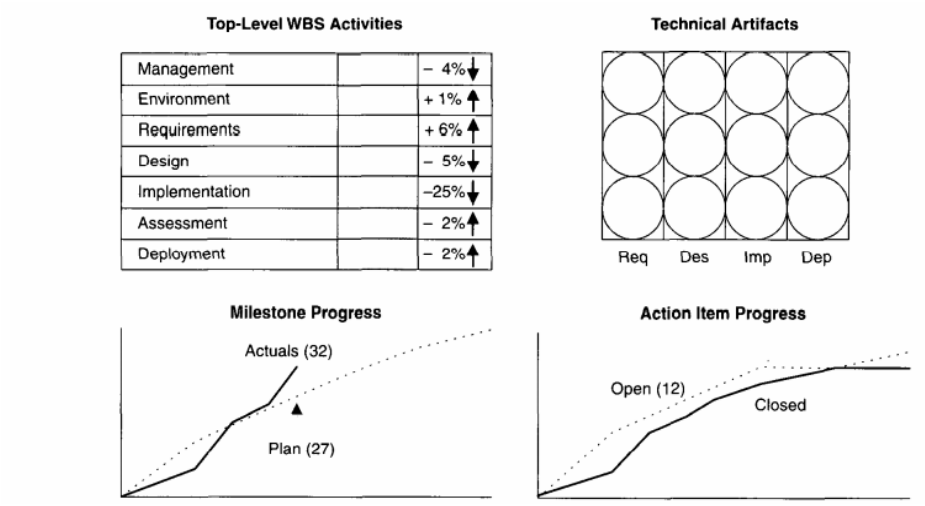


Fig 8 Example SPCP display of a top-level project situation

15.2 Technical artifact status- The graphical object in the upper right provides an overview of the status of the evolving technical artifacts. Req light represents the assessment of current state, Des light represent the design models, the Imp light represents the source code baseline, and the Dep light represents the test program.

15.3 Milestone progress- The graphical object in the lower left provides a progress assessment of the milestones achieved against plan and provides indicators of the current value.

15.4 Action item progress- The graphical in the lower right provides a different perspective of progress, showing the current number of open and closed issues.

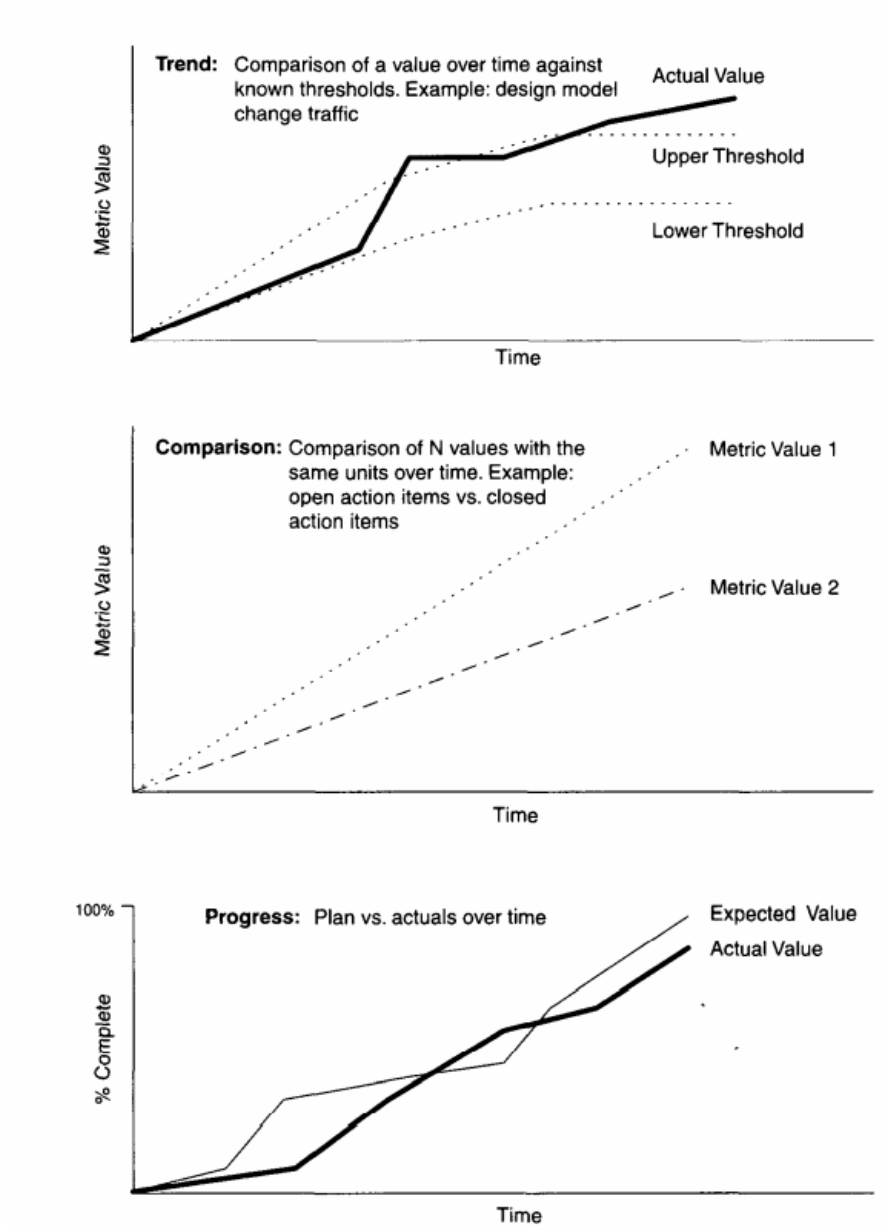


Fig 9 Example of the fundamental metric classes

16. The following top-level use case, which describes the basic operational concept for an SPCP, corresponds to a monitor interacting with the control panel.

16.1 start the SPCP- The SPCP starts and shows the most current information that was saved when the user last used the SPCP.

16.2 Select a panel preference- The user selects from a list of previously defined default panel preferences. The SPCP displays the preference selected.

16.3 Select a value or graph metric- The user selects whether the metric should be displayed for a given point in time or in graph as a trend.

16.4 Select to superimpose controls- The user points to a graphical object and requests that the control value for that metric and point in time be displayed.

16.5 Drill down to trend- The user points to a graphical object displaying a point in time and drills down to view the trend for the metric.

16.6 Drill down to point in time- The user points to a graphical object displaying a trend and drills down to view the values for the metric.

16.7 Drill down to lower levels of information- The user points to a graphical object displaying a point in time and drills down to view the next level of information.

16.8 Drill down to lower level of indicators- The user points to a graphical object displaying an indicator and drills down to view the breakdown of the next level of indicators.

12.8 SUMMARY

In this chapter we studied about seven core metrics The seven core metrics can be used in numerous ways to help manage projects and organizations. In an iterative development project or an organization structured around a software line of business, the historical values of previous iterations and projects provide precedent data for planning subsequent iterations and projects.

12.9 REFERENCES

[1] Software Engineering by Sommerville 8th Edition

12.10 QUESTIONS

Q1. Describe the different dimension of Management Indicators?
Q2. Illustrate with example different dimension of Quality Indicators?
Q3. List down the name of seven core metrics?
Q4. What is life cycle expectation?
Q5. Suppose a company want to develop a dash board for sales analysis for last 5 years. Justify your answer with metric automation?
Q6. What is pragmatic software metrics?

13

TAILORING THE PROCESS

Unit Structure

13.0 OBJECTIVES

At the end of this unit, a learner will be able to

- Describe the concepts of Scale as process discriminants.
- Illustrate need of process maturity for project development.
- Apply knowledge of Scale, process flexibility, Architectural Risk in handling real time project
- Differentiate between the process for handling the small scale and large scale projects.

13.1 INTRODUCTION

1. Software management efforts span a broad range of domains. While there are some universal themes and techniques, it is always necessary to tailor the process to the specific needs of the project at hand.
2. A commercial software tool developer with complete control of its investment profile will use a very different process from that of a

software integrator on contract to automate the security system for a nuclear power plant.

3. There is no doubt that a mature process and the effective software management approaches offer much greater value to the large scale software integrator than they do to the small scale tool developer.

4. Nevertheless relative to their business goals, the return on investment realized by better software management approaches is worth while for any software organizations.

5. The process framework must be configured to the specific characteristics of the project.

6. The scale of the project in particular team size drives the process configuration more than any other factor.

7. Other key factors include stakeholder relationships, process flexibility, process maturity, architectural risk, and domain experience.

8. While specific process implementations will vary, the spirit underlying the process is the same

13.2 PROCESS DISCRIMINANTS

1 To deal with the tailoring process, the project manager see two dimensions of the system as technical complexity and management complexity.

2.If the project occupies with these two dimensions then the formality of reviews, the quality control of artifacts, the priorities of concerns and numerous other process are governed to check the technical complexity, lower management complexity, higher management complexity and lower technical complexity.

3. A process framework is not a project specific process implementation with a well-defined recipe for success. To achieve a success in process implementation, judgement must be injected, and the methods, techniques culture, formality and organization must be tailored to the specific domain.

4. The major differences among project processes is organized around six process parameters, that is size of the project and the five parameters that affect the process exponent and hence the economies of scale in COCOMO II.

5.These are some of the critical dimensions that a software project manager must consider when tailoring a process framework to create a practical process implementation.

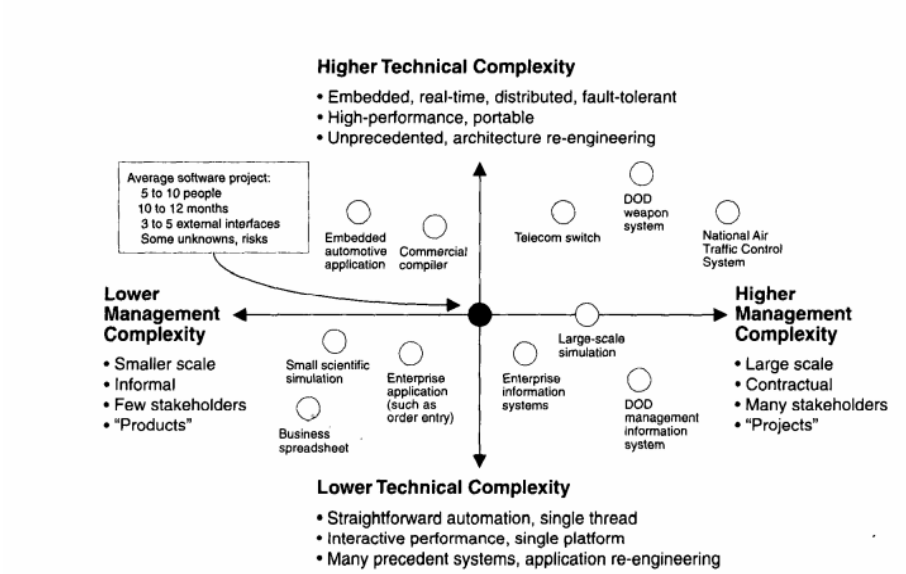


Fig 1 The two primary dimensions of process variability

1 Scale

1. The total scale of the software application, is the single most important factor in tailoring a software process framework to the specific needs of a project.

2. The parameters to be considered for measuring scale is, source lines of code, number of function points, number of use cases, and number of dollars.

3. The primary measure of scale is the size of the team, from the process tailoring point of view.

4. The diseconomies of scale can have a serious impact on achievement of the project objectives.

5. Many studies indicate that most people can best manage four to seven things at a time. For eg there are different management approaches needed to manage a team of 1(trivial), a team of 5(small), a team of 25(moderate), a team of 125(large), a team of 625(huge) and so on.

6. As team size grows, a new level of personnel management is introduced at rougly each factor by 5. This model can be used to describe some of the processes differences among projects of different sizes.

7. Trivial sized projects require almost no management overhead(planning, communication, coordination, progress assessment, review, administration).

8. Small projects comprises of 5 people in a team require very little amount of management overhead, but team leadership will always run

behind the objective of the project. Project milestones are easily planned, informally conducted and easily changed.

9. Moderate-sized projects comprises of 25 people in a team which require only moderate amount of management overhead, in which dedicated software project manager to synchronize team workflow and balance resources. Project milestones are formally planned and conducted and the impacts of changes are typically handled with properly. Process maturity is valuable. An environment can have a considerable impact on performance, but success can be achieved with certain key tools in place.

10. Large size projects comprises of 125 people in a team which require substantial management overhead, including a dedicated software project manager and several subproject managers to synchronize project-level and sub-project level workflows and to balance resources. Project milestones are formally planned and conducted and changes to milestone plans are expensive. Performance is highly dependent on the skills of key personnel, especially subproject managers and team leads.

11. Project performance is dependent on average people, for two reasons.

11.1 There are numerous variety of jobs in any large project, especially in the overhead workflows.

11.2 The probability of recruiting, maintaining and retaining a large number of exceptional people is small.

12. Process maturity is necessary, particularly the planning and controls aspects of managing project commitments, progress, and stakeholder expectations.

13. Huge projects team comprises of 625 people which require substantial management overhead, including multiple software project managers and many subproject managers to synchronize project-level and subproject-level workflows and to balance resources. Project milestones are very formally planned and conducted, and changes to milestone plans typically cause malicious replanning. Performance is highly dependent on the skills of key personnel, especially subproject managers and team leads.

14. Software process maturity and domain experience are mandatory to avoid risks, and ensure synchronization of expectation across numerous stakeholders. A mature highly integrated, common environment across the development teams is necessary to mange change, automate artifact production, maintain consistency among the evolving artifacts and improve the return on investment of common processes, common tools, common notation and common metrics.

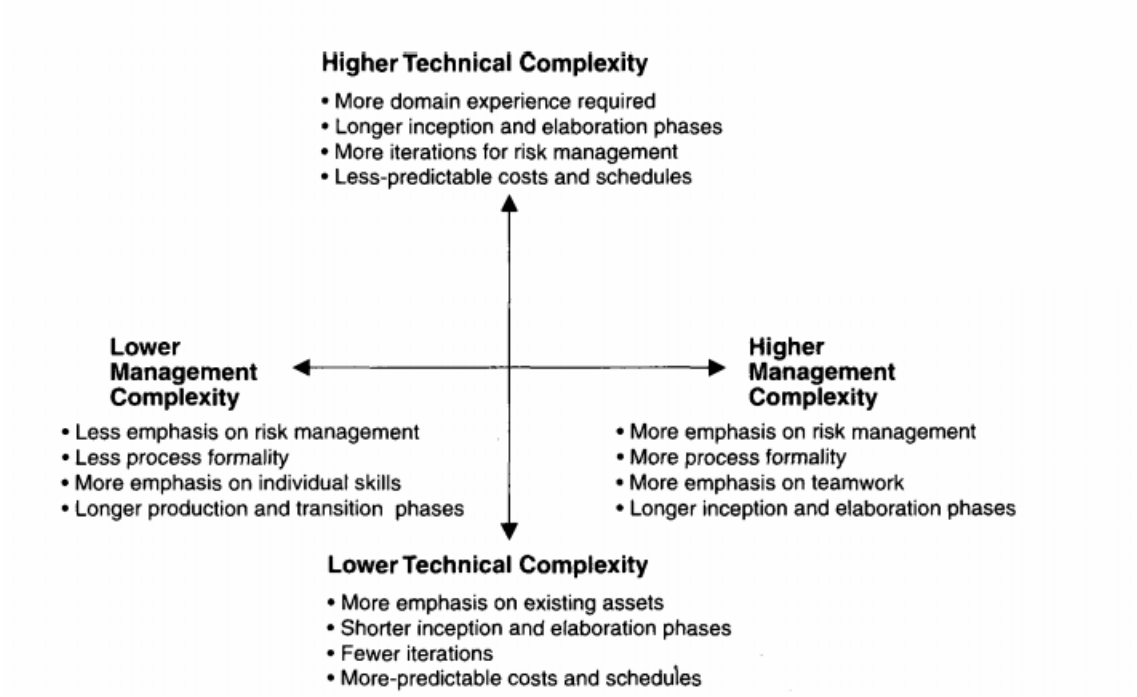


Fig 2 Priorities for tailoring the process framework

15. Table 1 Summarizes some key differences in the process primitives for small and large projects.

Sr.No	Process Primitive	Smaller Team	Larger team
1	Life-cycle phases	Weak boundaries between phases	Well-defined phase transitions to synchronize progress among concurrent activities.
2	Artifacts	Focus on technical artifacts. Few discrete baselines. Very few management artifacts required	Change management of technical artifacts, which may result in numerous baselines. Management artifacts are important to consider
3	Workflow effort allocation	More need for generalists, people who perform roles in multiple workflows	Higher percentage of specialists. More people and teams focused on a specific workflow
4	Checkpoints	Many informal events for maintaining technical	A few formal events synchronization among teams, which can take days.

		consistency. No disruption in schedule is there	
5	Management discipline	Informal planning, project control and organization	Formal planning, project control and organization
6	Automation discipline	More ad hoc environment managed by individuals	Infrastructure to ensure a consistent up-to-date environment available across all teams. Additional tool integration to support project control and change control

2 Stakeholder Cohesion or Contention

1. The degree of cooperation and coordination among stakeholders (buyers, developers, users, subcontractors and maintainer among others) can significantly drive the specifics of how a process is defined.

2. This process parameter can range from cohesive to adversarial. Cohesive teams have common goals, complementary skills and close communications.

3. Whereas adversarial teams have conflicting goals, competing or incomplete skills and less-than open communication.

4. A product that is funded, developed, marketed and sold by the same organization can be set up with a common goal(for eg profitability). A small collocated organization can be established that has a cohesive skill base and excellent day-to-day communications among team members.

5. It is much more difficult to set up a large contractual effort without some contention across teams. Funding authorities and users want to minimize cost, maximize the feature set and accelerate time to market while development contractors want to maximize profitability.

6. Large teams are almost impossible to collocate and synchronizing stakeholder expectations is challenging. All these factors tend to degrade team cohesion and must be managed continuously.

7. Table 2 summarizes the key differences in the process primitives for varying levels of stakeholder cohesion

Sr.No	Process Primitive	Few Stakeholders, Cohesive teams	Multiple Stakeholders, Adversarial relationships
1	Life-Cycle Phases	Weak boundaries between phases	Well-defined phase transitions to synchronize progress among concurrent activities
2	Artifacts	Fewer and less detailed management artifacts required	Management artifacts, paramount, especially the business case, vision and status assessment
3	Workflow effort allocations	Less overhead in assessment	High assessment overhead to ensure stakeholder concurrence
4	Checkpoints	Many informal events	3 0r 4 formal events many informal technical walkthroughs necessary to synchronize among stakeholder teams, which can impede progress for weeks
5	Management discipline	Informal planning, project control and organization	Formal planning, project control and organization
6	Automation Discipline	Insignificant	On-line stakeholder environment necessary

3. Process Flexibility and Rigor

1. The degree of rigor, formality and change freedom inherent in a specific project's "contract" which consist of vision document, business case, and development plan, has a substantial impact on the implementation of the project's process.

2. For very loose contracts such as building a commercial product within a business unit of a software company(such as Microsoft application or a Rational Software Corporation development tool), management complexity is minimal.

3. In these sorts of development processes, feature set, time to market, budget and quality can all be freely traded off and changed with very little overhead. For eg if a company wanted to eliminate a few features in a product under development to capture market share from the competition by accelerating the product release, it would be feasible to make this decision in less than a week.

4. The entire coordination effort might involve only the development manager, marketing manager and business unit manager coordinating some key commitments.

5. On the other hand, for a very rigorous contract, it could take many months to authorize a change in a release schedule. For eg to avoid a large custom development effort, it might be desirable to incorporate a new commercial product in to the overall design of a next generation air traffic control system.

6. This sort of change would require coordination among the development contractor, funding agency, users, certification agencies, associate contractors for interfacing systems and the others.

7. Large-scale catastrophic cost-of-failure systems have extensive contractual rigor and require significantly different management approaches.

8. Table 3 summarizes key differences in the process primitives for varying levels of process flexibility

Sr.No	Process Primitive	Flexible Process	Inflexible process
1	Life-cycle phases	Tolerant of cavalier phase commitments	More credible basis required for inception phase commitments
2	Artifacts	Changeable business case and vision	Carefully controlled changes to business case and vision
3	Workflow effort allocations	Insignificant	Increased levels of management and assessment workflows
4	Checkpoints	Many informal events for main-training technical consistency	3 or 4 formal events synchronization among stakeholder teams, which can impede progress for days or weeks
5	Management discipline	Insignificant	More fidelity required for planning and project control
6	Automation Discipline	Insignificant	-

4 Process Maturity

1. The process maturity level of the development organization, as defined by the software Engineering Institute's Capability Maturity Model is another key driver of management complexity.

2. Managing a mature process(level 3 or higher) is far simpler than managing a immature process(level 1 and 2). Organizations with a mature process typically have a high level of precedent experience in developing software and a high level of existing process collateral that enables predictable planning and execution of the process.

3. This sort of collateral includes well defined methods, process automation tools, trained personnel, planning metrics, artifact templates and workflow templates.

4. Tailoring a mature organization's process for a specific project is generally a straightforward task.

5. Table 4 summarizes key differences in the process primitives for varying levels of process maturity

Sr.No	Process Primitive	Mature Level 3 or 4 organization	Level 1 Organization
1	Life-cycle phases	Well-established criteria for phase transitions	-
2	Artifacts	Well-established format, content and production methods	Free-form
3	Workflow effort allocations	Well-established basis	No basis
4	Checkpoints	Well-defined combination of formal and informal events	-
5	Management discipline	Predictable planning objective status assessments	Informal planning and project control
6	Automation discipline	Requires high levels of automation for round-trip engineering, change management and process instrumentation	Little automation or disconnected islands of automation

5. Architectural Risk

1. The degree of technical feasibility demonstrated before commitment to full-scale production is an important dimension of defining a specific project's process.

2. There are many sources of architectural risk. Some of the most important and recurring sources are system performance (resource utilization, response time, throughput, accuracy), robustness to change(addition of new features, incorporation of new technology, adaption to dynamic operational conditions), and system reliability(predictable behavior, fault tolerance).

3. The degree to which these risks can be eliminated before construction begins can have dramatic difficulties in the process tailoring.

4. Table 5 Summarizes key differences in the process primitives for varying levels of architectural risk

Sr.No	Process Primitive	Complete Architecture Feasibility demonstration	No Architecture feasibility demonstration
1	Life-Cycle Phases	More inception and elaboration phase iterations	Fewer early iterations more construction iterations
2	Artifacts	Earlier breadth and depth across technical artifacts	-
3	Workflow effort allocations	Higher level of design effort. Lower levels of implementation and assessment	Higher level of implementation and assessment to deal with increased scrap and rework
4	Checkpoints	More emphasis on executable demonstrations	More emphasis on briefings, documents and simulations
5	Management discipline	-	-
6	Automation Discipline	More environment resources required earlier in the life cycle	Less environment demand early in the life cycle

6. Domain Experience

1. The development organization's domain experience governs its ability to converge on an acceptable architecture in a minimum number of iterations.

2. An organization that has built five generations of radar control switches may be able to converge on an adequate baseline architecture for a new radar application in two or three prototype release iterations.

3. A skilled software organization building its first radar application may require four or five prototype releases before converging on an adequate baseline.

4. Table 6 Summarizes the key differences in the process primitives for varying levels of domain experience

Sr.No	Process Primitive	Experienced Team	Inexperienced Team
1	Life-Cycle Phases	Shortest engineering stage	Longer engineering stage
2	Artifacts	Less scrap and rework in requirements and design sets	More scrap and rework in requirements and design sets
3	Workflow effort allocations	Lower levels of requirements and design	Higher level of requirements and design
4	Checkpoints	-	-
5	Management discipline	Less emphasis on risk management. Less-frequent status assessments needed	More-frequent status assessments required
6	Automation discipline	-	-

13.3 EXAMPLE- SMALL SCALE PROJECT VERSUS LARGE SCALE PROJECT

1. An analysis of the differences between the phases, workflows and artifacts of two projects on opposite ends of the management complexity spectrum show how different two software project processes can be.

2. The following gross generalizations are intended to point out some of the dimensions of flexibility, priority and fidelity that can change when a process framework is applied to different applications, projects and domains

3. Table 7 explains the differences in schedule distribution for large and small projects across the life-cycle phases. A small commercial project (for example a 50,000 source line visual basic windows application, built by a team of five) may require only 1 month of inception, 2 months of elaborations, 5 months of construction and 2 months of transition.

3 A large, complex project(for example a 300,000 source-line embedded program, built by a team of 40) could require 80 months of inception, 14 months of elaboration, 20 months of construction and 8 months of transition.

Table 7

Sr.No	Domain	Engineering	Production
1	Small Commercial Project	Having a 10% inception and 20 % elaboration	Having 50 % construction and 20%transition
2	Large, Complex project	Having 15% inception and 30% elaboration	Having 40% construction and 15% transition

5. The biggest difference is the relative time at which the life-cycle architecture milestone occurs. This corresponds to the amount of time spent in the engineering stage compared to the production stage. For a small project, the split is about 30/70 for a large project, it is more like 45/55.

6. One key aspect of the differences between the two projects is the leverage of the various process components in the success or failure of the project. This reflect the importance of staffing or the level of associated risk management.

7. The following list elaborates some of the key differences in discriminators of success. Every process component is important depend upon their usage while developing a project

7.1 Design is key in both domain. Good design of a commercial product is valuable aspect in the market place and is the foundation for efficiently creating new releases of a product. Good design of a large, complex project is the foundation for predictable, cost efficient construction.

7.2 Management is key aspect for handling large projects, where the consequences of planning errors, resource allocation errors, inconsistent stakeholder expectations and other out-of-balance factors can have terrible consequences for the overall team dynamics. Whereas management is of less important for handling a small team, where opportunities for miscommunications are fewer and their consequences is less significant.

7.3 Deployment plays a far greater role for a small commercial product because there is a broad user base of diverse individuals and environments.

Table 8 Differences in workflow priorities between small and large prohects.

Rank	Small Commercial Project	Large Complex Project
1	Design	Management
2	Implementation	Design
3	Deployment	Requirements
4	Requirements	Assessment
5	Assessment	Environment
6	Management	Implementation
7	Environment	Deployment

8. A large, one of a kind complex project typically has a single deployment site. Legacy systems and continuous operations may pose several risks, but in general these problems are well understood and have a fairly static set of objectives.

9. Another key set of differences in inherent in the implementation of the various artifacts of the process. Table 9 provides a conceptual example of these differences

Sr. No	Artifact	Small Commercial Project	Large Complex Project
1	Work break down structure	1-page spreadsheet with 2 levels of WBS elements	Financial management system with 5 or 6 levels of WBS elements
2	Business Case	Spreadsheet and short memo	3-volume proposal including technical volume, cost volume, and related experience

3	Vision Statement	10-page concept paper	200-page subsystem specification
4	Development plan	10-page plan	200-page development plan
5	Release specifications and number of releases	3 interim release specifications	8 to 10 interim release specifications
6	Architecture description	5 Critical use cases, 50 Uml diagrams, 20 pages of text and other graphics	25 critical use cases, 200 UML diagrams, 100 pages of text, other graphics
7	Software	50,000 lines of Visual basic code	300,000 lines of Code
8	Release description	10-page release notes	100-page summary
9	Deployment	User training course sales rollout kit	Transition plan and installation plan
10	User Manual	On-line help and 100-page user manual	200-page user manual
11	Status assessment	Quarterly project reviews	Monthly project management reviews

Eg Case study on Small Project Cost Estimating at Percy Company

Paul graduated from college in June 1970 with a degree in industrial engineering. He accepted a job as a manufacturing engineer in the Manufacturing Division of Percy Company. His prime responsibility was performing estimates for the Manufacturing Division. Each estimate was then given to the appropriate project office for consideration. The estimation procedure history had shown the esti-mates to be valid.

In 1975, Paul was promoted to project engineer. His prime responsibility was the coordination of all estimates for work to be completed by all of the divisions. For one full year Paul went by the book and did not do any estimating except for project office personnel manager. After all, he was now in the project management division, which contained job descriptions including such words as “coordinating and integrating.”

In 1976, Paul was transferred to small program project management. This was a new organization designed to perform low-cost projects. The problem was that these projects could not withstand the expenses needed for formal divisional cost estimates. For five projects, Paul’s estimates were “right on the money.” But the sixth project incurred a cost overrun of $20,000 in the Manufacturing Division. In November 1977, a meeting was called to resolve the question of “Why did the overrun occur?” The

attendees included the general manager, all division managers and directors, the project manager, and Paul. Paul now began to worry about what he should say in his defense.

Eg Case Study on Project Planning at Payton Corporation

Payton Corporation had decided to respond to a government RFP for the R&D phase on a new project. The statement of work specified that the project must be completed within ninety days after go-ahead, and that the contract would be at a fixed cost and fee.

The majority of the work would be accomplished by the development lab. According to government regulations, the estimated cost must be based on the average cost of the entire department, which was $19.00 per hour (unburdened).

.Payton won the contract for a total package (cost plus fee) of $305,000. After the first weekly labor report was analyzed, it became evident that the development lab was spending $28.50 per hour. The project manager decided to discuss the problem with the manager of the development lab. Project manager :"Obviously you know why I'm here. At the rate that you are re-spending money, we'll overrun our budget by 50 percent." Lab manager: "That's your problem, not mine. When I estimate the cost to do a job, I submit only the hours necessary based on historical standards. The pricing department converts the hours to dollars based on department averages ."Project manager:" Well, why are we using the most expensive people? Obviously there must be lower-salaried people capable of performing the work."

Lab manager: "Yes, I do have lower-salaried people, but none who can complete the job within the two months required by the contract. I have to use people high on the learning curve, and they're not cheap. You should have told the pricing department to increase the average cost for the department."

Project manager: "I wish I could, but government regulations forbid this. If we were ever audited, or if this proposal were compared to other salary structures in other proposals, we would be in deep trouble. The only legal way to accomplish this would be to set up a new department for those higher-paid employees working on this project. Then the average department salary would be correct.

Unfortunately the administrative costs of setting up a temporary unit for only two months is prohibitive. For long-duration projects, this technique is often employed. "Why couldn't you have increased the hours to compensate for the increased dollars required?

Lab manager: "I have to submit labor justifications for all hours I estimate. If I were to get audited, my job would be on the line. Remember, we had to submit labor justification for all work as part of the proposal.

Perhaps next time management might think twice before bidding on a short-duration project. You might try talking to the customer to get his opinion."

Project manager: "His response would probably be the same regardless of whether I explained the situation to him before we submitted the proposal or now, after we have negotiated it. There's a good chance that I've just lost my Christmas bonus.

13.4 SUMMARY

Software management efforts span a broad range of domains. While there are some universal themes and techniques, it is always necessary to tailor the process to the specific needs of the project at hand. A commercial software tool developer with complete control of its investment profile will use a very different process from that of a software integrator on contract to automate the security system for a nuclear power plant. There is no doubt that a mature process and effective software management approaches offer much greater value to the large-scale software integrator than they do to the small-scale tool developer.

13.5 REFERENCES

[1] Software Engineering by Sommerville 8th Edition

13.6 QUESTIONS

Q1. Describe the need of scale as process discriminants ?
Q2. Illustrate the concept of Stakeholder Cohesion or Contention
Q3. Explain the concept of process flexibility or Rigor and process maturity?
Q4. Explain the six dimension of process discriminants?
Q5. Suppose Company A wants to build small scale project and Company B wants to build large scale project. So list the work flow priority followed by A and workflow priority followed by B?

Unit - VI

14

FUTURE SOFTWARE PROJECT MANAGEMENT

Unit Structure

14.0 OBJECTIVES

At the end of this unit, the learner will be able to

- Describe the profile of modern project with its different dimension
- Identify the need of Next-Generation Software economies
- Illustrate the concept of modern process transitions

14.1 INTRODUCTION

1. A modern process framework exploits approaches to solve issues having in conventional framework. These approaches are as follows

1.1 Protracted integration and late design breakage are resolved by forcing integration in to the engineering stage. This is achieved through continuous integration of an architecture baseline supported by executable demonstrations of the primary scenarios

1.2 Late risk resolution is resolved by emphasizing an architecture-first approach, in which the high-leverage elements of the systems are elaborated early in the life cycle.

1.3 The analysis paralysis of a requirements-driven functional decomposition is avoided by organizing lower level specifications

along the content of releases rather than along the product decomposition.

1.4 Adversarial stakeholder relationships are avoided by providing much more tangible and objective results throughout the life cycle

1.5 The conventional focus on documents and review meetings is replaced by a focus on demonstrable results and well-defined sets of artifacts, with more rigorous notations and extensive automation supporting a paperless environment.

2. Next-generation software economies is being practiced by some advanced software organizations.

3. A mature modern process is now where near the state of the practice for the average software organizations.

4. The new approaches of modern process framework will improve the accuracy and precision of software cost estimates and would accommodate dramatic improvements in software economies of scale.

5. Successful software management is hard work. Technical breakthroughs, process breakthroughs and new tools will make it easier for a project to success.

6. New technological advances will be accompanied by new opportunities for software applications, new dimensions of complexity, new avenues of automation and new customers with different priorities.

7. Accomodating new changes will disturb many of ingrained software management values and priorities but maintaining a balance among requirements, designs, and plans will retain the underlying objective of future software management activities, just as it is today.

8. Some of the important culture shifts to be prepared for in order to avoid as many sources of friction as possible in transitioning successfully to a modern process.

14.2 MODERN PROJECT PROFILES

1.Iterative development producesthe architecture first, allowing integration to occur as the verification activity of the design phase and enabling design flaws to be detected and resolved earlier in the life cycle.

2. This approach avoids the big-bang integration at the end of a project by stressing continuous integration throughout the project.

3. Figure 1 shown below explains the differences between the progress profile of a healthy modern project. The downstream integration nightmare, late patches and shoe-horned software fixes are avoided. The result is more robust and maintainable design.

4 It inherent in an iterative development process also enables better insight in to quality trade-offs. A recurring theme of successful iterative development projects is a cost profile very different from that experienced by conventional processes.

5. Conventional projects stuck in inefficient integration and late discovery of substantial design issues, expend roughly 40% or more of their total resources integration and test activities. Whereas modern projects with a mature, iterative process deliver a product with only about 25% of the total budget consumed by these activities.

Table 1 Differences in workflow cost allocations between a conventional process and a modern process

SR.No	Software Engineering Work Flows	Conventional Process Expenditures	Modern Process Expenditures
1	Management	5%	10%
2	Environment	5%	10%
3	Requirements	5%	10%
4	Design	10%	15%
5	Implementation	30%	25%
6	Assessment	40%	25%
7	Deployment	5%	5%
	Total	100%	100%

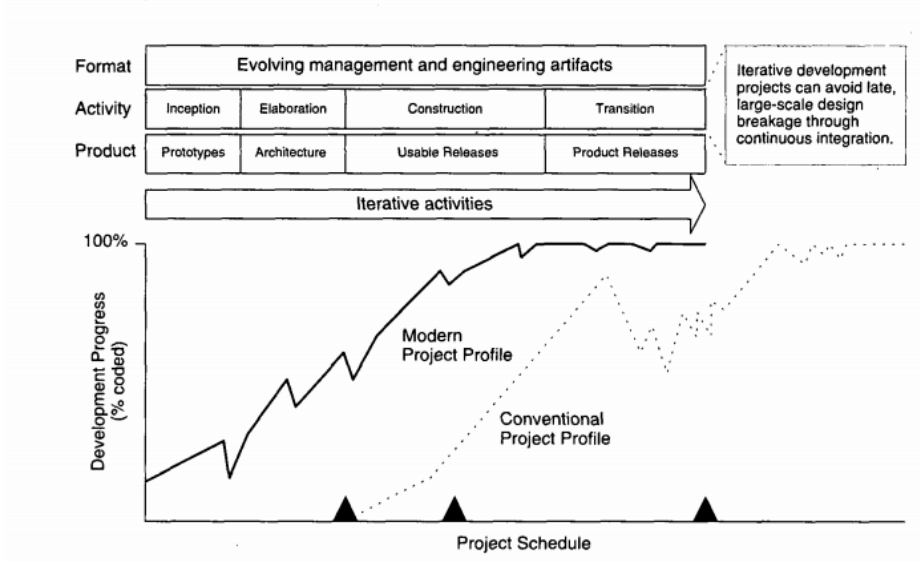


Fig 1 Progress profile of a modern project

2. Early Risk Resolution

1. The engineering stage of the life cycle (inception and elaboration phases) focuses on confronting the risks and resolving them before the big resource commitments of the production stage.

2. Conventional projects usually do the easy stuff first, thereby demonstrating early progress, whereas a modern process attacks the important 20% of the requirements, use cases, components and risks.

3. This is the essence of most important principle that is design the architecture first, to get the success about the project. The effect of the overall life-cycle philosophy on the 80/20 lessons learned over the past 30 years of software management experience provides a useful risk management perspective.

   3.1 80 % of the engineering is consumed by 20% of the requirements. Strive to understand the driving requirements completely before committing resources to full scale development.

   3.2 80% of the software cost is consumed by 20% of the components- Elaborate the cost-critical components first so that planning and control of cost drivers are well understood early in the life cycle.

   3.3 80 % of the errors are caused by 20% of the components- Elaborate the reliability critical components first so that assessment activities have enough time to achieve the necessary level of maturity.

   3.4 80% of the software scrap and rework is caused by 20% of the changes- Elaborate the change-critical components first so that broad-impact changes occur when the project is nimble.

   3.5 80% of the resource consumption(execution time, disk space, memory) is consumed by 20% of the components- Elaborate the performance-critical components first so that engineering trade-offs with reliability, changeability, and cost-effectiveness can be resolved as early in the life cycle as possible.

   3.6 80 % of the progress is made by 20% of the people- Make sure that the initial team for planning the project and designing the architecture is of the highest quality. An inadequate plan or inadequate architecture will probably not succeed, even with an expert construction team.

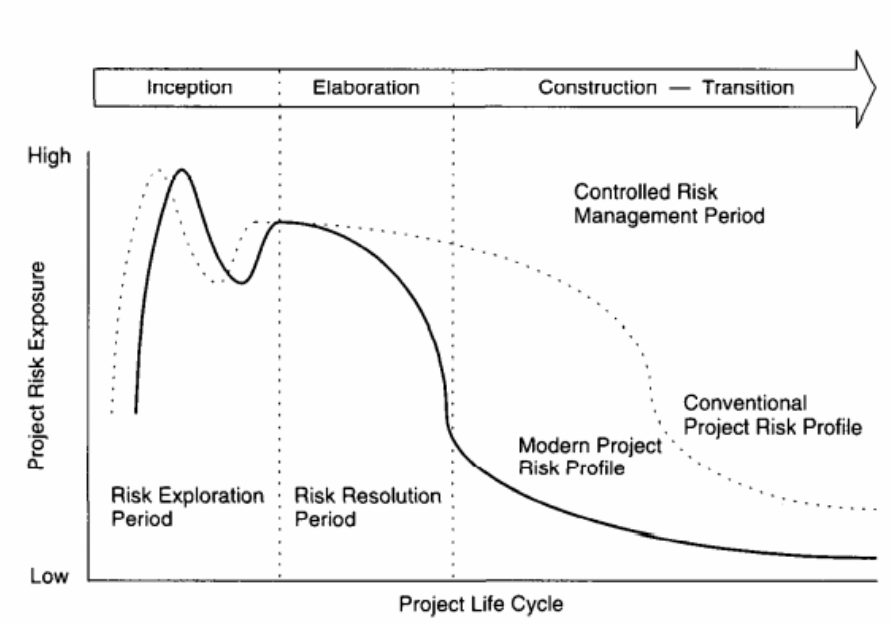


Fig 2 Risk profile of a typical modern project across its life cycle

3 Evolutionary Requirements

1. Conventional approaches decomposed system requirements in to subsystem requirements, subsystem requirements in to component requirements and component requirements in to unit requirements.

2. With an early life-cycle emphasis on requirements first, design second, then complete traceability between requirements and design components, the natural tendency was for the design structure to evolve in to an organization that closely paralleled the structure of the requirement organization.

3. Most modern architectures that use commercial components, legacy components, distributed resources and object-oriented methods are not irrelevantly traced to the requirement they satisfy. There are now complex relationships between requirement statements and design elements which includes 1 to 1, many to 1, 1 to many, conditional, time-based and state-based.

4. Top-level system requirements are retained as the vision, but lower level requirements are captured in evaluation criteria attached to each intermediate release. The fundamental difference from conventional requirements management approaches in which this fidelity was pursued far too early in the life cycle.

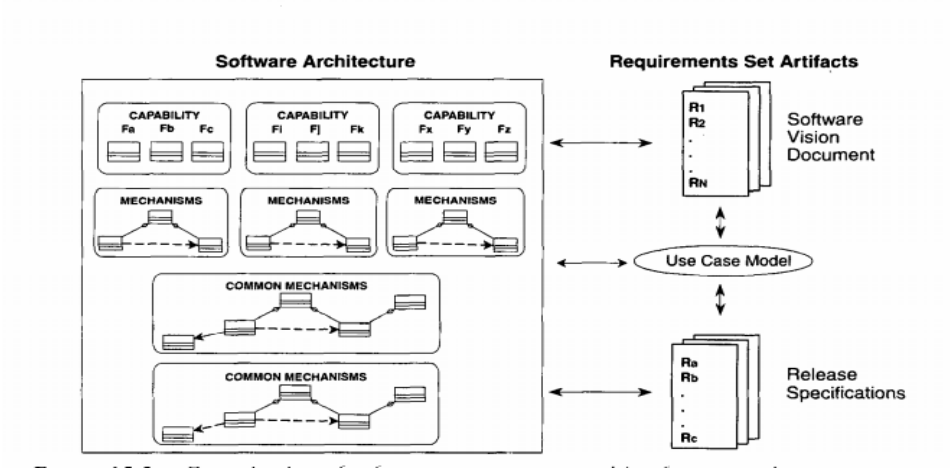


Fig 3 Organization of software components resulting from a modern process

4 Team work among stakeholder

1 Many aspects of the classic development process cause stakeholder relationships to degenerate in to mutual distrust making it difficult to balance requirements, product features, and plans.

2 A more iterative process, with more effective working relationships between stakeholders, allows trade-ffs to be based on a more objective understanding by everyone. This process requires that customers, users and monitors have both applications and software expertise, remain focused on the delivery of a usable system and be willing to allow the contractor to make a profit with good performance.

3. It also requires a development organization that is focused on achieving customer satisfaction and high product quality in a profitable manner.

4. The transition from the exchange of mostly paper artifacts to demonstration of intermediate results is one of the crucial mechanisms for promoting teamwork among stakeholders.

5. The project does not move forward until the objectives of the demonstration have been achieved. This prerequisite does not preclude the renegotiation of objectives once the demonstration and major milestone has achieved as per the requirements, design, plans and technology.

6. A modern iterative process that focuses on demonstrable results requires all stakeholders to be educated in the important difference between apparently negative results and proof of real progress.

7. For eg A flaw in design is if discovered early, indicate the positive flow of project progress, rather than a major issue.

8.Table 2 describes the results of major milestones in a modern process

Sr.No	Apparent Result	Real Result
1	Early demonstrations expose design issues and ambiguities in a tangible form.	Demonstrations expose the important assets and risks of complex software systems early, when they can be resolved within the context of life cycle goals.
2	The design is non compliant(so far)	Understanding of compliance matures from important perspectives (architecturally significant requirements and use cases)
3	Driving requirements issues are exposed, but detailed requirements traceability is lacking.	Requirements changes are considered in balance with design trade-offs
4	The design is considered "guilty until proven innocent"	Engineering progress and issues are a tangible for incorporation in to the next iteration's plans.

5. Top 10 Software Management Principles

1. The following list provides a concise, top-level description of the features and benefits of a modern process as viewed by a software project manager

1.1 Base the process on an architecture-first approach- Getting the architecturally important components to be well understood and stable before worrying about the complete breadth and depth of the artifacts should result in scrap and rework rates remain stable over the project life cycle.

1.2 Establish an iterative life-cycle process that confronts risk early- Resolving the critical issues first results in a predictable construction phase with no surprises as well as minmal exposure to sources of cost and schedule unpredictability.

1.3 Transition design methods to emphasize component based development- The complexity of a software effort is mostly a function of the number of human-generated artifacts. Making the solution smaller reduces management complexity.

1.4 Establish a change management environment – The dynamics of iterative development, including concurrent workflows by different

teams working on shared artifacts, necessitate highly controlled baselines.

1.5 Enhance change freedom through tools that support round-trip engineering- Automation enables teams to append more time on engineering and less time on overhead tasks.

1.6 Capture design artifacts in rigorous, model-based notation- An engineering notation for design enables teams to spend complexity control, objective assessment and automated analysis.

1.7 Instrument the process for objective quality control and progress assessment- Progress and quality indicators are derived directly from the evolving artifacts, providing more-meaningful insight in to trends and correlation with requirements.

1.8 Use a demonstration-based approach to assess intermediate artifacts- Integration occurs early and continuously throughout the life cycle.

1.9 Plan intermediate releases in groups of usage scenarios with evolving levels of detail- Requirements, designs and plans evolve in balance. Useful software releases are available early in the life cycle.

1.10 Establish a configurable process that is automatically scalable- Methods, techniques, tools, and experience can be applied straightforwardly to a broad domain, providing improved return on investment across a line of business.

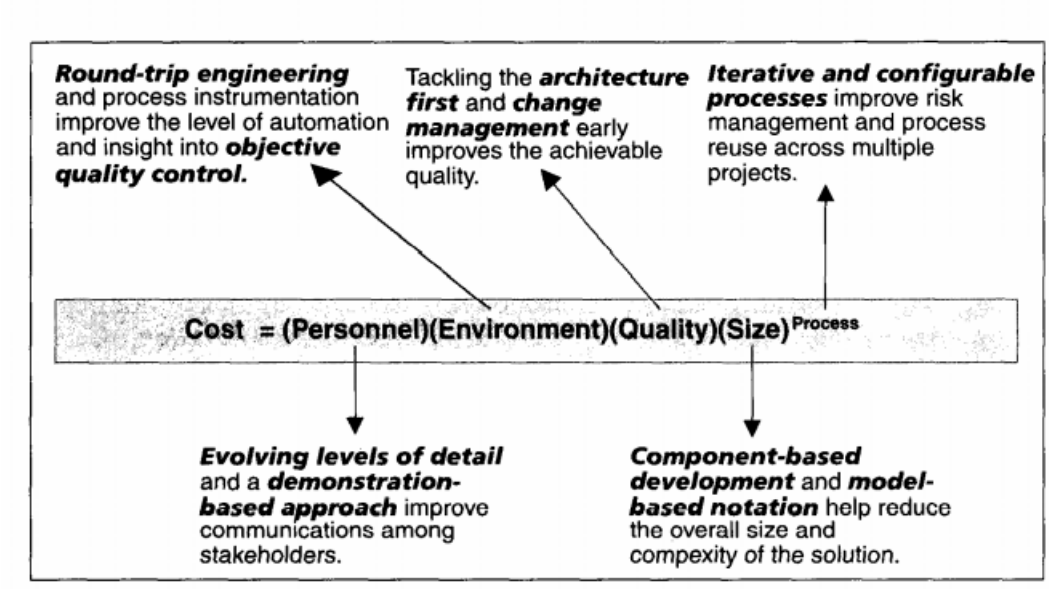


Fig 4 Balanced application of modern principles to achieve economic results

6. Software Management best practices

1. Many software management best practices have been captured by various authors and industry organizations. Brown summarized the intiative which has three components like the airline software council,

seven different issue panels and a program manager's panel. Each component produced recommendations and results and reviewed the work of the other components.

2. The Airline software e council was "purposely structured to include highly successful managers of large-scale software projects, internationally recognized authors, prominent consultants, and executives responsible for software development at major companies.

3. The nine best practices are described here as follows

   3.1 Formal risk management- using an iterative process that confronts risk is more or less what this is saying.

   3.2 Agreement or Interfaces- Getting the architecture baselined forces the project to gain agreement on the various external interfaces and the important internal interfaces, all of which are inherent in the architecture.

   3.3 Formal inspections-The assessment workflow throughout the life cycle, along with the other engineering workflows, must balance several different defect removal strategies.

   3.4 Metric based scheduling and management- This important principle is directly related to my model-based notation and objective quality control principles.

   3.5 Binary quality gates at the inch pebble level- Too many projects have taken exactly this approach early in the life cycle and have laid out a highly detailed plan at great expense. A better approach would be to maintain fidelity of the plan commensurate with an understanding of the requirements and the architecture.

   3.6 Program wide visibility of progress versus plan- This practice namely, open communications among project team members is obviously necessary.

   3.7 Defect tracking against quality targets- The make or break defects and quality targets are architectural. Getting a handle on these qualities early and tracking their trends are requirements for success.

   3.8 Configuration Management- It also recognized that automation is important because of the volume and dynamics of modern, large-scale projects which make manual methods cost-prohibitive and error-prone.

   3.9 People aware management accountability- This is another management principle that seems so obvious.

14.3 NEXT GENERATION SOFTWARE ECONOMICS

1 Next Generation Cost Models

1. Software experts hold varying opinions about software economics and its manifestation in software cost estimation models. Source lines of code versus function points.

2. Economy of scale versus diseconomy of scale, productivity measures versus quality measures, Java versus C++, object oriented versus procedure oriented, commercial components versus custom development. All these topics represent industry debates surrounded by high levels of bombast.

3. Accurate estimates are possible today, although honest estimates are imprecise. It will be difficult to improve empirical estimation models while the project data going in to these models are noisy and highly uncorrelated, and are based on different process and technology foundations.

4. Some of today's popular software cost models are not well matched to an iterative software process focused on an architecture first approach. Despite many advances in software cost estimation tools to update their project repository, but still many cost estimators are using conventional process to estimate the modern project profile.

5. A next-generation software cost model should explicitly separate architectural engineering from application production, just as an architecture first process does. When an organization achieves a stable architecture, the production costs should be an exponential function of size, quality and complexity with a much more stable range of process and personnel influence.

6. Next-generation software cost models should estimate large-scale architectures with economy of scale.

7. In the conventional process, the minimal level of automation that supported the overhead activities of planning, project control and change management led to labor intensive workflows and a diseconomy of scale. Next generation environment and infrastructures are moving to automate and standardize many of these management activities, thereby requiring a lower percentage of effort for overhead activities as scale increases. The figure given below summarizes an hypothesized cost model for an architecture first development process.

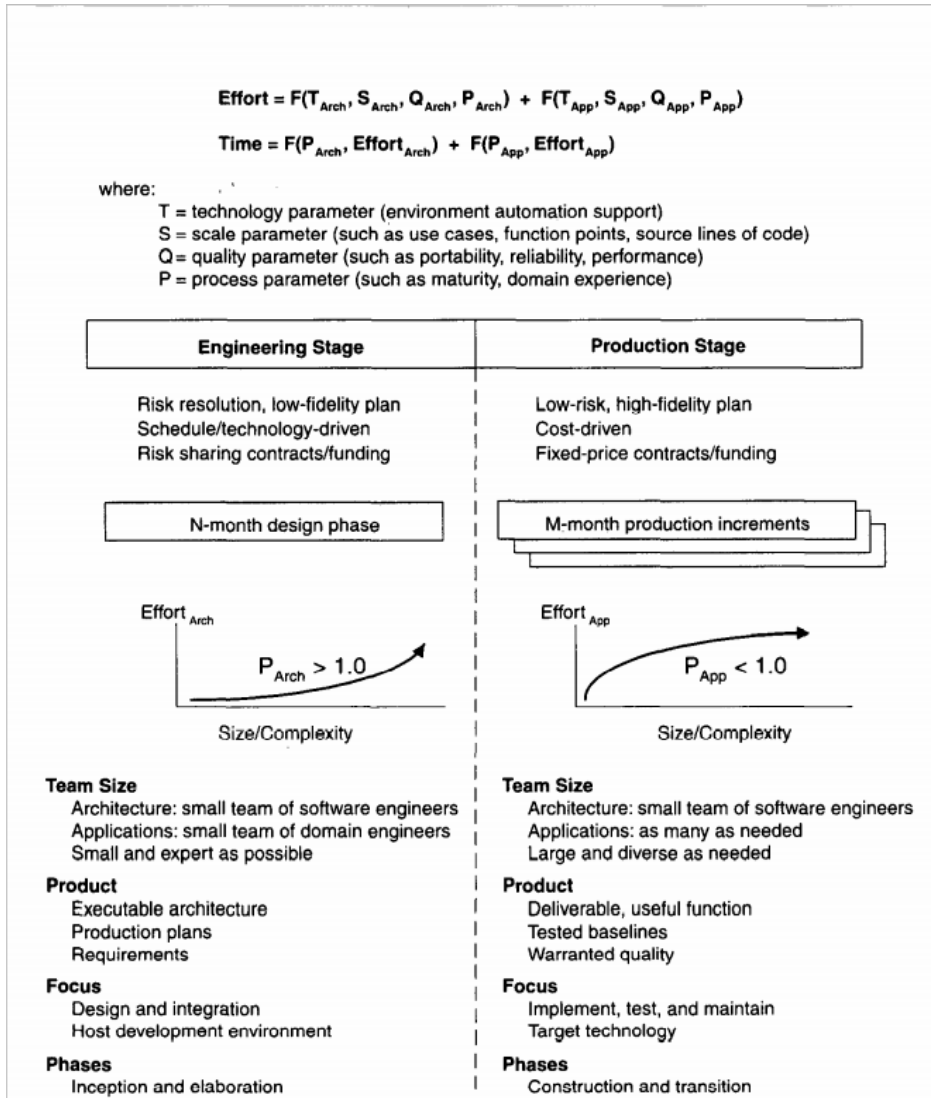


Fig 1 Next-generation Cost models

8. Reusing common processes across multiple iterations of a single project, multiple releases of a single product, or multiple projects in an organization also relieves many of the sources of diseconomy of scale. While most reuse of components results in reducing the size of the production effort, the reuse of processes tools, and experience has a direct impact on the economies of scale.

9. Another important difference in this cost model is that architectures and applications have different units of mass (scale V/s size) and are representations of the solution space. However there are many solutions are available for any given problem as illustrated in fig shown below.

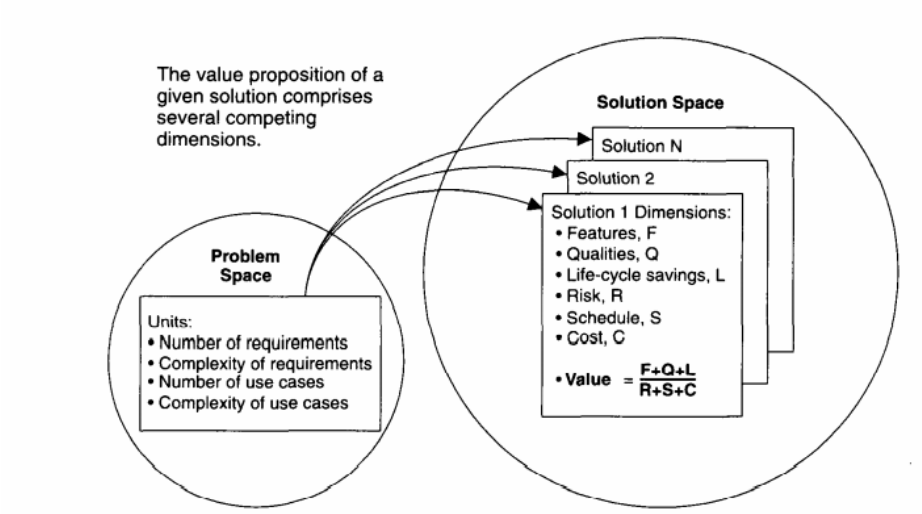


Fig 2 Differentiating potential solutions through cost estimation

10. The value of the function points is that they are better at depicting the overall scale of the solution, independently of the actual size and implementing language of the final realization. A rigorous notation for design artifacts is a necessary prerequisite to improvements in the fidelity with which the scale of a design can be estimated.

11. Two major improvements in next-generation software cost estimation models are as follows

1. Separation of the engineering stage from the production stage will force estimators to differentiate between architectural scale and implementation size.

2. Rigorous design notations such as UML will offer an opportunity to define units of measure of scale that are most standardized and therefore can be automated and tracked.

12. The first breakthrough would be the availability of integrated tools that automate the transitions of information between requirements, design, implementation and deployment elements. These tools would allow more comprehensive round-trip engineering among the engineering artifacts.

13. The second breakthrough would focus on collapsing today's four set of fundamental technical artifacts in to three sets by automating the activities associated with human-generated source code, thereby eliminating the need for a separate implementation set. This technology is illustrated in fig below.

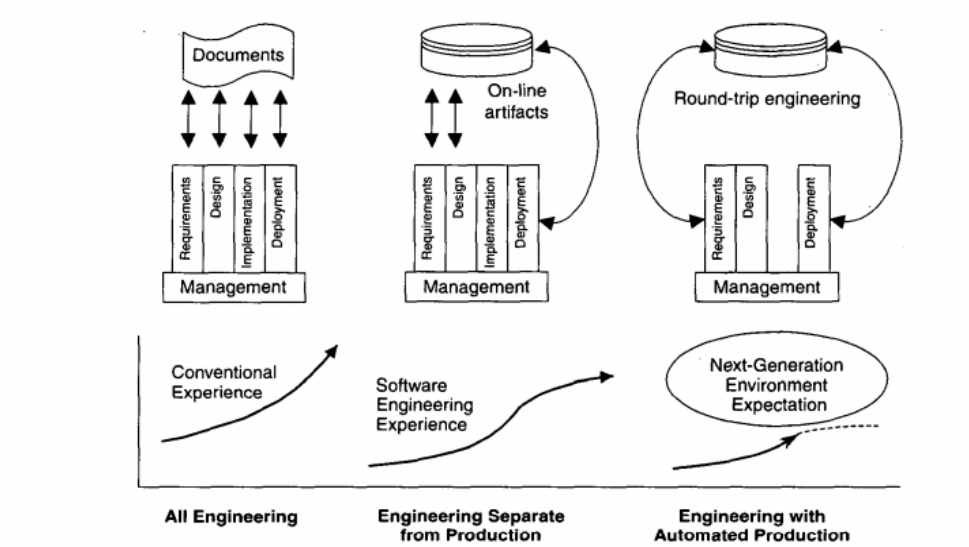


Fig 3 Automation of the construction process in next generation environment

2 Modern Software economics

1. Finding and fixing a software problem after delivery costs 100 times more than finding and fixing the problem in early design phases- Modern processes, component based development technologies and architecture frameworks are explicitly targeted at improving this relationship.

2. You can compress software development schedules 25% nominal, but no more- This metric remain valid for the engineering stage of the lifecycle, when the intellectual content of the system is evolved.

3. For every $1 you spend on development, you will spend %2 on maintenance- It is difficult to generalize about this metric, as there are many different maintenance models.

4. Software development and maintenance costs are primarily a function of the number of source lines of code-This metric says that the size of the product is the primary cost driver, and the fundamental unit of size is a line of code. The next-generation cost models should become less sensitive to the number of source lines and more sensitive to the discrete numbers of components and their ease of integration.

5. Variatians among people account for the biggest differences in software productivity- For any engineering venture in which intellectual property is the real product, the dominant productivity factors will be personnel skills, team work and motivation.

6. The overall ratio of software to hardware costs is still growing. In 1955, it was 15:85, in 1985, 85:15-The main impact of this metric on software economics is that hardware continues to get cheaper.

7. Only about 15% of software development effort is devoted to programming- In the past 10 years there has been a noticeable shift away from investments in languages and compilers. Modern technology investments have transitioned in to process maturity, automated software quality, configuration management, metrics and other aspects of software engineering.

8. Software systems and products typically cost 3 times as much per SLOC as individual software programs. Software -system products (i.e system of systems) cost 9 times as much- This diseconomy of scale should be greatly relieved with a modern process and modern technologies.

9. Walkthroughs catch 60% of the errors- The really important architectural issues can be exposed only through demonstration and early testing and resolved through human scrutiny

10. 80% of the contribution comes from 20% of the contributors- This relationship is timeless and constitutes the background philosophy to be applied throughout the planning and conduct of a modern software management process.

14.4 MODERN PROCESS TRANSITIONS

1 Culture Shifts- Several culture shifts must be overcome to transition successfully to a modern software management process. Many of these indicators are derived directly from the already existed process framework.

1.1 Lower level and mid-level managers are performers- There should be no pure managers in an organization or suborganization with 25 or fewer people. The need for pure manager arises only when personnel resources exceed this level. Hand-on management skills vary, but competent managers typically spend much of their time performing, especially with regard to understanding the status of the project first hand and developing plans and estimates.

1.2 Requirements and design are fluid and tangible- The conventional process focused too much on producing documents that attempted to describe the software product and focused too little on producing tangible increments of the products themselves. The transition to a less document-driven environment will be embraced by the engineering team, it will probably be resisted by traditional contract monitors.

1.3 Ambitious demonstrations are encouraged- The purpose of early life-cycle demonstration is to expose design flaws, not to put up a façade. The management team is most likely to resist this transition as it will expose any engineering or process issues that were easy to hide using the conventional process.

1.4 Good and bad project performance is much more obvious earlier in the life cycle- In an iterative development, success breeds success and early failures are extremely risky to turn around. If the planning and architecture phases are not performed adequately, all the expert programmers and testers in the world probably will not achieve success.

1.5 Early increments will be immature-External stakeholders such as customers and users cannot expect initial deliveries to perform up to specification to be complete to be fully reliable or to have end-target levels of quality or performance. Customers and users will have difficulty accepting the flaws of early releases, although they should be impressed by later increments.

1.6 Artifacts are less important early more important later- It is waste of time to worry about the details of the artifacts sets until a baseline is achieved that is useful enough and stable enough to warrant time-consuming analysis of these quality factors.

1.7 Real issues are surfaced and resolved systematically- Successful projects recognize that requirements and designs evolve together, with continuous negotiation, trade-off and bartering toward best value, rather than blindly adhering to an ambiguous contract statement.

1.8 Quality assurance is everyone's job not a separate discipline- Many organizations have a separate group called quality assurance. The software project manager or designee should assume the role of ensuring that quality assurance is properly integrated in to the process.

1.9 Performance issues arise early in the life cycle- Early performance issues surfaced on almost every successful project. Development engineers will embrace the emphasis on early demonstration and the ability to assess and evaluate performance trade-offs in subsequent releases.

1.10 Investments in automation are necessary- As iterative development projects require extensive automation it is important not to underinvest in the capital environment. The investment may be opposed by organization managers overly focused on near term financial results or by project personnel who favor the preference of the individual project over the global solution that serves both the project and the organization goals.

1.11 Good software organizations should be more profitable- In the commercial software domain, this is not an issue. This may be issue for governmental contracts. The simple profit motive that underlies commercial transactions and incentives efficiency is replaced by complex contractual incentives that are usually suboptimal. For the software industry to prosper, good contractors should be rewarded and bad contractors should be punished. This is one area in which the commercial domain is far more effective than the government contracting domain.

2. Denouement

1. The conventional software process was characterized by following points as illustrated below

   1.1 Sequentially transitioning from requirements to design to code to test.

   1.2 Achieving 100% completeness of each artifact at each life-cycle stage.

   1.3 Treating all requirements, artifacts, components and so forth as equals .

   1.4 Achieving high fidelity traceability among all artifacts at each stage in the life cycle.

2 A modern iterative development process framework is characterized by the following

   2.1 Continuous round-trip engineering from requirements to test at evolving levels of abstraction.

   2.2 Achieving high fidelity understanding of the drivers (the 20%) as early as practical.

   2.3 Evolving the artifacts in breadth and depth based on risk management priorities.

   2.4 Postponing completeness and consistency analysis until later in the life cycle.

3 Figure shown below illustrates the next generation of software project performance by depicting the development progress versus time, where progress is defined as percent coded.

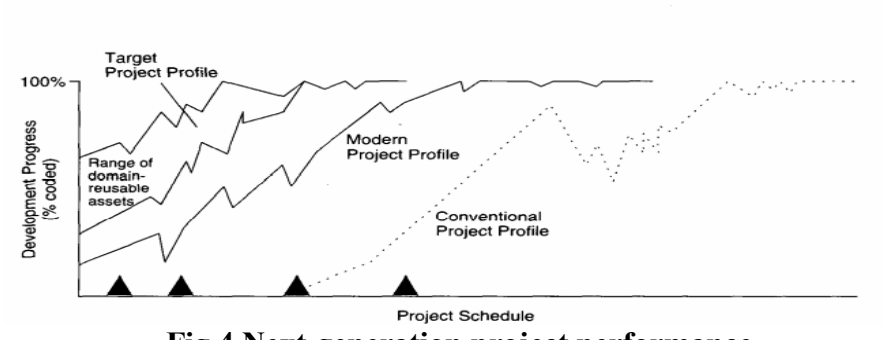


Fig 4 Next-generation project performance

4 When an organization decides to make a transition, these two pieces of conventional wisdom are usually offered by internal champions as well as external change events 1)Pioneer any new techniques on a small pilot program 2) Be prepared to spend more resources money and time on your first project that makes the transition.

5 A better way to transition to a more mature iterative development process that supports automation technologies and modern architectures is to take the following shot-

5.1 Ready- Analyze modern approaches and technologies. Support it with mature environments, tools and components. Plan thoroughly

5.2 Aim- Select a critical project. Staff it with the right team of complementary resources and demand improved results.

5.3 Fire- Execute the organizational and project-level plans with vigor and follow-through.

14.5 SUMMARY

Next-generation software economies is being practiced by some advanced software organizations.A mature modern process is now where near the state of the practice for the average software organizations.The new approaches of modern process framework will improve the accuracy and precision of software cost estimates and would accommodate dramatic improvements in software economies of scale.

14.6 REFERENCES

[1] Software Engineering by Sommerville 8th Edition

14.7 QUESTIONS

Q1. Explain the dimensions of Next-generation software economies

Q2. Illustrate with example why modern process transition is necessity for developing project

Q3. Explain the Cultural shift concept in detail?

Q4. Explain top 10 software management principles?

Q5. Describe about the software management best practices?