CS605
Software Engineering-II

Table of Content

Lecture No. 1

Introduction to Software Engineering

This course is a continuation of the first course on Software Engineering. In order to set the context of our discussion, let us first look at some of the definitions of software engineering.

Software Engineering is the set of processes and tools to develop software. *Software Engineering is the combination of all the tools, techniques, and processes that used in software production.* Therefore Software Engineering encompasses all those things that are used in software production like:

- Programming Language
- Programming Language Design
- Software Design Techniques
- Tools
- Testing
- Maintenance
- Development etc.

So all those thing that are related to software are also related to software engineering.

Some of you might have thought that how programming language design could be related to software engineering. If you look more closely at the software engineering definitions described above then you will definitely see that software engineering is related to all those things that are helpful in software development. So is the case with programming language design. Programming language design is one of the major successes in last fifty years. The design of Ada language was considered as the considerable effort in software engineering

MC

These days object-oriented programming is widely being used. If programming languages will not support object-orientation then it will be very difficult to implement object-oriented design using object-oriented principles. All these efforts made the basis of software engineering.

Well-Engineered Software

Let’s talk something about what is well-engineered software. Well-engineered software is one that has the following characteristics.

- It is reliable
- It has good user-interface
- It has acceptable performance
- It is of good quality
- It is cost-effective

Every company can build software with unlimited resources but well-engineered software is one that conforms to all characteristics listed above.

MC

Software has very close relationship with economics. When ever we talk about engineering systems we always first analyze whether this is economically feasible or not. Therefore you have to engineer all the activities of software development while keeping its economical feasibility intact.

MC

The major challenges for a software engineer is that he has to build software within limited time and budget in a cost-effective way and with good quality

Therefore well-engineered software has the following characteristics.

- Provides the required functionality
- Maintainable
- Reliable
- Efficient
- User-friendly
- Cost-effective

But most of the times software engineers ends up in conflict among all these goals. It is also a big challenge for a software engineer to resolve all these conflicts.

The Balancing Act!

Software Engineering is actually the balancing act. You have to balance many things like cost, user friendliness, Efficiency, Reliability etc. You have to analyze which one is the more important feature for your software is it reliability, efficiency, user friendliness or something else. There is always a trade-off among all these requirements of software. It may be the case that if you try to make it more user-friendly then the efficiency may suffer. And if you try to make it more cost-effective then reliability may suffer. Therefore there is always a trade-off between these characteristics of software.

These requirements may be conflicting. For example, there may be tension among the following:

- C st vs. Efficiency
- Cost vs. Reliability
- Efficiency vs. User-interface

A

Software Engineer is required to analyze these conflicting entities and tries to strike a balance.

Challenge is to balance these requirements.

Software Engineers always confront with the challenge to make a good balance of all these tings depending on the requirements of the particular software system at hand. He should analyze how much weight should all these things get such that it will have acceptable quality, acceptable performance and will have acceptable user-interface.

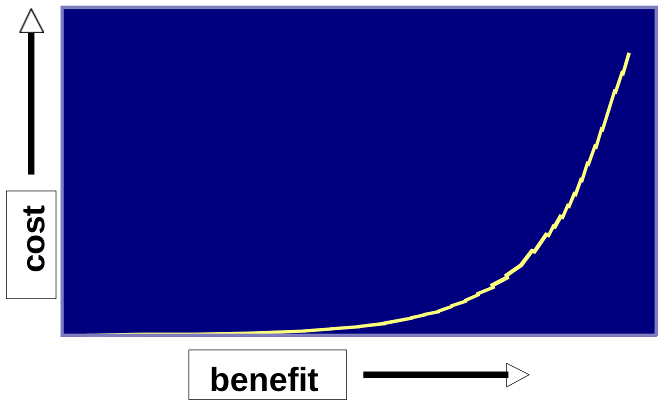
In some software the efficiency is more important and desirable. For example if we talk about a cruise missile or a nuclear reactor controller that are droved by the software systems then performance and reliability is far more important than the cost-effectiveness and user-friendliness. In these cases if your software does not react within a certain amount of time then it may result in the disaster like Chernobyl accident.

Therefore software development is a process of balancing among different characteristics of software described in the previous section. An it is an art to come up with such a good balance and that art can be learned from experience.

Law of diminishing returns

In order to understand this concept lets take a look at an example. Most of you have noticed that if you dissolve sugar in a glass of water then the sweetness of water will increase gradually. But at a certain level of saturation no more sugar will dissolved into water. Therefore at that point of saturation the sweetness of water will not increase even if you add more sugar into it.

The law of diminishing act describes the same phenomenon. Similar is the case with software engineering. Whenever you perform any task like improving the efficiency of the system, try to improve its quality or user friendliness then all these things involves an element of cost. If the quality of your system is not acceptable then with the investment of little money it could be improved to a higher degree. But after reaching at a certain level of quality the return on investment on the system's quality will become reduced. Meaning that the return on investment on quality of software will be less than the effort or money we invest. Therefore, in most of the cases, after reaching at a reasonable level of quality we do not try to improve the quality of software any further. This phenomenon is shown in the figure below.



Software Background

Caper Jones a renounced practitioner and researcher in the filed of Software Engineering, had made immense research in software team productivity, software quality, software cost factors and other fields relate to software engineering. He made a company named Software Productivity Research in which they analyzed many projects and published the results in the form of books. Let's look at the summary of these results.

MC

He divided software related activities into about twenty-five different categories listed in the table below. They have analyzed around 10000 software projects to come up with such a categorization. But here to cut down the discussion we will only describe nine of them that are listed below.

MC

- Project Management
- Requirement Engineering
- Design
- Coding
- Testing
- Software Quality Assurance
- Software Configuration Management
- Software Integration and
- Rest of the activities

One thing to note here is that you cannot say that anyone of these activities is dominant among others in terms of effort putted into it. Here the point that we want to emphasize is that, though coding is very important but it is not more than 13-14% of the whole effort of software development. MC

MC

Fred Brook is a renowned software engineer; he wrote a great book related to software engineering named "A Mythical Man Month". He combined all his articles in this book. Here we will discuss one of his articles named "No Silver Bullet" which he included in the book.

An excerpt from "No Silver Bullet" – Fred Brooks

Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these we seek bullets of silver that can magically lay them to rest. The familiar software project has something of this character (at least as seen by the non-technical manager), usually innocent and straight forward, but capable of becoming a monster of missed schedules, blown budgets, and flawed projects. So we hear desperate cries for a silver bullet, something to make software costs drop as rapidly as computer hardware costs do. Scepticism is not pessimism, however. Although we see no startling breakthroughs, and indeed, such to be inconsistent with the nature of the software, many encouraging innovations are under way. A disciplined, consistent effort to develop, propagate and exploit them should indeed yield an order of magnitude improvement. There is no royal road, but there is a road. The first step towards the management of disease was replacement of demon theories and humours theories by the germ theory. The very first step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, *and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today.*

So, according to Fred Brook, in the eye of an unsophisticated manager software is like a giant. Sometimes it reveals as an unscheduled delay and sometimes it shows up in the form of cost overrun. To kill this giant the managers look for magical solutions. But unfortunately magic is not a reality. We do not have any magic to defeat this giant. There is only one solution and that is to follow a disciplined approach to build software. We can defeat the giant named software by using disciplined and engineered approach towards software development.

Therefore, *Software Engineering is nothing but a disciplined and systematic approach to software development*.

Now we will look at some of the activities involved in the course of software development. The activities involved in software development can broadly be divided into two major categories first is construction and second is management.

Software Development

Th construction activities are those that are directly related to the construction or development of the software. While the management activities are those that complement the process of construction in order to perform construction activities smoothly and effectively. A greater detail of the activities involved in the construction and management categories is presented below.

Construction

The construction activities are those that directly related to the development of software, e.g. gathering the requirements of the software, develop design, implement and test the software etc. Some of the major construction activities are listed below.

- Requirement Gathering
- Design Development
- Coding
- Testing

Management

Management activities are kind of umbrella activities that are used to smoothly and successfully perform the construction activities e.g. project planning, software quality assurance etc. Some of the major management activities are listed below.

- Project Planning and Management
- Configuration Management
- Software Quality Assurance
- Installation and Training

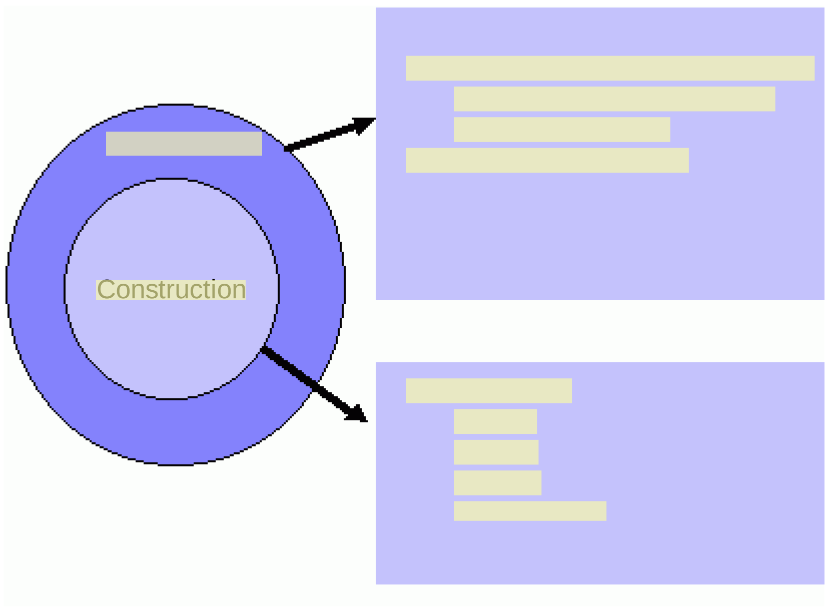


Figure1
Development Activities

As we have said earlier that management activities are kind of umbrella activities that surround the construction activities so that the construction process may proceed smoothly. This fact is empathized in the Figure1. The figure shows that construction is surrounded by management activities. That is, certain processes and rules govern all construction activities. These processes and rules are related to the management of the construction activities and not the construction itself.

A Software Engineering Framework

The software development organization must have special focus on quality while performing the software engineering activities. Based on this commitment to quality by the organization, a software engineering framework is proposed that is shown in Figure 2. The major components of this framework are described below.

1 Quality Focus: As we have said earlier, the given framework is based on the organizational commitment to quality. The quality focus demands that processes be defined for rational and timely development of software. And quality should be emphasized while executing these processes.

2 Processes: The processes are set of key process areas (KPAs) for effectively manage and deliver quality software in a cost effective manner. The processes define the tasks to be performed and the order in which they are to be performed. Every task has some deliverables and every deliverable should be delivered at a particular milestone.

level=3 Methods: Methods provide the technical "how-to's" to carryout these tasks. There could be more than one technique to perform a task and different techniques could be used in different situations.

level Tools: Tools provide automated or semi-automated support for software processes, methods, and quality control.

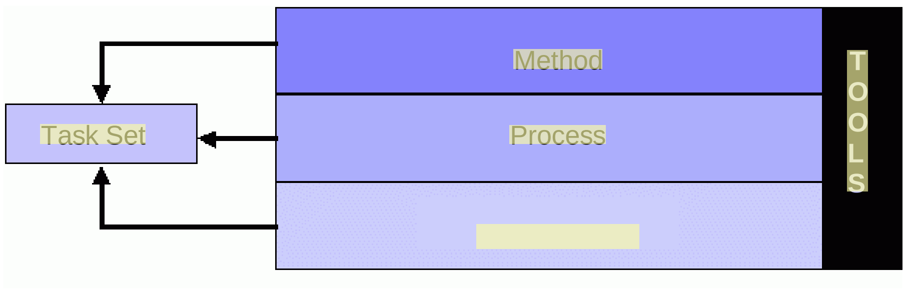


Figure 2

Software Engineering Framework

Software Development Loop

Let's now look at software engineering activities from a different perspective. Software development activities could be performed in a cyclic and that cycle is called software development loop which is shown in Figure3. The major stages of software development loop are described below.

Problem Definition: In this stage we determine what is the problem against which we are going to develop software. Here we try to completely comprehend the issues and requirements of the software system to build.

Technical Development: In this stage we try to find the solution of the problem on technical grounds and base our actual implementation on it. This is the stage where a new system is actually developed that solves the problem defined in the first stage.

Solution Integration: If there are already developed system(s) available with which our new system has to interact then those systems should also be the part of our new system. All those existing system(s) integrate with our new system at this stage.

Status Quo: After going through the previous three stages successfully, when we actually deployed the new system at the user site then that situation is called status quo. But once we get new requirements then we need to change the status quo.

After getting new requirements we perform all the steps in the software development loop again. The software developed through this process has the property that this could be evolved and integrated easily with the existing systems.

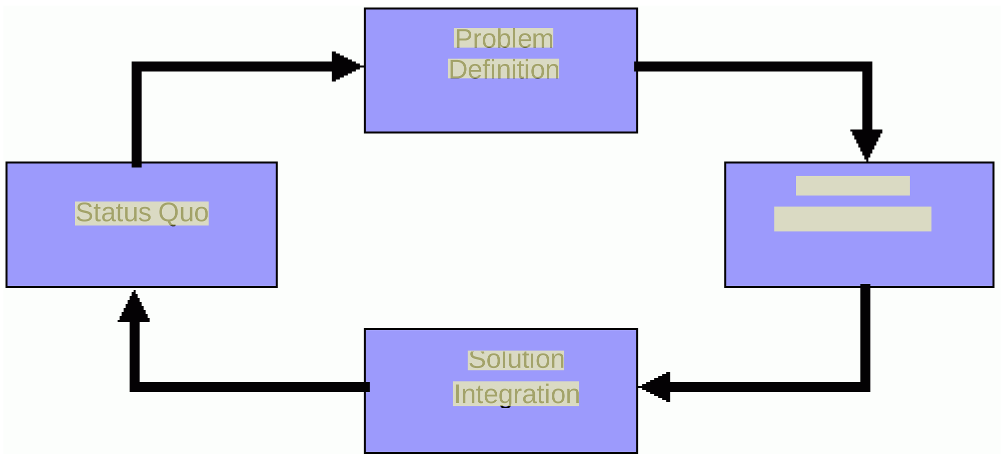


Figure3
Software Development Loop

Overview of the course contents

In the first course we studied the technical processes of software development to build industrial strength software. That includes requirement gathering and analysis, software design, coding, testing, and debugging. In this course our focus will be on the second part of Software Engineering, that is, the activities related to managing the technical development. This course will therefore include the following topics:

1. Software development process
2. Software process models
3. Project Management Concepts
4. Software Project Planning
5. Risk Analysis and Management
6. Project Schedules and Tracking
7. Software Quality Assurance
8. Software Configuration Management
9. Software Process and Project Metrics
10. Requirement Engineering Processes
11. Verification and Validation
12. Process Improvement
13. Legacy Systems
14. Software Change
15. Software Re-engineering

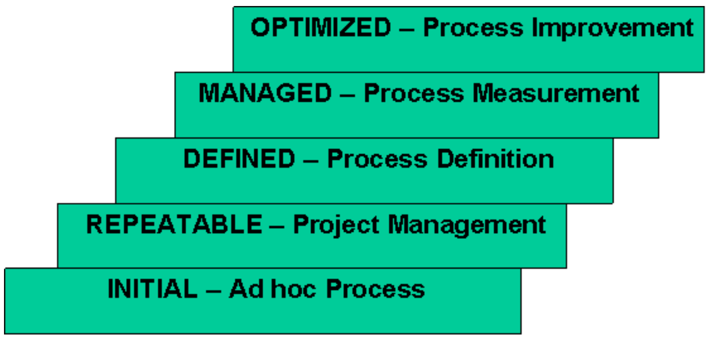
Lecture No. 2

Software Process

A software process is a road map that helps you create a timely, high quality result. It is the way we produce software and it provides stability and control. Each process defines certain deliverables known as the work products. These include programs, documents, and data produced as a consequence of the software engineering activities.

Process Maturity and CMM

The Software Engineering Institute (SEI) has developed a framework to judge the process maturity level of an organization. This framework is known as the Capability Maturity Model (CMM). This framework has 5 different levels and an organization is placed into one of these 5 levels. The following figure shows the CMM framework.


OPTIMIZED – Process Improvement
MANAGED – Process Measurement
DEFINED – Process Definition
REPEATABLE – Project Management
INITIAL – Ad hoc Process

These levels are briefly described as follows"

1. Level 1 – Initial: The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends upon individual effort. By default every organization would be at level 1.
2. Level 2 – Repeatable: Basic project management processes are established to track cost, schedule, and functionality. The necessary project discipline is in place to repeat earlier successes on projects with similar applications.
3. Level 3 – Defined: The software process for both management and engineering activities is documented, standardized, and integrated into an organizational software process. All projects use a documented and approved version of the organization's process for developing and supporting software.
4. Level 4 – Managed Detailed measures for software process and product quality are controlled. Both the software process and products are quantitatively understood and controlled using detailed measures.
5. Level 5 – Optimizing: Continuous process improvement is enabled by qualitative feedback from the process and from testing innovative ideas and technologies.

SEI has associated key process areas with each maturity level. The KPAs describe those software engineering functions that must be present to satisfy good practice at a particular level. Each KPA is described by identifying the following characteristics:

1. Goals: the overall objectives that the KPA must achieve.
2. Commitments: requirements imposed on the organization that must be met to achieve the goals or provide proof of intent to comply with the goals.
3. Abilities: those things that must be in place – organizationally and technically – to enable the organization to meet the commitments.
4. Activities: the specific tasks required to achieve the KPA function
5. Methods for monitoring implementation: the manner in which the activities are monitored as they are put into place.
6. Methods for verifying implementation: the manner in which proper practice for the KPA can be verified.

Each of the KPA is defined by a set of practices that contribute to satisfying its goals. The key practices are policies, procedures, and activities that must occur before a key process area has been fully instituted.

The following table summarizes the KPAs defined for each level.

long question in mid term

Level	KPAs
1	No KPA is defined as organizations at this level follow ad-hoc processes
2	• Software Configuration Management • Software Quality Assurance • Software subcontract Management • Software project tracking and oversight • Software project planning • Requirement management
3	• Peer reviews • Inter-group coordination • Software product Engineering • Integrated software management • Training program • Organization process management • Organization process focus
4	• Software quality management • Quantitative process management
5	• Process change management • Technology change management • Defect prevention

Lecture No. 3

Software Lifecycle Models

Recalling from our first course, a software system passes through the following phases:

Mid Term:

1. Vision – focus on *why* ✓
2. Definition – focus on *what* ✓
3. Development – focus on *how*
4. Maintenance – focus on *change*

During these phases, a number of activities are performed. A lifecycle model is a series of steps through which the product progresses. These include requirements phase, specification phase, design phase, implementation phase, integration phase, maintenance phase, and retirement. Software Development Lifecycle Models depict the way you organize your activities.

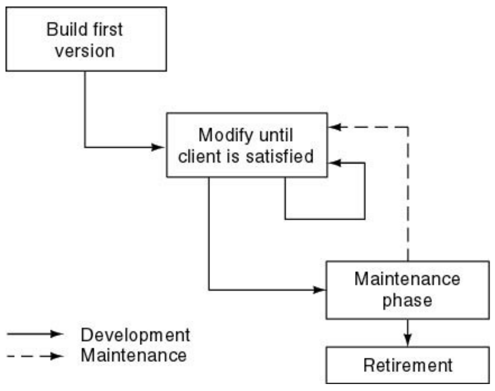
There are a number of Software Development Lifecycle Models, each having its strengths and weaknesses and suitable in different situations and project types. The list of models includes the following:

- Build-and-fix model
- Waterfall model
- Rapid prototyping model
- Incremental model
- Extreme programming
- Synchronize-and-stabilize model
- Spiral model
- Object-oriented life-cycle models

In the following sections we shall study these models in detail and discuss their strengths and weaknesses.

Build and Fix Model

This model is depicted in the following diagram:

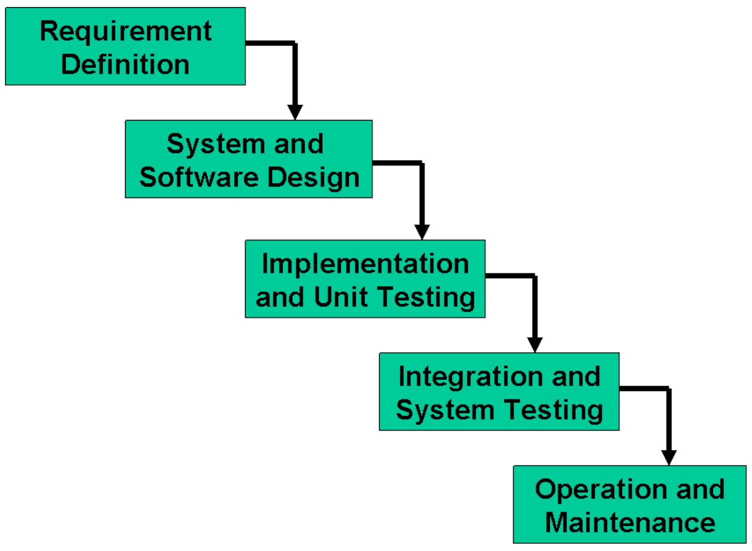


✓

It is unfortunate that many products are developed using what is known as the build-and-fix model. In this model the product is constructed without specification or any attempt at design. The developers simply build a product that is reworked as many times as necessary to satisfy the client. This model may work for small projects but is totally unsatisfactory for products of any reasonable size. The co t of build-and fix is actually far greater than the cost of properly specified and carefully designed product.
Main enance of the product can be extremely in the absence of any documentation. mc

Waterfall Model

The first published model of the software development process was derived from other engineering processes. Because of the cascade from one phase to another, this model is known as the waterfall model. This model is also known as linear sequential model. This model is depicted in the following diagram.


Requirement Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

The principal stages of the model map directly onto fundamental development activities.

It suggests a systematic, sequential approach to software development that begins at the system level and progresses through the analysis, design, coding, testing, and maintenance.

In the literature, people have identified from 5 to 8 stages of software development.

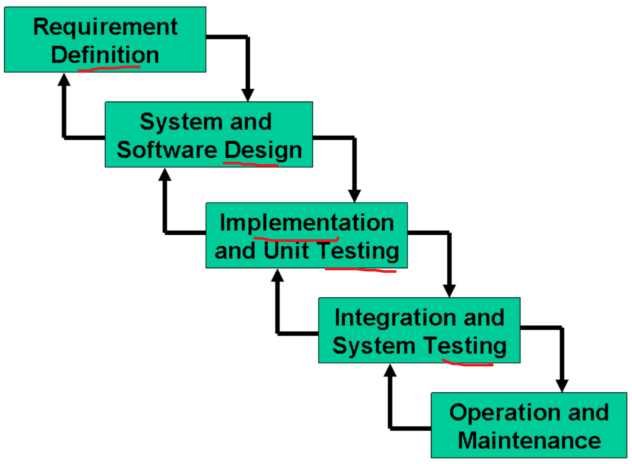
The five stages above are as follows:

1. Requirement Analysis and Definition: What - The systems services, constraints and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.
2. System and Software Design: How – The system design process partitions the requirements to either hardware of software systems. It establishes and overall system architecture. Software design involves fundamental system abstractions and their relationships.

3. Implementation and Unit Testing: - How – During this stage the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specifications.
4. Integration and system testing: The individual program unit or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
5. Operation and Maintenance: Normally this is the longest phase of the software life cycle. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life-cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

In principle, the result of each phase is one or more documents which are approved. No phase is complete until the documentation for that phase has been completed and products of that phase have been approved. The following phase should not start until the previous phase has finished. mcq

Real projects rarely follow the sequential flow that the model proposes. In general these phases overlap and feed information to each other. Hence there should be an element of iteration and feedback. A mistake caught any stage should be referred back to the source and all the subsequent stages need to be revisited and corresponding documents should be updated accordingly. This feedback path is shown in the following diagram.


Requirement Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

Because of the costs of producing and approving documents, iterations are costly and require significant rework.

MC The Waterfall Model is a documentation-driven model. It therefore generates complete and comprehensive documentation and hence makes the maintenance task much easier. It however suffers from the fact that the client feedback is received when the product is finally delivered and hence any errors in the requirement specification are not discovered until the product is sent to the client after completion. This therefore has major time and cost related consequences. MC MC

Rapid Prototyping Model

3

The Rapid Prototyping Model is used to overcome issues related to understanding and capturing of user requirements. In this model a mock-up application is created "rapidly" to solicit feedback from the user. Once the user requirements are captured in the prototype to the satisfaction of the user, a proper requirement specification document is developed and the product is developed from scratch.

An essential aspect of rapid prototype is embedded in the word "rapid". The developer should endeavour to construct the prototype as quickly as possible to speedup the software development process. It must always be kept in mind that the sole purpose of the
MC rapid prototype is to capture the client's needs; once this has been determined, the rapid prototype is effectively discarded. For this reason, the internal structure of the rapid prototype is not relevant.

Integrating the Waterfall and Rapid Prototyping Models

Despite the many successes of the waterfall model, it has a major drawback in that the delivered product may not fulfil the client's needs. One solution to this is to combine rapid prototyping with the waterfall model. In this approach, rapid prototyping can be used as a requirement gathering technique which would then be followed by the activities performed in the waterfall model.

Masters

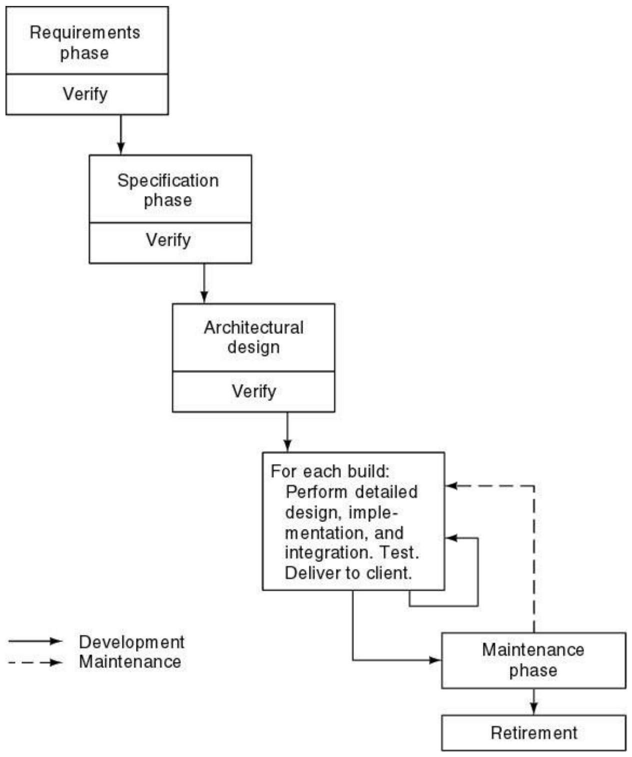
4

Lecture No. 4

Incremental Models

As discussed above, the major drawbacks of the waterfall model are due to the fact that the entire product is developed and delivered to the client in one package. This results in delayed feedback ~~from~~ the client. Because of the long elapsed time, a huge new investment of time and money may be required to fix any errors of omission or commission or to accommodate any new requirements cropping up during this period. This may render the product as unusable. Incremental model may be used to overcome these issues.

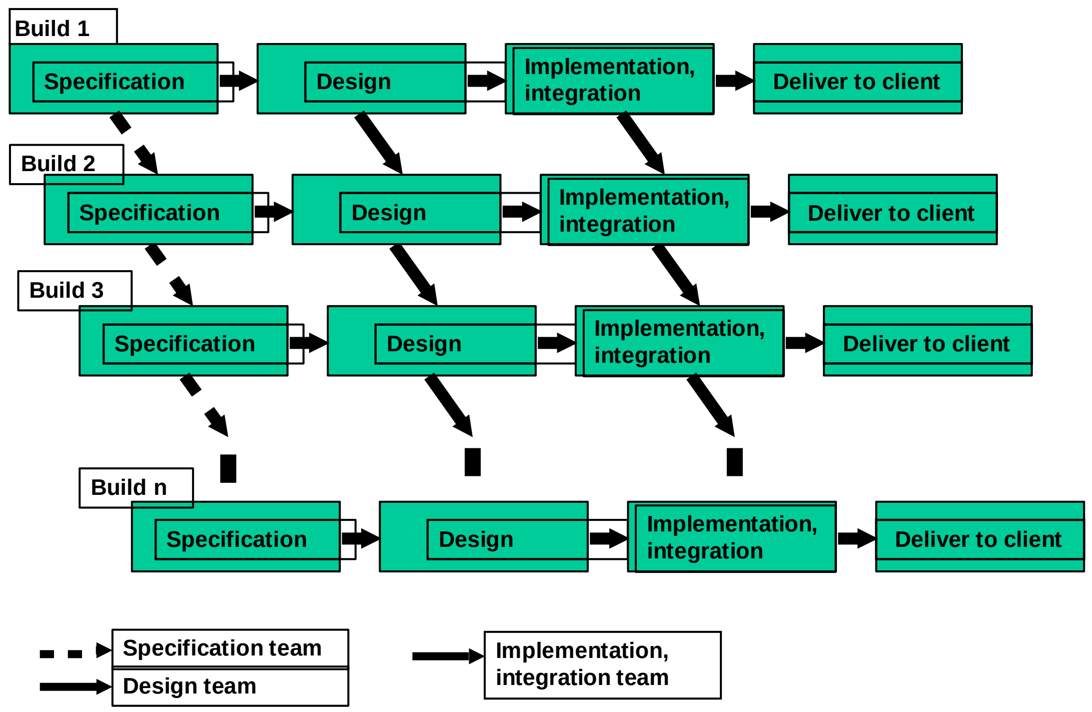
In the incremental models, as opposed to the waterfall model, the product is partitioned into smaller pieces which are then built and delivered to the client in increments at regular intervals. Since each piece is much smaller than the whole, it can be built and sent to the client quickly. This results in quick feedback from the client and any requirement related errors or changes can be incorporated at a much lesser cost. It is therefore less traumatic as compared to the waterfall model. It also required smaller capital outlay and yield a



rapid return on investment. However, this model needs and open architecture to allow integration of subsequent builds to yield the bigger product. A number of variations are used in object-oriented life cycle models.

There are two fundamental approaches to the incremental development. In the first case, the requirements, specifications, and architectural design for the whole product are completed before implementation of the various builds commences.

In a more risky version, once the user requirements have been elicited, the specifications of the first build are drawn up. When this has been completed, the specification team



turns to the specification of the second build while the design team designs the first build. Thus the various builds are constructed in parallel, with each team making use of the information gained in the all the previous builds.

This approach incurs the risk that the resulting build will not fit together and hence requires careful monitoring.

Rapid Application Development (RAD) 5

Rapid application development is another form of incremental model. It is a high speed adaptation of the linear sequential model in which fully functional system in a very short time (2-3 months). This model is only applicable in the projects where requirements are MC well understood and project scope is constrained. Because of this reason it is used primarily for information systems.

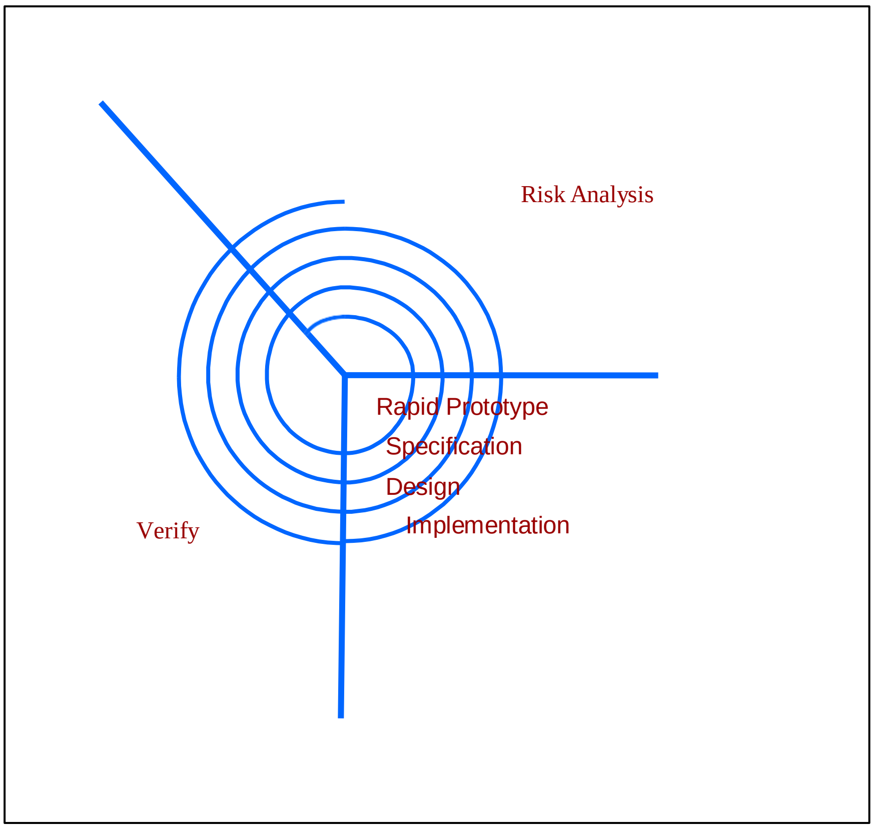
Synchronize 6 and Stabilize Model

This is yet another form of incremental model adopted by Microsoft. In this model, during the requirements analysis interviews of potential customers are conducted and requirements document is developed. Once these requirements have been captured, specifications are drawn up. The project is then divided into 3 or 4 builds. Each build is carried out by small teams working in parallel. At the end of each day the code is synchronized (test and debug and at the end of the build it is stabilized by freezing the build and removing any remaining defects. Because of the synchronizations, components always work together. The presence of an executable provides early insights into operation of product.

Spiral Model

This model was developed by Barry Boehm. The main idea of this model is to avert risk as there is always an element of risk in development of softwa e. For example, key personnel may resign at a critical juncture, the manufacturer of the software development may go bankrupt, etc.

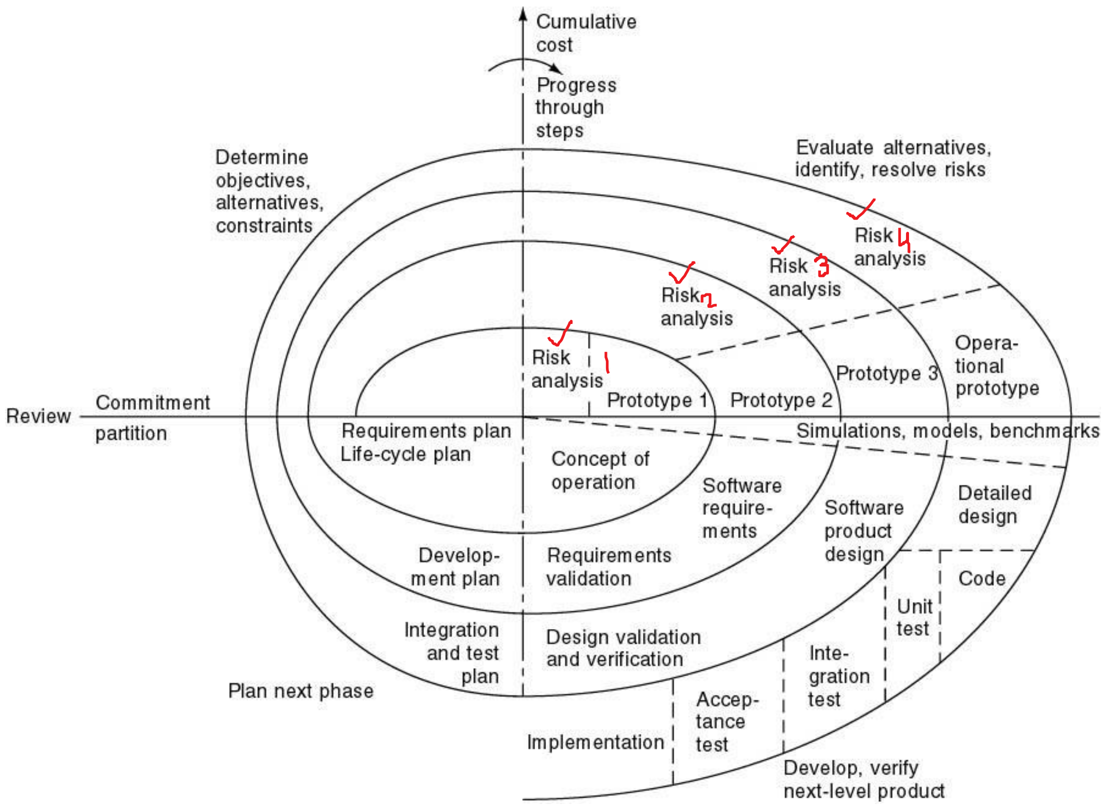
In its simplified form, the Spiral Model is Waterfall model plus risk analysis. In this case each stage is preceded by identification of alternatives and risk analysis and is then followed by evaluation and planning for the next phase. If risks cannot be resolved, project is immediately terminated. This is depicted in the following diagram.



As can be seen, a Spiral Model has two dimensions. Radial dimension represents the cumulative cost to date and the angular dimension represents the progress through the spiral. Each phase begins by determining objectives of that phase and at each phase a new process model may be followed.

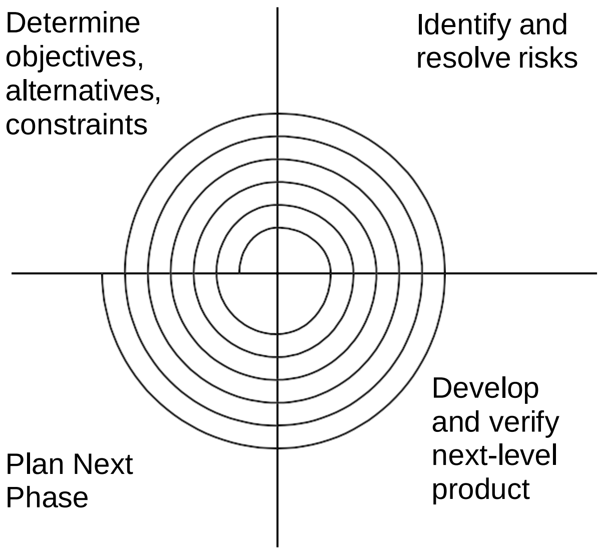
MC

A full version of the Spiral Model is shown below:



MC

The main strength of the Spiral Model comes from the fact that it is very sensitive to the risk. Because of the spiral nature of development it is easy to judge how much to test and there is no distinction between development and maintenance. It however can only be used for large-scale software development and that too for internal (in-house) software only. MC


Determine objectives, alternatives, constraints
Identify and resolve risks
Develop and verify next-level product
Plan Next Phase

Masters

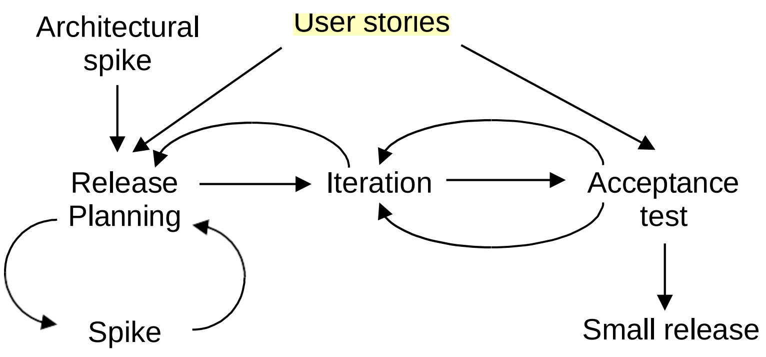
Lecture No. 5

Object-Oriented Lifecycle Models

Object-oriented lifecycle models appreciate the need for iteration within and between phases. There are a number of these models. All of these models incorporate some form of iteration, parallelism, and incremental development.

eXtreme Programming

It is a somewhat controversial new approach. In this approach user requirements are captured through ~~stories which~~ are the scenarios presenting the features needed by the client? Estimate for duration and cost of each story is then carried out. Stories for the next build are selected. Then each build is divided into tasks. Test cases for task are drawn up first before and development and continuous testing is performed throughout the development process.



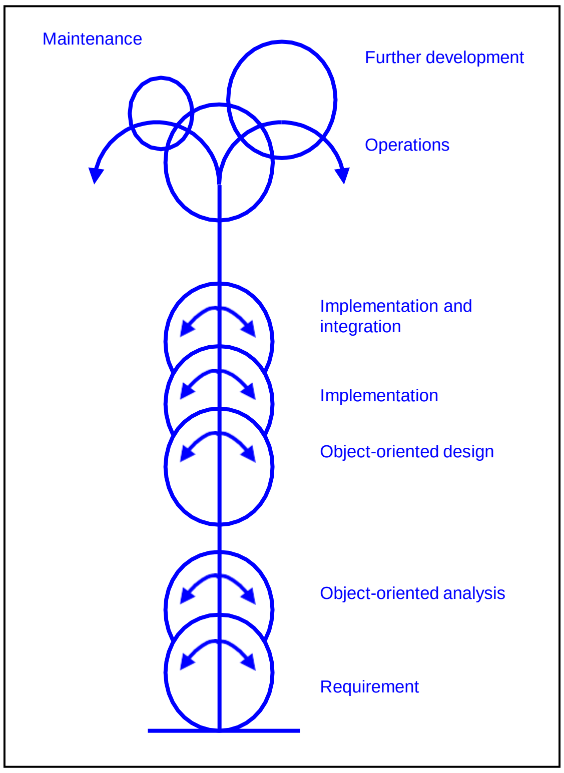
One very important feature of eXtreme programming is the concept of pair programming. MC In this, a team of two developers develop the software, working in team as a pair to the extent that they even share a single computer.

In eXtereme Programming model, computers are put in center of large room lined with cubicles and client representative is always present. One very important restriction imposed in the model is that no team is allowed to work overtime for 2 successive weeks MC

XP has had some successes. It is good when requirements are vague or changing and the overall scope of the project is limited. It is however too soon to evaluate XP.

Fountain Model

Fountain model is another object-oriented lifecycle model. This is depicted in the following diagram.



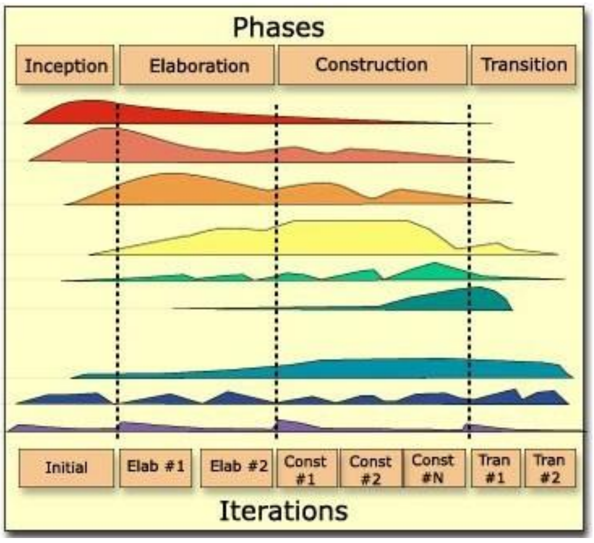
In this model the circles representing the various phases overlap, explicitly representing an overlap between activities. The arrows within a phase represent iteration with~~in the~~ phase. The maintenance cycle is smaller, to symbolize reduced maintenance effort when the object oriented paradigm is used.

10 Rational Unified Process (RUP)

Rational Unified Process is very closely associated with UML and Krutchen's architectural model.

In this model a software product is designed and built in a succession of incremental iterations. It incorporates early testing and validation of design ideas and early risk mitigation. The horizontal dimension represents the *dynamic aspect* of the process. This includes cycles, phases, iterations, and milestones. The vertical dimension represents the *static aspect* of the process described in terms of process components which include activities, disciplines, artifacts, and roles. The process emphasizes that during development, all activities are performed in parallel, however, and at a given time one activity may have more emphasis than the other.

The following figure depicting RUP is taken from Krutchen's paper.

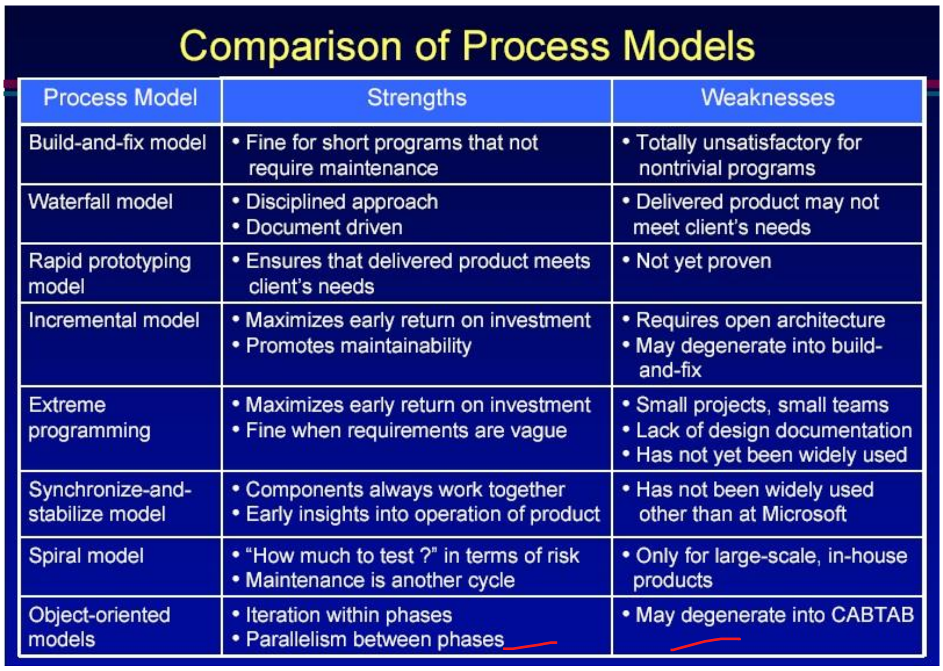


Comparison of Lifecycle Models

assignment question 2023 + Mid term question

√Q

As discussed above, each lifecycle model has some strengths and weaknesses. These are summarized in the following table:



Comparison of Process Models

Process Model	Strengths	Weaknesses
1 Build-and-fix model	• Fine for short programs that not require maintenance	• Totally unsatisfactory for nontrivial programs
2 Waterfall model	• Disciplined approach • Document driven	• Delivered product may not meet client's needs
3 Rapid prototyping model	• Ensures that delivered product meets client's needs	• Not yet proven
4 Incremental model	• Maximizes early return on investment • Promotes maintainability	• Requires open architecture • May degenerate into build-and-fix
5 Extreme programming	• Maximizes early return on investment • Fine when requirements are vague	• Small projects, small teams • Lack of design documentation • Has not yet been widely used
6 Synchronize-and-stabilize model	• Components always work together • Early insights into operation of product	• Has not been widely used other than at Microsoft
7 Spiral model	• "How much to test ?" in terms of risk • Maintenance is another cycle	• Only for large-scale, in-house products
8 Object-oriented models	• Iteration within phases • Parallelism between phases	• May degenerate into CABTAB

The criteria to be used for deciding on a model include the organization, its management, skills of the employees, and the nature of the product. No single model may fulfill the needs in a given situatio . It may therefore be best to devise a lifecycle model tuned to your own needs by creating a "Mix-and-match" life-cycle model.

Quality Assurance and Documentation

It may be noted that there is no separate QA or documentation phase. QA is an activity performed throughout software production. I involves verification and validation.

Verification is performed at the end of each phase whereas validation is performed before delivering the product to the client.

Similarly, every phase must be fully documented before starting the next phase. It is important to note that postponed documentation may never be completed as the responsible individual may leave. Documentation is important as the product is constantly changing—we need the documentation to do this. The design (for example) will be modified during development, but the original designers may not be available to document it.

The following table shows the QA and documentation activities associated with each stage.

Phase	Documents	QA
1 Requirement Definition	• Rapid prototype, or • Requirements document	• Rapid prototype • Reviews
2 Functional Specification	• Specification document (specifications) • Software Product Management Plan	• Traceability • FS Review • Check the SPMP
3 Design	• Architectural Design • Detailed Design	• Traceability • Review
4 Coding	• Source code • Test cases	• Traceability • Review • Testing
5 Integration	• Source code • Test cases	• Integration testing • Acceptance testing
6 Maintenance	• Change record • Regression test cases	• Regression testing

Masters

Lecture No. 6

Software Project Management Concepts

Software project management is a very important activity for successful projects. In fact, in an organization at CMM Level basic project management processes are established to track cost, schedule, and functionality. That is, it is characterized by basic project management practices. It also implies that without project management not much can be achieved. Capers Jones, in his book on Software Best Practices, notes that, for the projects they have analyzed, good project management was associated with 100% of the successful project and bad project management was associated with 100% of the unsuccessful projects. Therefore, understanding of good project management principles and practices is essential for all project manag rs and software engineers.

Software project management involves that planning, organization, monitoring, and control of the people and the processes.

Software Project Management: Factors that influence results

The first step towards better project management is the comprehension of the factors that influence results of a project. Among these, the most important factors are:

Q: What are the duties/responsibilities of project manager.

Q In

1 – Project size

As the project size increases, the complexity of the problem also increases and therefore its management also becomes more difficult.

2 – Delivery deadline

Delivery deadline directly influences the resources and quality. With a realistic deadline, chances of delivering the product with high quality and reasonable resources increase tremendously as 3 compared to an unrealistic deadline. So a project manager has to first determine a realistic and reasonable deadline and then monitor the project progress and ensure timely delivery.

– Budgets and costs

A project manager is responsible for ensuring delivery of the project within the allocated budget and schedule. A good estimate of budget, cost and schedule is essential for any successful project. It is therefore imperative that the project manager understand and learns the techniques and principle needed to develop these estimates.

4 – Application domain

Application domain also plays an important role in the success of a project. The chances of success of a project in a well-known application domain would be much better than of a project in a relatively unknown domain. The project manager thus needs to implement measures to handle unforeseen problems that may arise during the project lifecycle.

5 – Technology to be implemented

Technology also plays a very significant role in the success or failure of a project. One the one hand, a new "state-of-the-art" technology may increase the productivity of the team and quality of the product. On the other hand, it may prove to be unstable and hence prove to be difficult to handle. Resultantly, it may totally blow you off the track. So, the project manager should be careful in choosing the implementation technology and must take proper safeguard measures.

– System constraints 6

The non-functional requirement or system constraints specify the conditions and the restrictions imposed on the system. A system that fulfils all its functional requirements but does not satisfy the non-functional requirements would be rejected by the user.

7 – User requirements

A system has to satisfy its user requirements. Failing to do so would render this system unusable.

8 – Available resources

A project has to be developed using the available resources who know the domain as well as the technology. The project manager has to ensure that the required number of resources with appropriate skill-set is available to the project.

Project Management Concerns

In order to plan and run a project successfully, a project manager needs to worry about the following issues: Q In

1. Product quality: what would be the acceptable quality level for this particular project and how could it be ensured?
2. Risk assessment: what would be the potential problems that could jeopardize the project and how could they be mitigated?
3. Measurement: how could the size, productivity, quality and other important factors be measured and benchmarked?
4. Cost estimation: how could cost of the project be estimated?
5. Project schedule: how could the schedule for the project be computed and estimated?
6. Customer communication: what kind of communication with the customer would be needed and how could it be established and maintained consistently?
7. Staffing: how many people with what kind of resources would be needed and how that requirement could be fulfilled?
8. Other resources: what other hardware and software resources would be needed for the project?
9. Project monitoring: how the progress of the project could be monitored?

Thorough understanding and appreciation of these issues leads to the quest for finding satisfactory answers to these problems and improves the chances for success of a project.

Why Projects Fail?

A project manager is tasked to ensure the successful development of a product. Success cannot be attained without understanding the reasons for failure. The main reasons for the failure of software projects are:

1. changing customer requirements
2. ambiguous/incomplete requirements

3. unrealistic deadline
4. an honest underestimate of effort
5. predictable and/or unpredictable risks
6. technical difficulties
7. miscommunication among project staff
8. failure in project management

The first two points relate to good requirement engineering practices. Unstable user requirements and continuous requirement creep has been identified as the top most reason for project failure. Ambiguous and incomplete requirements lead to undesirable product that is rejected by the user.

As discussed earlier, delivery deadline directly influences the resources and quality. With a realistic deadline, chances of delivering the product with high quality and reasonable resources increase tremendously as compared to an unrealistic deadline. An unrealistic deadline could be enforced by the management or the client or it could be due to error in estimation. In both these cases it often results in disaster for the project.

A project manager who is not prepared and without a contingency plan for all sorts of predictable and unpredictable risks would put the project in jeopardy if such a risk should happen. Risk assessment and anticipation of technical and other difficulties allows the project manager to cope with these situations.

Miscommunication among the project staff is another very important reason for project failure. Lack of proper coordination and communication in a project results in wastage of resources and chaos.

The Management Spectrum

Effective project management focuses on four aspects of the project known as the 4 P's. These are: people, product, process, and project.

People
Software development is a highly people intensive activity. In this business, the software factory comprises of the people working there. Hence taking care of the first P, that is people, should take the highest priority on a project manager's agenda.

Product
The product is the outcome of the project. It includes all kinds of the software systems. No meaningful planning for a project can be carried-out until all the dimensions of the product including its functional as well as non-functional requirements are understood and all technical and management constraints are identified.

Process
Once the product objectives and scope have been determined, a proper software development process and lifecycle model must be chosen to identify the required work products and define the milestones in order to ensure streamlined development activities. It includes the set of all the framework activities and software engineering tasks to get the job done.

Project

A project comprises of all work the required to make the product a reality. In order to avoid failure, a project manager and software engineer is required to build the software product in a controlled and organized fashion and run it like other projects found in more concrete domains.

We now discuss these 4 in more detail.

People

In a study published by IEEE, the project team was identified by the senior executives as the most important contributor to a successful software project. However, unfortunately, people are often taken for granted and do no get the attention and focus they deserve. There are a number of players that participate in software process and influence the outcome of the project. These include senior managers, project (technical) managers, practitioners, customers, and end-users. Senior managers define the business vision whereas the project managers plan, motivate, organize and control the practitioners who work to develop the software product. To be effective, the project team must be organized to use each individual to the best of his/her abilities. This job is carried out by the team leader.

Team Leader

Project management is a people intensive activity. It needs the right mix of people skills. Therefore, competent practitioners often make poor team leaders.

Leaders should apply a problem solving management style. That is, a project manager should concentrate on understanding the problem to be solved, managing the flow of ideas, and at the same time, letting everyone on the team know that quality counts and that it will not be compromised.
MOI model of leadership developed by Weinberg suggest that a leadership needs Motivation, Organization, and Innovation.

Motivation is the ability to encourage technical people to produce to their best. **Organization** is the ability to mold the existing processes (or invent new ones) that will enable the initial concept to be translated into a final product, and **Idea or Innovation** is the ability to encourage people to create and feel creative.

It is suggested that successful project managers apply a problem solving management style. This involves developing an understanding of the problem and motivating the team to generate ideas to solve the problem.

Edgemon suggests that the following characteristics are needed to become an effective project manager:

- Problem Solving
  - Should be able to diagnose technical and organizational issues and be willing to change direction if needed.
- Managerial Identity
  - Must have the confidence to take control when necessary

- Achievement
  - Reward initiative (controlled risk taking) and accomplishment
- Influence and team building
  - Must remain under control in high stress conditions. Should be able to read signals and address peoples' needs.

DeMarco says that a good leader possesses the following four characteristics:

– Heart: the leader should have a big heart.
– Nose: the leader should have good nose to spot the trouble and bad smell in the project.
– Gut: the leader should have the ability to make quick decision on gut feeling.
– Soul: the leader should be the soul of the team.

If analyzed closely, all these researchers seem to say essentially the same thing and they actually complement each other's point of view.

Masters

Lecture No. 7

The Software Team

There are many possible organizational structures. In order to identify the most suitable structure, the following factors must be considered:

- the difficulty of the problem to be solved
- the size of the resultant program(s) in lines of code or function points
- the time that the team will stay together (team lifetime)
- the degree to which the problem can be modularized
- the required quality and reliability of the system to be built
- the rigidity of the delivery date
- the degree of sociability (communication) required for the project

Constantine suggests that teams could be organized in the following generic structural paradigms:

mcqs

- **closed paradigm**—structures a team along a traditional hierarchy of authority
- 2 **random paradigm**—structures a team loosely and depends on individual initiative of the team members
- 3 **open paradigm**— ttempts to structure a team in a manner that achieves some of the controls associated with the closed paradigm but also much of the innovation that occurs when using the random paradig
- 4 **synchronous paradigm**—relies on the natural compartmentalization of a problem and organizes team members to work on pieces of the problem with little active communication among themselves

Mantei suggests the following three generic team organizations:

- Democratic decentralized (DD)

In this organization there is no permanent leader and task coordinators are appointed for short duration. Decisions on problems and approach are made by group consensus and communication among team is horizontal.

- Controlled decentralized (CD)

In CD, there is a defined leader who coordinates specific tasks. However, problem solving remains a group activity and communication among subgroups and individuals is horizontal. Vertical communication along the control hierarchy also occurs.

- Controlled centralized (CC)

In a Controlled Centralized structure, top level problem solving and internal team coordination are managed by the team leader and communication between the leader and team members is vertical.

Centralized structures complete tasks faster and are most useful for handling simple problems. On the other hand, decentralized teams generate more and better solutions than individuals and are most useful for complex problems
For the team morale point of view, DD is better.

Coordination and Communication Issues

Lack of coordination results in confusion and uncertainty. On the other hand, performance is inversely proportional to the amount of communication and hence too much communication and coordination is also not healthy for the project. Very large projects are best addressed with CC or CD when sub-grouping can be easily accommodated.

Kraul and Steeter categorize the project coordination techniques as follows:

- Formal, impersonal approaches

In these approaches, coordination is achieved through impersonal and formal mechanism such as SE documents, technical memos, schedules, error tracking reports.

- Formal, interpersonal procedures

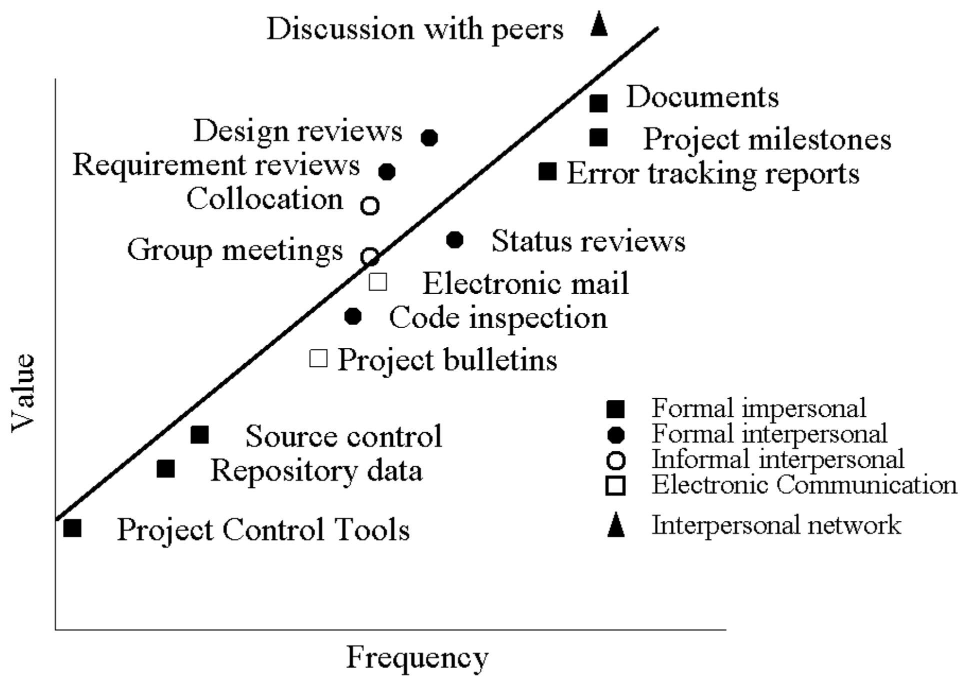
In this case, the approaches are interpersonal and formal. These include QA activities, design and code reviews, and status meetings.

- Informal, interpersonal procedures

This approach employs informal interpersonal procedures and includes group meetings and collocating different groups together.

- Electronic communication includes emails and bulletin boards.
- Interpersonal networking includes informal discussions with group member

The effectiveness of these approaches has been summarized in the following diagram:



Techniques that fall above the regression line yield more value to use ratio as compared to the ones below the line.

The Product: Defining the Problem

In order to develop an estimate and plan for the project, the scope of the problem must be established. This includes context, information objectives, and function and performance requirements. The estimate and plan is then developed by decomposing the problem and establishing a functional partitioning

The Process

The next step is to decide which process model to pick. The project manager has to look at the characteristics of the product to be built and the project environment. For examples, for a relatively small project that is similar to past efforts, degree of uncertainty is minimized and hence Waterfall or linear sequential model could be used. For tight timelines, heavily compartmentalized, and known domain, RAD model would be more suitable. Projects with large functionality, quick turn around time are best developed incrementally and for a project in which requirements are uncertain, prototyping model will be more suitable.

Lecture No. 8

The Project Management

As discussed earlier, a project manager must understand what can go wrong and how to do it right. Reel has defined a 5 step process to improve the chances of success. These are: Q In

– Start on the right foot: this is accomplished by putting in the required effort to understand the problem, set realistic objectives, build the right team, and provide the needed infrastructure.
– Maintain momentum: many projects, after starting on the right, loose focus and momentum. The initial momentum must be maintained till the very end.
– Track progress: no planning is useful if the progress is not tracked. Tracking ensures timely delivery and remedial action, if needed, in a suitable manner.
– Make smart decisions
– Conduct a postmortem analysis: in order to learn from the mistakes and improve the process continuously, a project postmortem must be conducted.

W5HH Principle

Q In

Barry Boehm has suggested a systematic approach to project management. It is known as the WWWWWHH principle. It comprises of 7 questions. Finding the answers to these 7 questions is essentially all a project manager has to do. These are:

- **WHY** is the system being developed?
- **WHAT** will be done?
- By **WHEN**?
- **WHO** is responsible for a function?
- **WHERE** they are organizationally located?
- **HOW** will the job be done technically and managerially?
- **HOW MUCH** of each resource (e.g., people, software, tools, database) will be needed?

Boehm's W^{5}HH principle is applicable, regardless of the size and complexity of the project and provide excellent planning outline.

Critical Practices

The Airlie Council has developed a list of critical success practices that must be present for successful project management. These are:

- Formal risk analysis
- Empirical cost and schedule estimation
- Metrics-based project management
- Earned value tracking
- Defect tracking against quality targets
- People aware project management

Finding the solution to these practices is the key to successful projects. We'll therefore spend a considerable amount of time in elaborating these practices.

Lecture No. 9

Software Size Estimation

The size of the software needs to be estimated to figure out the time needed in terms of calendar and man months as well as the number and type of resources required carrying out the job. The time and resources estimation eventually plays a significant role in determining the cost of the project.

Most organizations use their previous experience to estimate the size and hence the resource and time requirements for the project. If not quantified, this estimate is subjective and is as good as the person who is conducting this exercise. At times this makes it highly contentious. It is therefore imperative for a government organization to adopt an estimation mechanism that is:

1. Objective in nature.
2. It should be an acceptable standard with wide spread use and acceptance level.
3. It should serve as a single yardstick to measure and make comparisons.
4. Must be based upon a deliverable that is meaningful to the intended audience.
5. It should be independent of the tool and technology used for the developing the software.

A number of techniques and tools can be used in estimating the size of the software. These include:

1. Lines of code (LOC)
2. Number of objects
3. Number of GUIs
4. Number of document pages
5. Functional points (FP)

Comparison of LOC and FPA Q

Out of these 5, the two most widely used metrics for the measurement of software size are FP and LOC. LOC metric suffer from the following shortcomings:

1. There are a number of questions regarding the definition for lines of code. These include:
    a. Whether to count physical line or logical lines?
    b. What type of lines should be counted? For example, should the comments, data definitions, and blank lines be counted or not?
2. LOC is heavily dependent upon the individual programming style.
3. It is dependent upon the technology and hence it is difficult to compare applications developed in two different languages. This is true for even seemingly very close languages like in C++ and Java.
4. If a mixture of languages and tools is used then the comparison is even more difficult. For example, it is not possible to compare a project that delivers a 100,000-line mixture of Assembly, C++, SQL and Visual Basic to one that delivers 100,000 lines of COBOL.

FP measures the size of the functionality provided by the software. The functionally is measured as a function of the data and the operations performed on that data. The measure is independent of the tool and technology used and hence provides a consistent measure for comparison between various organizations and projects.

The biggest advantage of FP over LOC is that LOC can be counted only AFTER the code has been developed while FP can be co nted even at the requirement phase and hence can be used for planning and estimation while the LOC cannot be used for this purpose.

Another major distinction between the FP and LOC is that the LOC measures the application from a developer's perspective while the FP is a measure of the size of the functionality from the user's perspective. The user's view, as defined by IFPUG, is as follows:

> A *user view* is a description of the business functions and is approved by the user. It represents a formal description of the user's business needs in the user's language. It can vary in physical form (e.g., catalog of transactions, proposals, requirements document, external specifications, detailed specifications, user handbook). Developers translate the user information into information technology language in order to provide a solution. Function point counts the application size from the user's point of view. It is accomplished using the information in a language that is common to both user(s) and developers.

Therefore, Function Point Analysis measures the size of the functionality delivered and used by the end user as opposed to the volume of the artifacts and code.

	Assembler Version	Ada Version	Difference
Source Code Size	100,000	25,000	-75,000
Activity - in person months			
Requirements	10	10	0
Design	25	25	0
Coding	100	20	-80
Documentation	15	15	0
Integration and Testing	25	15	-10
Management	25	15	-10
Total Effort	200	100	-100
Total Cost	$1,000,000	$500,000	-$500,000
Cost Per Line	$10	$20	$10
Lines Per Person-Month	500	250	-250

The Paradox of Reversed Productivity for High-Level Languages

Consider the following example:
In this example, it is assumed that the same functionality is implemented in Assembly and Ada. As coding in Assembly is much more difficult and time consuming as compared to Ada, it takes more time and it is also lengthy. Because there is a huge difference in the code size in terms of Lines of Code, th cost per line in case of Assembly is much less as compared to Ada. Hence coding in Assembly appears to be more cost effective than Ada while in reality it is not. This is a paradox!

Function Point Analysis - A Brief History and Usage

In the mid 70's, IBM felt the need to establish a more effective and better measure of system size to predict the delivery of software. It commissioned Allan Albrecht to lead this effort. As a result h developed this approach which today known as the Function Point Analysis. After several years of internal use, Albrecht introduced the methodology at a joint/share conference. From 1979 to 1984 continued statistical analysis was performed on the method and refinements were made. At that point, a non-profit organization by the name of International Function Point User Group (IFPUG) was formed which formally took onto itself the role of refining and defining the counting rules. The result is the function point methodology that we use today.

Since 1979, when Albrecht published his first paper on FP, its popularity and use has been increasing consistently and today it is being used as a de facto standar for software measurement. Following is a short list of organizations using FP for estimation

1. IEEE recommends it for use in productivity measurement and reporting.
2. Several governments including UK, Canada, and Hong Kong have been using it and it has been recommended to these governments that all public sector project use FP as a standard for the measurement of the software size.

3. Government of the Australian state Victoria has been using FP since 1997 for managing and outsourcing projects to the tune of US$ 50 Million every year.
4. In the US several large government departments including IRS have adopted FP analysis as a standard for outsourcing, measurement, and control of software projects.
5. A number of big organizations including Digital Corporation and IBM have been using FP for their internal use for the last many years.

Usage of FP includes:

- Effort Scope Estimation
- Project Planning
- Determine the impact of additional or changed requirements
- Resource Planning/Allocation
- Benchmarking and target setting
- Contract Negotiation

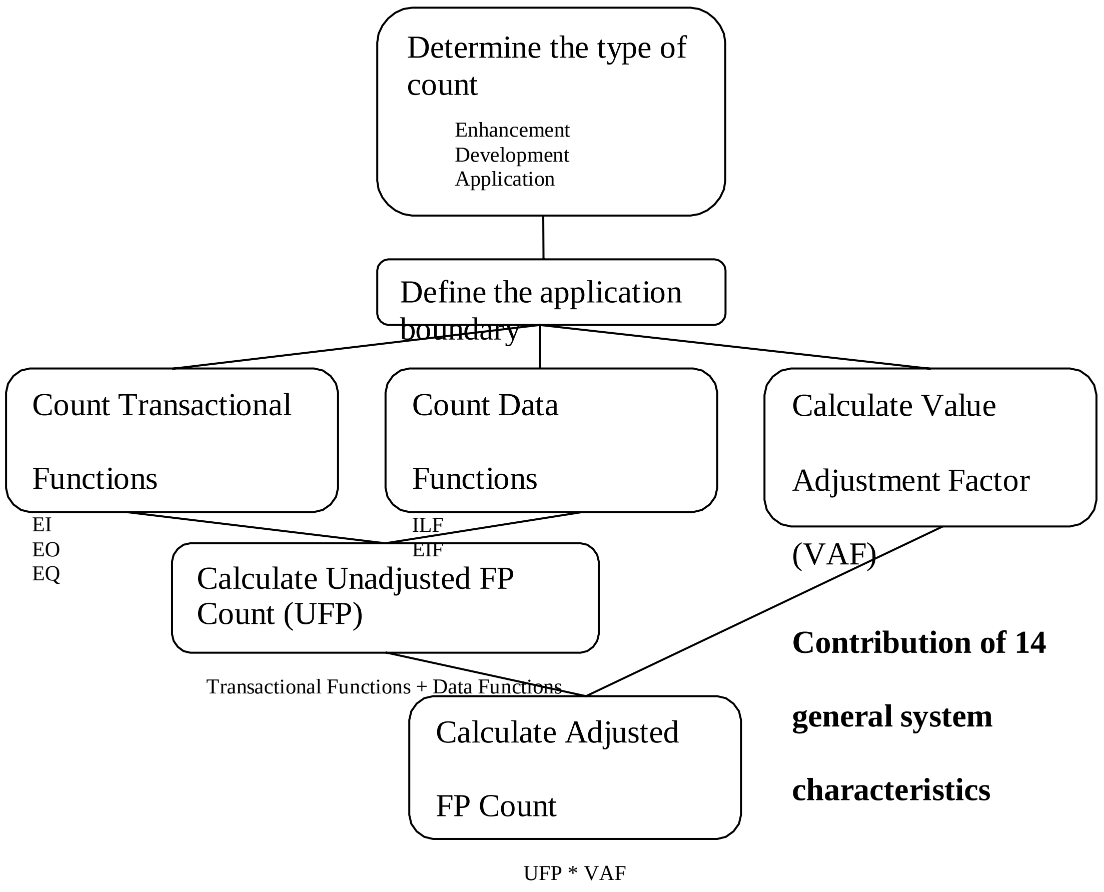
Following is a list of some of the FP based metrics used for these purposes:

- Size – Function Points
- Defects – Per Function Point
- Effor – Staff-Months
- Productivity – Function Points per Staff-Month
- Duration – Schedule (Calendar) Months
- Time Efficiency – Function Points per Month
- Cost – Per Function

Lecture No. 10

Function Point Counting Process

The following diagram depicts the function point counting process.


Determine the type of count
Enhancement
Development
Application
Define the application boundary
Count Transactional Functions
EI
EO
EQ
Count Data Functions
ILF
EIF
Calculate Value Adjustment Factor (VAF)
Calculate Unadjusted FP Count (UFP)
Transactional Functions + Data Functions
Contribution of 14 general system characteristics
Calculate Adjusted FP Count
UFP * VAF

These steps are elaborated in the following subsections. The terms and definitions are the ones used by IFPUG and have been taken directly from the IFPUG Function Point Counting Practices Manual (CPM) Release 4.1. The following can therefore be treated as an abridged version of the IFPUG CPM Release 4.1.

Determining the type of count

A Function Point count may be divided into the following types:

1. **Development Count**: A development function point count includes all functions impacted (built or customized) by the project activities.
2. **Enhancement Count**: An enhancement function point count includes all the functions being added, changed and deleted. The boundary of the application(s) impacted remains the same. The functionality of the application(s) reflects the impact of the functions being added, changed or deleted.
3. **Application Count**: An application function point count may include, depending on the purpose (e.g., provide a package as the software solution):
   a) only the functions being used by the user

b) all the functions delivered
c) The application boundary of the two counts is the same and is independent of the *scope*.

Defining the Application Boundary

The application boundary is basically the system context diagram and determines the scope of the count. It indicates the border between the software being measured and the user. It is the conceptual interface between the 'internal' application and the 'external' user world. It depends upon the user's external view of the system and is independent of the tool and technology used to accomplish the task.

The position of the application boundary is important because it impacts the result of the function point count. The application boundary assists in identifying the data entering the application that will be included in the scope of the count.

Count Data Functions

Count of the data functions is contribution of the data manipulated and used by the application towards the final function point count. The data is divided into two categories: the *Internal Logical Files* (ILF) and the *External Interface Files* (EIF). These and the related concepts are defined and explained as follows.

Internal Logical Files (ILF)

An internal logical file (ILF) is a user identifiable group of logically related data or control information maintained within the boundary of the application. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted.

External Interface Files

An external interface file (EIF) is a user identifiable group of logically related data or control information referenced by the application, but maintained within the boundary of another application. The primary intent of an EIF is to hold data referenced through one or more elementary processes within the boundary of the application counted. This means an EIF counted for an application must be in an ILF in another application.

Difference between ILFs and EIFs

The primary difference between an internal logical file and an external interface file is that an EIF **is not** maintained by the application being counted, while an ILF is.

Definitions for Embedded Terms

The following paragraphs further define ILFs and EIFs by defining embedded terms within the definitions.

Control Information

Control Information is data that influences an elementary process of the application being counted. It specifies what, when, or how data is to be processed. For example, someone in the payroll department establishes payment cycles to schedule when the employees for each location are to be paid. The payment cycle, or schedule, contains timing information that affects when the elementary process of paying employees occurs.

User Identifiable

The term *user identifiable* refers to defined requirements for processes and/or groups of data that are agreed upon, and understood by, both the user(s) and software developer(s). For example, users and software developers agree that a Human Resources Application will maintain and store Employee information in the application.

Maintained

The term *maintained* is the ability to modify data through an elementary process. Examples include, but are not limited to, add, change, delete, populate, revise, update, assign, and create.

Elementary Process

An *elementary process* is the smallest unit of activity that is meaningful to the user(s). For example, a user requires the ability to add a new employee to the application. The user definition of employee includes salary and dependent information. From the user perspective, the smallest unit of activity is to add a new employee. Adding one of the pieces of information, such as salary or dependent, is not an activity that would qualify as an elementary process. The *elementary process* must be self-contained and leave the business of the application being counted in a consistent state. For example, the user requirements to add an employee include setting up salary and dependent information. If all the employee information is not added, an employee has not yet been created. Adding some of the information alone leaves the business of adding an employee in an inconsistent state. If both the employee salary and dependent information is added, this unit of activity is completed and the business is left in a consistent state.

ILF/EIF Counting Rules

This section defines the rules that apply when counting internal logical files and external interface files.

Summary of Counting Procedures

The ILF and EIF counting procedures include the following two activities:

1) Identify the ILFs and EIFs.
2) Determine the ILF or EIF complexity and their contribution to the unadjusted function point count. ILF and EIF counting rules are used for each activity.

There are two types of rules:

- Identification rules
- Complexity and contribution rules

The following list outlines how the rules are presented:

- ILF identification rules
- EIF identification rules
- Complexity and contribution rules, which include:
  - Data element types (DETs)
  - Record element types (RETs)

ILF Identification Rules

To identify ILFs, look for groups of data or control information that satisfy the definition of an ILF. All of the following counting rules must apply for the information to be counted as an ILF.

- The group of data or control information is logical and user identifiable.
- The group of data is maintained through an elementary process within the application boundary being counted.

EIF Identification Rules

To identify EIFs, look for groups of data or control information that satisfy the definition of an EIF. All of the following counting rules must apply for the information to be counted as an EIF.

- The group of data or control information is logical and user identifiable.
- The group of data is referenced by, and external to, the application being counted.
- The group of data **is not maintained** by the application being counted.
- The group of data is maintained in an ILF of another application.

Complexity and Contribution Definitions and Rules

The number of ILFs, EIFs, and their relative functional complexity determine the contribution of the data functions to the unadjusted function point count. Assign each identified ILF and EIF a functional complexity based on the number of data element types (DETs) and record element types (RETs) associated with the ILF or EIF. This section defines DETs and RETs and includes the counting rules for each.

DET Definition

A *data element type* is a unique user recognizable, non-repeated field.

DET Rules

The following rules apply when counting DETs:

1. Count a DET for each unique user recognizable, non-repeated field maintained in or retrieved from the ILF or EIF through the execution of an elementary process. For example:
   - An account number that is stored in multiple fields is counted as one DET.

- A before or after image for a group of 10 fields maintained for audit purposes would count as one DET for the before image (all 10 fields) and as one DET for the after image (all 10 fields) for a total of 2 DETs.
- The result(s) of a calculation from an elementary process, such as calculated sales tax value for a customer order maintained on an ILF is counted as one DET on the customer order ILF.
- Accessing the price of an item which is saved to a billing file or fields such as a time stamp if required by the user(s) are counted as DETs.
- If an employee number which appears twice in an ILF or EIF as (1) the key of the employee record and (2) a foreign key in the dependent record, count the DET only once.
- Within an ILF or EIF, count one DET for the 12 Monthly Budget Amount fields. Count one additional field to identify the applicable month. For Example:

2. When two applications maintain and/or reference the same ILF/EIF, but each maintains/references separate DETs, count only the DETs being used by each application to size the ILF/EIF. For Example:
   - Application A may specifically identify and use an address as street address, city, state and zip code. Application B may see the address as one block of data without regard to individual components. Application A would count four DETs; Application B would count one DET.
   - Application X maintains and/or references an ILF that contains a SSN, Name, Street Name, Mail Stop, City, State, and Zip. Application Z maintains and/or references the Name, City, and State. Application X would count seven DETs; Application Z would count three DETs.
3. Count a DET for each piece of data required by the user to establish a relationship with another ILF or EIF.
   - In an HR application, an employee's information is maintained on an ILF. The employee’s job name is included as part of the employee's information. This DET is counted because it is required to relate an employee to a job that exists in the organization. This type of data element is referred to as a *foreign key*.
   - In an object oriented (OO) application, the user requires an association between object classes, which have been identified as separate ILFs. Location name is a DET in the Location EIF. The location name is required when processing employee information; consequently, it is also counted as a DET within the Employee ILF.

Lecture No. 1

Function Point Counting Process (cont.)

RET Definition

A *record element type* (RET) is a user recognizable subgroup of data elements within an ILF or EIF. There are two types of subgroups:

- Optional
- Mandatory

Optional subgroups are those that the user has the option of using one or none of the subgroups during an elementary process that adds or creates an instance of the data.
Mandatory subgroups are subgroups where the user must use at least one. For example, in a Human Resources Application, information for an employee is added by entering some general information. In addition to the general information, the employee is a salaried or hourly employee. The user has determined that an employee must be either salaried or hourly. Either type can have information about dependents. For this example, there are three subgroups or RETs as shown below:

- Salaried employee (mandatory); includes general information
- Hourly employee (mandatory); includes general information
- Dependent (optional)

RET Rules

One of the following rules applies when counting RETs:

- Count a RET for each optional or mandatory subgroup of the ILF or EIF.
  Or
- If there are no subgroups, count the ILF or EIF as one RET.

Hints to Help with Counting

The following hints may help you apply the ILF and EIF counting rules.
Caution: These hints *are not* rules and should not be used as rules.

1. Is the data a logical group that supports specific user requirements?
   a) An application can use an ILF or EIF in multiple processes, but the ILF or EIF is counted only once. mc
   b) A logical file cannot be counted as both an ILF and EIF for the same application. mc If the data group satisfies both rules, count as an ILF.
   c) If a group of data was not counted as an ILF or EIF itself, count its data elements as DETs for the ILF or EIF, which includes that group of data.
   d) Do not assume that one physical file, table or object class equals one logical file when viewing data logically from the user perspective.
   e) Although some storage technologies such as tables in a relational DBMS or sequential flat file or object classes relate closely to ILFs or EIFs, do not assume that this always equals a one-to-one physical-logical relationship. mc
   f) Do not assume all physical files must be counted or included as part of an ILF or EIF. mc
2. Where is data maintained? Inside or outside the application boundary?
   a) Look at the workflow.

b) In the process functional decomposition, identify where interfaces occur with the user and other applications.
c) Work through the process diagram to get hints.
d) Credit ILFs maintained by more than one application to each application at the time the application is counted. Only the DETs being used by each application being counted should be used to size the ILF/EIF.

3. Is the data in an ILF maintained through an elementary process of the application?
   a) An application can use an ILF or EIF multiple times, but you count the ILF or EIF only once.
   b) An elementary process can maintain more than one ILF. mc
   c) Work through the process diagram to get hints.
   d) Credit ILFs maintained by more than one application to each application at the time the application is counted.

Hints to Help with Identifying ILFs, EIFs, and RETs

Differentiating RETs from ILFs and EIFs is one of the most activities in FP analysis. Different concepts regarding entities play a pivotal role in this regards. Let us therefore understand what an entity is and what different types of entities are.

Entity

An entity is defined by different people as follows:

- A thing that can be distinctly identified. (Chen)
- Any distinguishable object that is to be represented in the database. (Date)
- Any distinguishable person, place, thing, event or concept about which information is kept. (Bruce)
- A data entity represents some "thing" that is to be stored for later reference. The term entity refers to the logical representation of data. (Finkelstein)
- mc An entity may also represent the relationship between two or more entities, called **associative entity** (Reingruber)
- An entity may represent a subset of information relevant to an instance of an entity, called **subtype entity**. (Reingruber)

That is, an entity is a principal data object about which information is collected that is a fundamental thing of relevance to the user, about which a collection of facts is kept.

im

An entity can be a weak entity or a strong entity. A weak entity is the one which does not have any role in the problem domain without some other entity. Weak entities are RETs and strong entities are ILFs and EIFs. I entification of weak entities is therefore important for distinguishing between RETs and logical files.

Weak Entities im

There are three types of weak entities: Associative entity types, attributive entity type, and entity subtype. These are elaborated as follows:

- **Associative Entity Type** – An entity which defines many-to-many relationship between two or more entities.
  - Student – course
  - Part – dealer

- **Attributive Entity Type** – An entity type which further describes one or more characteristics of another entity.
  - Product – Part
  - Product – Product Price Information
- **Entity Subtype** – A subdivision of entity. A subtype inherits all the attributes of its parent entity type, and may have additional, unique attributes.
  - **Employee**
    - **Permanent Employee**
    - **Contract Employee**
  - **Employee**
    - **Married Employee**
    - **Single Employee**

Logical Files

Grouping of data into logical files is the result of combined effect of two grouping methods:

- How data is accessed as a group by elementary processes? (process driven)
- The relationship between the entities and their interdependency based on business rules. (data driven)

The following guideline can be used for this purpose:

- Process Driven Approach
- Data Driven Approach

Process Driven Approach

If several entities are always **created** together and **deleted** together then this is a strong indication that they should be grouped into a single logical file.

- A customer PO is a single group of data from a user business perspective.
- It consists of a header and items information.
- From a business perspective, an order cannot be created unless it has at least one item and if the order is deleted both the order header and items are deleted. However the header and the items may have independent maintenance transactions.

Data Driven Approach

Entity Independence: an entity has significance to the business in and of itself without the presence of other entities. This is a logical file.

Entity Dependence: an entity is not meaningful, has no significance to the business in and of itself **without** the presence of other entities. This is an RET.

- Given two linked entities A and B, whether B is dependent or independent:
  - Is B significant to the business apart from the occurrence of A linked to it?
  - If we delete an occurrence "a" of A, what happens to occurrence "b" of B linked to "a"?

For example in the following scenarios, the first one is the example of entity dependence while the second one is the example of entity independence.

- – Employee – Child
- – Employee - Company Adopted Child

These concepts are summarized in the following table:

E/R Concept	E/R Term	FPA Term	IFPUG CPM 4.1 Definition
Principal data objects about which information is collected	Entity or Entity Type	ILF or EIF	File refers to a logically related group of data and not the physical implementation of those groups of data.
An entity type which contains attributes which further describe relationships between other entities	Associative entity type	Optional or mandatory subgroup	User recognizable subgroup of data elements within an ILF or EIF
An entity type that further describes one or more characteristics of another entity type	Attributive entity type	Optional or mandatory subgroup	User recognizable subgroup of data elements within an ILF or EIF
A division of an entity type, which inherits all the attributes and relationships of its parent entity type; may have additional, unique attributes and relationships	Entity subtype	Optional or mandatory subgroup	User recognizable subgroup of data elements within an ILF or EIF

Definitions: EIs, EOs and EQs

This section includes the definitions of EIs, EOs and EQs. Embedded terms within the definitions are defined, and examples are included throughout this definition section.

External Inputs

An external input (EI) is an elementary process that processes data or control information that comes from outside the application boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system.

External Outputs

An external output (EO) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external output is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information . The processing logic must contain at least one mathematical formula or calculation, or create derived data. An external output may also maintain one or more ILFs and/or alter the behavior of the system.

External Inquiry

An external inquiry (EQ) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF or EIF. The processing logic contains no mathematical formulas or calculations, and creates no derived data. No ILF is maintained during the processing, nor is the behavior of the system altered.

Summary of the Functions Performed by EIs, EOs, and EQs

The main difference between the transactional function types is their primary intent. The table below summarizes functions that may be performed by each transactional function type, and specifies the primary intent of each. Note the primary intent for an EI—this is the main difference from EOs and EQs. Some of the differences between EOs and EQs are that an EO may perform the functions of altering the behavior of the system or maintaining one or more ILFs when performing the primary intent of presenting information to the user. Other differences are identified in the section below that summarizes forms of processing logic used by each transactional function.

Function	Transactional Function Type		
	EI	EO	EQ
Alter the behavior of the system	PI	F	N/A
Maintain one or more ILFs	PI	F	N/A
Present information to a user	F	PI	PI

Legend:

PI	The primary intent of the transactional function type
F	A function of the transactional function type, but is not the primary intent and is sometimes present
N/A	The function is not allowed by the transactional function type.

Processing Logic

Processing logic is defined as requirements specifically requested by the user to complete an elementary process. Those requirements may include the following actions:

1. Validations are performed

For example, when adding a new employee to an organization, the employee process has processing logic that validates the information being added.

2. Mathematical formulas and calculations are performed.

For example, when reporting on all employees within an organization the process includes calculating the total number of salaried employees, hourly employees and all employees.

3. Equivalent values are converted

For example, an elementary process references currency conversion rates from US dollars to other currencies. The conversion is accomplished by retrieving values from tables, so calculations need not be performed.

4. Data is filtered and selected by using specified criteria to compare multiple sets of data.

For example, to generate a list of employees by assignment, an elementary process compares the job number of a job assignment to select and lists the appropriate employees with that assignment.

5. Conditions are analyzed to determine which are applicable.

For example, processing logic exercised by the elementary process when an employee is added and will depend on whether an employee is paid based on salary or hours worked.

6. One or more ILFs are updated.

For example, when adding an employee, the elementary process updates the employee ILF to maintain the employee data.

7. One or more ILFs or EIFs are referenced.

For example, when adding an employee, the currency EIF is referenced to use the correct US dollar conversion rate to determine an employee's hourly rate.

8. Data or control information is retrieved.

a) For example, to view a list of possible pay grades, pay grade information is retrieved.

9. Derived data is created by transforming existing data to create additional data.

For example, to determine (derive) a patient's registration number (e.g., SMIJO01), the following data is concatenated:

a) the first three letters of the patient's last name (e.g., SMI for Smith)
b) the first two letter of the patient's first name (e.g., JO for John)
c) a unique two-digit sequence number (starting with 01)

10. Behavior of the system is altered.

For example, the behavior of the elementary process of paying employees is altered when a change is made to pay them every other Friday versus on the 15th and the last day of the month.

11. Prepare and present information outside the boundary.

For example, a list of employees displayed for the user.

12. Capability exists to accept data or control information that enters the application boundary.

For example, a user enters several pieces of information to add a customer order to the system.

13. Data is resorted or rearranged.

For example, a user requests the list of employees in alphabetical order.

Note: Resorting or rearranging a set of data does not impact the identification of the type or uniqueness of a transactional function.

Summary of Processing Logic Used by EIs, EOs and EQs

The following table summarizes which forms of g logic may be performed by EIs, Eos, and EQs. Foe each transactional function type, certain types of processing logic must be performed to accomplish the primary intent of that type.

Form of Processing Logic	Transactional Functional Type		
	EI	**EO**	**EQ**
1. Validations are performed	c can	c can	c can
2. Mathematical Formula and calculations are performed	c	m*	n no
3. Equivalent Values are converted	c	c	c
4. Data is filtered and selected by using specified criteria to compare multiple sets of data.	c	c	c
5. Conditions are analyzed to determine which are applicable	c	c	c
6. At least one ILF is updated	m*	m*	n
7. At least one ILF or EIF is referenced	c	c	m
8. Data or control information is retrieved	c	c	m
9. Derived data is created	c	m*	n
10. Behavior of system is altered	m*	m*	n
11. Prepare and present information outside the boundary	c	m	m
12. Capability to accept data or control information that enters the application boundary	m*	c	c
13. Resorting or rearranging a set of data	c	c	c

Legend

m	it is **mandatory** that the function type perform the form of processing logic.
m*	it is **mandatory** that the function type perform at least on of these (m*) forms of processing logic
c	the function type **can** perform the form of processing logic, but it is not mandatory.
n	function type **cannot** perform the form of processing logic

EI/EO/EQ Counting Rules

This section defines the rules that apply when counting EIs, EOs and EQs.

Elementary Process Identification Rules

To identify elementary processes, look for user activities occurring in the application.
All of the following counting rules must apply for the process to be identified as an elementary process.

- The process is the smallest unit of activity that is meaningful to the user.
- The process is self-contained and leaves the business of the application in a consistent state.

Transactional Functions Counting Rules

To classify each elementary process, determine which of the primary intent descriptions apply, and use the associated rules to identify a specific transactional function type.

Primary Intent Description for EIs

The primary intent of an elementary process is to maintain an ILF or alter the behavior of the system.

External Input Counting Rules

For each elementary process that has a primary intent to maintain one or more ILFs or to alter the behavior of the system, apply the following rules to determine if the function should be classified as an external input. All of the rules must apply for the elementary process to be counted as a unique occurrence of an external input.

- The data or control information is received from outside the application boundary.
- At least one ILF is maintained if the data entering the boundary is not control information that alters the behavior of the system.
- For the identified process, one of the following three statements must apply:
    - Processing logic is unique from the processing logic performed by other external inputs for the application.
    - The set of data elements identified is different from the sets identified for other external inputs for the application.
    - The ILFs or EIFs referenced are different from the files referenced by other external inputs in the application.

Primary Intent Description for EOs and Eqs

The primary intent of the elementary process is to present information to a user.

Shared EO and EQ Counting Rules

For each elementary process that has a primary intent to present information to a user, apply the following rules to determine if the process may be classified as an external output or external inquiry. All of the rules must apply for the elementary process to be counted as a unique occurrence of an external output or external inquiry.

- The function sends data or control information external to the application boundary.
- For the identified process, one of the following three statements must apply:
  - Processing logic is unique from the processing logic performed by other external outputs or external inquiries for the application.
  - The set of data elements identified is different from the sets identified for other external outputs and external inquiries in the application.
  - The ILFs or EIFs referenced are different from the files referenced by other external outputs and external inquiries in the application.

Additional External Output Counting Rules

In addition to adhering to all shared EO and EQ rules, one of the following rules must apply for the elementary process to be counted as a unique external output.

- The processing logic of the elementary process contains at least one mathematical formula or calculation.
- The processing logic of the elementary process creates derived data.
- The processing logic of the elementary process maintains at least one ILF.
- The processing logic of the elementary process alters the behavior of the system.

Additional External Inquiry Counting Rules

In addition to adhering to all shared EO and EQ rules, all of the following rules must apply for the elementary process to be counted as a unique external inquiry.

- The processing logic of the elementary process retrieves data or control information from an ILF or EIF.
- The processing logic of the elementary process does not contain a mathematical formula or calculation.
- The processing logic of the elementary process does not create derived data.
- The processing logic of the elementary process does not maintain an ILF.
- The processing logic of the elementary process does not alter the behavior of the system.

Complexity and Contribution Definitions and Rules

The number of EIs, EOs, and EQs and their relative functional complexities determine the contribution of the transactional functions to the unadjusted function point count.
Assign each identified EI, EO and EQ a functional complexity based on the number of file types referenced (FTRs) and data element types (DETs).

FTR Definition

A *file type referenced* is
 An internal logical file read or maintained by a transactional function or
 An external interface file read by a transactional function

DET Definition

A *data element type* is a unique user recognizable, non-repeated field.

EI Complexity and Contribution Rules

This section defines FTR and DET rules used to determine the complexity and contribution of external inputs.

FTR Rules for an EI

The following rules apply when counting FTRs:

- Count an FTR for each ILF maintained.
- Count an FTR for each ILF or EIF read during the processing of the external input.
- Count only one FTR for each ILF that is both maintained and read.

DET Rules for an EI

The following rules apply when counting DETs:

- Count one DET for each user recognizable, non-repeated field that enters or exits the application boundary and is required to complete the external input. For example, job name and pay grade are two fields that the user provides when adding a job.
- Do not count fields that are retrieved or derived by the system and stored on an ILF during the elementary process if the fields did not cross the application boundary.

For example, when the customer order is added to the system, the unit price is automatically retrieved for each ordered item and stored on the billing record. The unit price would not be counted as a DET for the EI because it did not cross the boundary when the user adds the customer order.

For example, in order to maintain the US hourly rate for hourly employees working in other countries with other currencies, the local hourly rate is provided by the user. During the processing of all the pieces of data provided to add an employee, a conversion rate is retrieved from the currency system to calculate the US hourly rate. The calculated US hourly rate is maintained on the employee ILF as a result of adding the employee. The US hourly rate would not be counted as a DET for the EI because it does not enter the boundary, but is internally calculated (i.e., it is derived data).

- Count one DET for the capability to send a system response message outside the application boundary to indicate an error occurred during processing, confirm that processing is complete or verify that processing should continue.

For example, if a user tries to add an existing employee to a Human Resources application, the system generates one of several error messages and the incorrect field is highlighted. Count one DET that includes all the system responses which indicate the error conditions, confirm that processing is complete or verify that processing should continue.

- Count one DET for the ability to specify an action to be taken even if there are multiple methods for invoking the same logical process.

For example, if the user can initiate the adding of an employee clicking on the OK button or by pressing a PF key, count one DET for the ability to initiate the process.

EO/EQ Complexity and Contribution Rules

This section defines FTR and DET rules used to determine the complexity and contribution of external outputs and external inquiries.

Shared FTR Rules for EOs and EQs

The following rule applies when counting FTRs for both EOs and EQs:

- Count one FTR for each ILF or EIF read during the processing of the elementary process.

Additional FTR Rules for an EO

The following additional rules apply when counting FTRs for EOs:

- Count one FTR for each ILF maintained during the processing of the elementary process.
- Count only one FTR for each ILF that is both maintained and read during the elementary process.

Shared DET Rules for EOs and EQs

The following rules apply when counting DETs for both EOs and EQs.

- Count one DET for each user recognizable, non-repeated field that enters the application boundary and is required to specify when, what and/or how the data is to be retrieved or generated by the elementary process. For example (EO/EQ), to generate a list of employees, employee name is a field the user provides when indicating which employees to list.
- Count one DET for each user recognizable, non-repeated field that exits the boundary.

For example (EO/EQ), a text message may be a single word, sentence, or phrase—a line or paragraph included on a report to indicate an explanatory comment counts as a single DET.

For example (EO/EQ), an account number or date physically stored in multiple fields is counted as one DET when it is required as a single piece of information.

For example (EO/EQ), a pie chart might have a category label and a numerical equivalent in a graphical output. Count two DETs —one for designating the category and one for the numerical value.

- If a DET both enters and exits the boundary, count it only once for the elementary process.
- Count one DET for the capability to send a system response message outside the application boundary to indicate an error occurred during processing, confirm that processing is complete or verify that processing should continue.

For example (EO/EQ), if a user tries to request a listing, but does not have access to the information, count one DET for the system response.

- Count one DET for the ability to specify an action to be taken even if there are multiple methods for invoking the same logical process.

For example (EO/EQ), if the user can initiate the generation of a report by clicking on the OK button or by pressing a PF key, count one DET for the ability to initiate the report.

- Do not count fields that are retrieved or derived by the system and stored on an ILF during the elementary process if the fields did not cross the application boundary.

  For example (EO), when a paycheck is printed, a status field on the employee ILF is updated to indicate that the check has been printed. Do not count the status field as a DET since it did not cross the boundary.

- Do not count literals as DETs.

For example (EO/EQ), literals include report titles, screen or panel identification, column headings, and field titles.

- Do not count paging variables or system-generated stamps.

  For example (EO/EQ), system-generated variables and stamps include
  - Page numbers
  - Positioning information such as "Rows 37 to 54 of 211"
  - Paging commands such as previous, next, and paging arrows on a GUI application
  - Date and time fields if they are displayed.

Hints to Help with Counting EIs, EOs and EQs

The following hints may help you apply the EI, EO and EQ counting rules.
Caution: The hints *are not* rules and should not be used as rules.

- Is data received from outside the application boundary?
  - Look at the work flow.
  - Identify where the user and other application interfaces occur in the process functional decomposition.
- Is the process the smallest unit of activity from the user perspective?
  - Look at the different paper or on-line forms used.
  - Review the ILFs to identify how the user groups the information.
  - Identify where the user and other application interfaces occur in the process functional decomposition.
  - Look at what happened in the manual system.
  - Note that one physical input or transaction file or screen can, when viewed logically, correspond to a number of EIs, EOs or EQs.
  - Note that two or more physical input or transaction files or screens can correspond to one EI, EO or EQ if the processing logic is identical.
- Is the process self-contained and does it leave the business in a consistent state?
  - Review other external inputs, external outputs and external inquiries to understand how the user works with the information.
  - Work through the process diagram to get hints.
  - Look at what happened in the manual system.
  - Check for consistency with other decisions.
- Is the processing logic unique from other EIs, EOs and EQs?
  - Identify batch inputs or outputs based on the processing logic required.
  - Remember that sorting or rearranging a set of data does not make processing logic unique.
- Are the data elements different from those for other EIs, EOs or EQs?
  - If the data elements appear to be a subset of the data elements of another EI, EO, or EQ, be sure two elementary processes are required by the user – one for the main data elements and one for the subsets.
- Identify the primary intent of the elementary process before classifying it as an EI, EO, or EQ.
- Identification of the elementary process(es) is based on a joint understanding or interpretation of the requirements between the user and the developers.
- Each element in a functional decomposition may not map to a unique elementary process.
- The identification of the elementary processes requires interpretation of the user requirements. Count only one FTR for each ILF/EIF referenced even if the ILF/EIF has multiple RETs.

Additional Hints to Help Counting EOs and EQs

- Is the process the smallest unit of activity from the user perspective?
  - An EO or EQ can be triggered by a process inside the application boundary. For example, the user requires that a report of all changed employee pay rates be sent to the budgeting area every 8 hours based on an internal clock. Situation A. The report contains employee name, SSN, and hourly pay rate which are all retrieved from the employee file. This is the smallest unit of activity from the user's perspective, contains no mathematical formulas or calculations, and no ILF is maintained in the process. This is one EQ. Situation B. The report contains employee name, SSN, and hourly pay rate which are all retrieved from the employee file. The report also

includes the percentage pay change for the employee which is calculated from the data on the employee file. This is the smallest unit of activity from the user's perspective, and no ILF is maintained in the process. However, since the process contains a mathematical formula, this is one EO.

- Derived data for an EO does not have to be displayed on the output. For example, each month, a report is generated listing all employees due for appraisal in the next 30 days. The records are selected by calculating next appraisal date based on the employee's last appraisal date, which is a field on the employee file, and the current date plus 30 days. This would be counted as one EO, and not as an EQ.

General System Characteristics

The Unadjusted Function Point count is multiplied by an adjustment factor called the Value Adjustment Factor (VAF). This factor considers the system's technical and operational characteristics and is calculated by answering 14 questions. The factors are:

1. DATA COMMUNICATIONS

The *data* and *control* information used in the application are sent or received over communication facilities. Terminals connected locally to the control unit are considered to use communication facilities. Protocol is a set of conventions which permit the transfer or exchange of information between two systems or devices. All data communication links require some type of protocol.

Score As:

0 Application is pure batch processing or a standalone PC.
1 Application is batch but has remote data entry *or* remote printing.
2 Application is batch but has remote data entry *and* remote printing.
3 Application includes online data collection or TP (teleprocessing) front end to a batch process or query system.
4 Application is more than a front-end, but supports only one type of TP communications protocol.
5 Application is more than a front-end, and supports more than one type of TP communications protocol.

2. DISTRIBUTED DATA PROCESSING

Distributed data or processing functions are a characteristic of the application within the application boundary.

Score As:

0 Application does not aid the transfer of data or processing function between components of the system.
1 Application prepares data for end user processing on another component of the system such as PC spreadsheets and PC DBMS.
2 Data is prepared for transfer, then is transferred and processed on another component of the system (not for end-user processing).
3 Distributed processing and data transfer are online and in one direction only.
4 Distributed processing and data transfer are online and in both directions.
5 Processing functions are dynamically performed on the most appropriate component of the system.

3. PERFORMANCE

Application performance objectives, stated or approved by the user, *in either* response or throughput, influence (or will influence) the design, development, installation, and support of the application.

Score As:

0 No special performance requirements were stated by the user.
1 Performance and design requirements were stated and reviewed but no special actions were required.
2 Response time or throughput is critical during peak hours. No special design for CPU utilization was required. Processing deadline is for the next business day.
3 Response time or throughput is critical during all business hours. No special design for CPU utilization was required. Processing deadline requirements with interfacing systems are constraining.
4 In addition, stated user performance requirements are stringent enough to require performance analysis tasks in the design phase.
5 In addition, performance analysis tools were used in the design, development, and/or implementation phases to meet the stated user performance requirements.

4. HEAVILY USED CONFIGURATION

A heavily used operational configuration, requiring special design considerations, is a characteristic of the application. For example, the user wants to run the application on existing or committed equipment that will be heavily used.

Score As:

0 No explicit or implicit operational restrictions are included.
1 Operational restrictions do exist, but are less restrictive than a typical application. No special effort is needed to meet the restrictions.
2 Some security or timing considerations are included.
3 Specific processor requirements for a specific piece of the application are included.
4 Stated operation restrictions require special constraints on the application in the central processor or a dedicated processor.
5 In addition, there are special constraints on the application in the distributed components of the system.

5. TRANSACTION RATE

The transaction rate is high and it influenced the design, development, installation, and support of the application.

Score As:

1 Peak transaction period (e.g., monthly, quarterly, seasonally, annually) is anticipated.
2 Weekly peak transaction period is anticipated.
3 Daily peak transaction period is anticipated.
4 High transaction rate(s) stated by the user in the application requirements or service level agreements are high enough to require performance analysis tasks in the design phase.

5 High transaction rate(s) stated by the user in the application requirements or service level agreements are high enough to require performance analysis tasks and, in addition, require the use of performance analysis tools in the design, development, and/or installation phases. Online data entry and control functions are provided in the application.

6. ONLINE DATA ENTRY

On-line data entry and control information functions are provided in the application.

Score As:

0 All transactions are processed in batch mode.
1 1% to 7% of transactions are interactive data entry.
2 8% to 15% of transactions are interactive data entry.
3 16% to 23% of transactions are interactive data entry.
4 24% to 30% of transactions are interactive data entry.
5 More than 30% of transactions are interactive data entry.

7. END-USER EFFICIENCY

The online functions provided emphasize a design for end-user efficiency. The design includes:

- Navigational aids (for example, function keys, jumps, dynamically generated menus)
- Menus
- Online help and documents
- Automated cursor movement
- Scrolling
- Remote printing (via online transactions)
- Pre-assigned function keys
- Batch jobs submitted from online transactions
- Cursor selection of screen data
- Heavy use of reverse video, highlighting, colors underlining, and other indicators
- Hard copy user documentation of online transactions
- Mouse interface
- Pop-up windows.
- As few screens as possible to accomplish a business function
- Bilingual support (supports two languages; count as four items)
- Multilingual support (supports more than two languages; count as six items)

Score As:

0 None of the above.
1 One to three of the above.
2 Four to five of the above.
3 Six or more of the above, but there are no specific user requirements related to efficiency.

4 Six or more of the above, and stated requirements for end user efficiency are strong enough to require design tasks for human factors to be included (for example, minimize key strokes, maximize defaults, use of templates).
5 Six or more of the above, and stated requirements for end user efficiency are strong enough to require use of special tools and processes to demonstrate that the objectives have been achieved.

8. ONLINE UPDATE

The application provides online update for the internal logical files.

Score As:

0 None.
1 Online update of one to three control files is included. Volume of updating is low and recovery is easy.
2 Online update of four or more control files is included. Volume of updating is low and recovery easy.
3 Online update of major internal logical files is included.
4 In addition, protection against data lost is essential and has been specially designed and programmed in the system.
5 In addition, high volumes bring cost considerations into the recovery process. Highly automated recovery procedures with minimum operator intervention are included.

9. COMPLEX PROCESSING

Complex processing is a characteristic of the application. The following components are present:

- Sensitive control (for example, special audit processing) and/or application specific security processing
- Extensive logical processing
- Extensive mathematical processing
- Much exception processing resulting in incomplete transactions that must be processed again, for example, incomplete ATM transactions caused by TP interruption, missing data values, or failed validations
- Complex processing to handle multiple input/output possibilities, for example, multimedia, or device independence

Score As:

0 None of the above.
1 Any one of the above.
2 Any two of the above.
3 Any three of the above.
4 Any four of the above.
5 All five of the above.

10. REUSABILITY

The application and the code in the application have been specifically designed, developed, and supported to be usable in *other* applications.

Score As:

0 No reusable code.
1 Reusable code is used within the application.
2 Less than 10% of the application considered more than one user's needs.
3 Ten percent (10%) or more of the application considered more than one user's needs.
4 The application was specifically packaged and/or documented to ease re-use, and the application is customized by the user at source code level.
5 The application was specifically packaged and/or documented to ease re-use, and the application is customized for use by means of user parameter maintenance.

11. INSTALLATION EASE

Conversion and installation ease are characteristics of the application. A conversion and installation plan and/or conversion tools were provided and tested during the system test phase.

Score As:

0 No special considerations were stated by the user, and no special setup is required for installation.
1 No special considerations were stated by the user *but* special setup is required for installation.
2 Conversion and installation requirements were stated by the user, and conversion and installation guides were provided and tested. The impact of conversion on the project is not considered to be important.
3 Conversion and installation requirements were stated by the user, and conversion and installation guides were provided and tested. The impact of conversion on the project is considered to be important.
4 In addition to 2 above, automated conversion and installation tools were provided and tested.
5 In addition to 3 above, automated conversion and installation tools were provided and tested.

12. OPERATIONAL EASE

Operational ease is characteristic of the application. Effective start-up, back-up, and recovery procedures were provided and tested during the system test phase. The application minimizes the need for manual activities, such as tape mounts, paper handling, and direct on-location manual intervention.

Score As:

0 No special operational considerations other than the normal back-up procedures were stated by the user.

1 - 4 One, some, or all of the following items apply to the application. Select all that apply. Each item has a point value of one, except as noted otherwise.

- Effective start-up, back-up, and recovery processes were provided, but operator intervention is required.
- Effective start-up, back-up, and recovery processes were provided, but no operator intervention is required (count as two items).
- The application minimizes the need for tape mounts.
- The application minimizes the need for paper handling.

5 The application is designed for unattended operation. Unattended operation means *no operator intervention* is required to operate the system other than to start up or shut down the application. Automatic error recovery is a feature of the application.

13. MULTIPLE SITES

The application has been specifically designed, developed, and supported to be installed at multiple sites for multiple organizations.

Score As:

0 User requirements do not require considering the needs of more than one user/installation site.
1 Needs of multiple sites were considered in the design, and the application is designed to operate only under identical hardware and software environments.
2 Needs of multiple sites were considered in the design, and the application is designed to operate only *under similar* hardware and/or software environments.
3 Needs of multiple sites were considered in the design, and the application is designed to operate *under different* hardware and/or software environments.
4 Documentation and support plan are provided and tested to support the application at multiple sites and the application is as described by 1 or 2.
5 Documentation and support plan are provided and tested to support the application at multiple sites and the application is as described by 3.

14. FACILITATE CHANGE

The application has been specifically designed, developed, and supported to facilitate change.
The following characteristics can apply for the application:

- Flexible query and report facility is provided that can handle simple requests; for example, *and/or* logic applied to only one internal logical file (count as one item).
- Flexible query and report facility is provided that can handle requests of average complexity, for example, *and/or* logic applied to more than one internal logical file (count as two items).
- Flexible query and report facility is provided that can handle complex requests, for example, *and/or* logic combinations on one or more internal logical files (count as three items).
- Business control data is kept in tables that are maintained by the user with online interactive processes, but changes take effect only on the next business day.
- Business control data is kept in tables that are maintained by the user with online interactive processes, and the changes take effect immediately (count as two items).

Score As:

0	None of the above.
1	Any one of the above.
2	Any two of the above.
3	Any three of the above.
4	Any four of the above.
5	All five of the above.

Adjusted FP Count

Each of these factors is scored based between 0-5 on their influence on the system being counted. The resulting score will increase or decrease the Unadjusted Function Point count by 35%. This calculation provides us with the Adjusted Function Point count.

Degree of Influence (DI) = sum of scores of 14 general system characteristics

Value Adjustment Factor (VAF) = 0.65 + DI / 100
The final Function Point Count is obtained by multiplying the VAF times the Unadjusted Function Point (UFP).

FP = UFP * VAF

Lecture No. 12

Software Process and Project Metrics

Everyone asks this question: how do I identify the problem? The answer is measure your process. Measurement helps in identification of the problem as well as in determining the effectiveness of the remedy.

Measurement is fundamental for providing mechanisms for objective evaluation of any process or activity. According to Lord Kelvin:

> *When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of a science.*

In order to understand what your problems are, you need to measure. Only then a remedy can be applied. Take the example of a doctor. He measures and monitors different types of readings from a patient (temperature, heart beat, blood pressure, blood chemistry, etc) before proposing a medicine. After giving the medicine, the doctor monitors the effect of the medicine through the follow-up visits and makes the necessary adjustments. This process of measurement, correction and feedback is inherent in all kinds of systems. Software is no exception!

The idea is to measure your product and process to improve it continuously. Now the question is: how can we measure the quality of a software process and system?

Software project management primarily deals with metrics related to productivity and quality. For planning and estimation purposes, we look at the historic data – productivity of our team and the quality of their deliverables in the past projects. This data from the previous efforts is used to determine and estimate the effort required in our current project to deliver a product with a predictable quality. This data is also used to analyze the system bottlenecks and helps us in improving the productivity of our team and the quality of our product.

Measures, Metrics and Indicators

Before we can talk about the measurement process, we first need to understand the terms *measure, metrics,* and *indicators*. The terms *measure, measurement*, and *metrics* are often used interchangeable but there are significant differences among them. Within the software engineering domain, a measure provides a quantitative value of some attribute of a process or a product. For example, size is one measure of a software product. Measurement is the process or mechanism through which the measure is taken. For example, FP analysis is a mechanism to measure the size of software. Measurement involves taking one or more data points related to some aspect of the product or process. Software metric relates individual software measures to provide a normalized view. For example, defects per function point are one metric which relates two individual measures, that is, defects and size, into one metric.

Metrics give you a better insight into the state of the process or product. These insights are not the problems but just the *indicators* of problems. A software engineers collects measures and develops metrics and indicators.

Tom Gilb says, “Anything that you need to quantify can be measured in some way that is superior to not measuring at all!” This quote has two messages:

1. Anything can be measured
2. It is always better to measure than not doing it even if you do not have a good measuring device; it always gives you some information that you can use.

Metrics for software quality

The most important objective of any engineering activity is to produce high quality product with limited resources and time. The quality of the product cannot be determined if it is be measured.

The quality of the end result depends upon the quality of the intermediate work products. If the requirements, design, code, and testing functions are of high quality, then the chances are that the end product will also be of good quality. So, a good software engineer would adopt mechanisms to measure the quality of the analysis and design models, the source code, and the test cases.

mcq

At the project level, the primary focus is to measure errors and defects and derive relevant metrics such as requirement or design errors per function point, errors uncovered per review hour, errors per thousand lines of code. These metrics provide an insight into the effectiveness of the quality assurance activities at the team as well as individual level.

Lecture No. 13

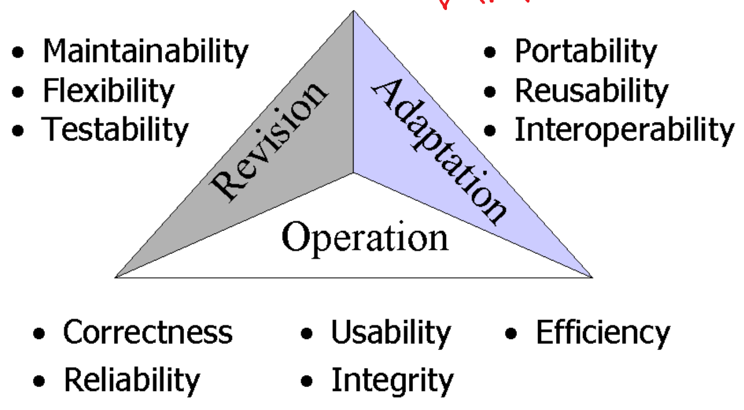
Software Quality Factors

mcq

In 1978, McCall identified factors that could be used to develop metrics for the software quality. These factors try to assess the inner quality of software from factors that can be observed from outside. The basic idea is that the quality of the software can be inferred if we measure certain attributes once the product is put to actual use. Once completed and implemented, it goes through three phases: operation (when it is used), during revisions (when it goes through changes), and during transitions (when it is ported to different environments and platforms). three phases short question

During each one of these phases, different types of data can be collected to measure the quality of the product. McCall's model is depicted and explained as follows.

✓ imp



qui **Factors related with operation** most important

- Correctness
  - The extent to which a program satisfies its specifications and fulfills the customer's mission objectives

- Reliability
  - The extent to which a program can be expected to perform its intended function with required precision.

- Efficiency
  - The amount of computing resources required by a program to perform its function

- Integrity
  - Extent to which access to software or data by unauthorized persons can be controlled.

- Usability
  - Effort required to learn, operate, prepare input, and interpret output of a program

Factors related with revision

- Maintainability
  - Effort required to locate and fix an error in a program
- Flexibility
  - Effort required to modify an operational program
- Testability
  - Effort required to test a program to ensure that it performs its intended function

Factors related with adaptation

- Portability
  - Effort required transferring the program from one hardware and/or software system environment to another.
- Reusability
  - Extent to which a program can be reused in other applications
- Interoperability
  - Effort required to couple one system to another

It is interesting to note that the field of computing and its theoretical have gone through phenomenal changes but McCall's quality factors are still as relevant as they were almost 25 years ago. mcq

Measuring Quality

mcq Gilb extends McCall's idea and proposes that the quality can be measured if we measure the correctness, maintainability, integrity, and usability of the product.

Correctness is defined as the degree to which software performs its function. It can be measured in defects/KLOC or defects/FP where defects are defined as verified lack of conformance to requirements. These are the problems reported by the user after release. These are counted over a standard period of time which is typically during the first year of operation.

Maintainability is defined as the ease with which a program can be corrected if an error is encountered, adapted if environment changes, enhanced if the customer requires an enhancement in functionality. It is an indirect measure of the quality.
A simple time oriented metric to gauge the maintainability is known as MMTC – mean time to change. It is defined as the time it takes to analyze the change request, design an appropriate modification, implement the change, test it, and implement it.

A cost oriented metric used to assess maintainability is called Spoilage. It is defined as the cost to correct defects encountered after the software has been released to the users. Spoilage cost is plotted against the overall project cost as a function of time to determine whether the overall maintainability of software produced by the organization is improving.

Integrity is an extremely important measure especially in today's context when the system is exposed to all sorts to attacks because of the internet phenomenon. It is defined as software's ability to withstand attack (both accidental and intentional) to its security. It includes integrity of program, data, and documents. To measure integrity, two additional attributes are needed. These are: threat and security.

Threat is the probability (derived or measured from empirical evidence) that an attack of a specific type will occur within a given time and security is the probability that an attack of a specific type will be repelled So the integrity of a system is defined as the sum of all the probability that the threat of a specific type will not take place and the probability that if that threat does take place, it will not be repelled.

Integrity = ∑ [(1-threat) x (1-security)] mc

Finally, usability is a measure of user friendliness – the ease with which a system can be used. It can be measured in terms of 4 characteristics:

- Physical or intellectual skill r quired learn the system
- The time required to become moderately efficient in the use of system
- The net increase in productivity
- A subjective assessment

It is important to note that except for the usability, the other three factors are essentially the same as proposed by McCall.

Defect Removal Efficiency

Defect removal efficiency is the measure of how many defects are removed by the quality assurance processes before the product is shipped for operation. It is therefore a measure that determines the effectiveness of the QA processes during development. It is useful at both the project and process level.
Defect removal efficiency is calculated as the number of defect removed before shipment as a percentage of total defects

DRE = E/(E+D) mcqs + Short numerical

Where

- E – errors found before delivery
- D – errors found after delivery (typically within the first year of operation)

Regarding the effectiveness of various QA activities, Capers Jones published some data in 1997 which is summarized in the following table.

Design Inspection					●		●	●	●
Code Inspection				●		●		●	●
Quality Assurance			●			●	●		●
Testing		●	●	●	●	●	●	●	●
Worst	30%	37%	50%	55%	65%	75%	77%	85%	95%
Median	40%	53%	65%	70%	80%	87%	90%	97%	99%
Best	50%	60%	75%	80%	87%	93%	95%	99%	99.9%

In this research, they tried to measure the effectiveness of 4 different activities namely design inspection, code inspection, quality assurance function, and testing. It is important to note that testing alone only yields a DRE of 40% on the average. However, when it is combined with design and code inspection, the DRE reaches 97%. That means, code and design inspection are extremely important activates that are unfortunately not given their due importance.

Lecture No. 14

Metrics for specification quality

As mentioned earlier, the quality of the software specification is of extreme importance as the entire building of software is built on this foundation. A requirement specification document is measured in terms of lack of ambiguity, completeness, consistency, correctness; understand ability, verifiability, achievability, concision, traceability, modifiability, precision, and reusability.

Metrics to assess the quality of the analysis models and the corresponding software specification were proposed by Davis in 1993 for these seemingly qualitative characteristics.

For example, the numbers of requirements are calculated as:

$$n_r = n_f + n_{nf}$$

mcqs

where
n_r – total number of requirements
n_f – functional requirements
n_{nf} – non-functional requirements

Now lack of ambiguity in the requirements is calculated as:

$$Q_1 = n_{ui}/n_r$$

Where
n_{ui} – number of requirements for which all reviewers had identical interpretation (i.e. unambiguous requirements)

Similarly, completeness is measures as follows:

$$Q_2 = n_u/ [n_i \text{ x } n_s]$$

n_u – unique functional requirements
n_i – number of inputs (stimuli)
n_s – number of states specified

On the similar lines, the quality of design is also measured quantitatively.
The quality of the architectural design can be measured by measuring its complexity as shown below:

- Structural complexity $S = (f_{out})^2$
- Data complexity $D = v/(f_{out} + 1)$
  - 'v' is the number of input and output variables
- System complexity $C = \sum (S_i + D_i)$

fanout means that on how many system it is depend.more means more couple with others

Baseline

In order to use the data for estimation and drawing conclusions, it must be base-lined. In the baseline, data from past projects is collected, cleaned, and put in a database. Such metrics baseline is used to reap benefits at the process, project, and product level.

As mentioned above, the metrics baseline consists of data collected from past project over a period of time. To be effective, the data must be reasonably accurate, it should be collected over many projects, measures must be consistent – same technique or yardstick for data collection should have been used, applications should be similar to work that is to be estimated, and feedback to improve baseline's quality.

Metrics for small organizations

The metric program can be quite complex and extensive. Small organization would find it difficult to implement a full-fledged metrics program as it may require quite a number of resources. However, it is important to appreciate that a metrics program of even a smaller scale would also be of immense value and therefore all organizations of all sizes should have should have one. It can be a very simple and can be implemented with the help of simple and inexpensive tools and methods.

Small organizations – around 20 or so people – must measure in a cost effective manner. In order for it to be effective, it should be simple and value-oriented and should focus on result rather than measurement. It is important to establish the objectives of measurement.

This is illustrated by the following example.

Let us assume that we wanted to reduce the time to evaluate and implement change requests in our organization. In order to achieve this objective, we needed to measure the following:

- Time (in hours or days) elapsed from the time a request is made until evaluation is complete - t_{queue}
- Size (fp) of the change request
- Effort (in person months) to perform the evaluation- W_{eval}
- Time elapsed from completion of evaluation to assignment of change order – t_{eval}
- Effort required to make the change – W_{change}
- Time required to make the change – t_{change}
- Errors uncovered during work to make change – E_{change}
- Defects uncovered after change is released – D_{change}

This data was then collected and stored in a simple database as shown below.

Project	Size (FP)	Effort (Pm)	Cost Rs. (000)	Pages of documentation	Pre-shipment errors	Post-shipment defects	People
abc	120	24	168000	365	134	29	3
def	270	62	440000	1224	321	86	5
ghi	200	43	314000	1050	256	64	6

This data is then normalized on per function point basis as follows:

Project	Size (FP)	Effort (Pm)	Cost Rs. (000)	Pages of documentation	Pre-shipment errors	Post-shipment defects	People
abc	120	0.2	1400	3.04	1.12	0.24	3
def	270	0.23	1629	4.53	1.19	0.32	5
ghi	200	0.22	1570	5.25	1.28	0.32	6

We are now ready to use this data to analyze the results of process changes and their impact on the time to implement change requests.

In order to do that, we ne d to employ some statistical techniques and plot the result graphically. This is known as statistical control techniques. mcq

Lecture No. 15

Statistical Control Techniques – control charts

Same process metrics vary from project to project. We have to determine whether the trend is statistically valid or not. We also need to determine what changes are meaningf l.
mc A graphical technique known as control charts is used to determine this.

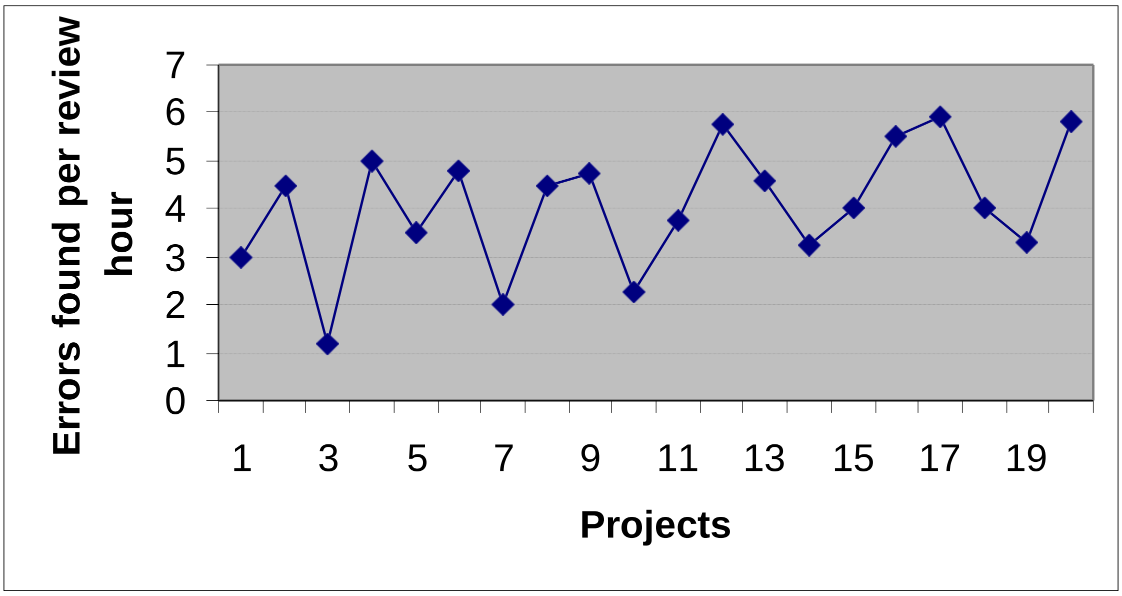
This technique was initially developed for manufacturing processes in the 1920's by Walter Shewar and is still very applicable even in disciples like software engineering.
mc Control charts are of two typ s: moving range control chart and individual control chart.
This technique enables individuals interested in software process improvement to determine whether the dispersion (variability) and "location" (moving average) of process metrics are stable (i.e. the process exhibits only natural or controlled changes) or unstable (i.e. the process exhibits out-of-control changes and metrics cannot be used to predict performance).

Let us now demonstrate the use of these control charts with the help of an example.
Let us assume that the data shown in the following table regarding the average change implementation time was collected over the last 15 months for 20 small projects in the same general software domain 20 projects. To improve the effectiveness of reviews, the software organization provided training and mentoring to all project team members beginning with project 11.

Project	Time /change implementation
1	3
2	4.5
3	1.2
4	5
5	3.5
6	4.8
7	2
8	4.5
9	4.75
10	2.25
11	3.75
12	5.75
13	4.6
14	3.25
15	4
16	5.5
17	5.9
18	4
19	3.3
20	5.8

In order to determine whether our change in process had any impact, we use control charts.

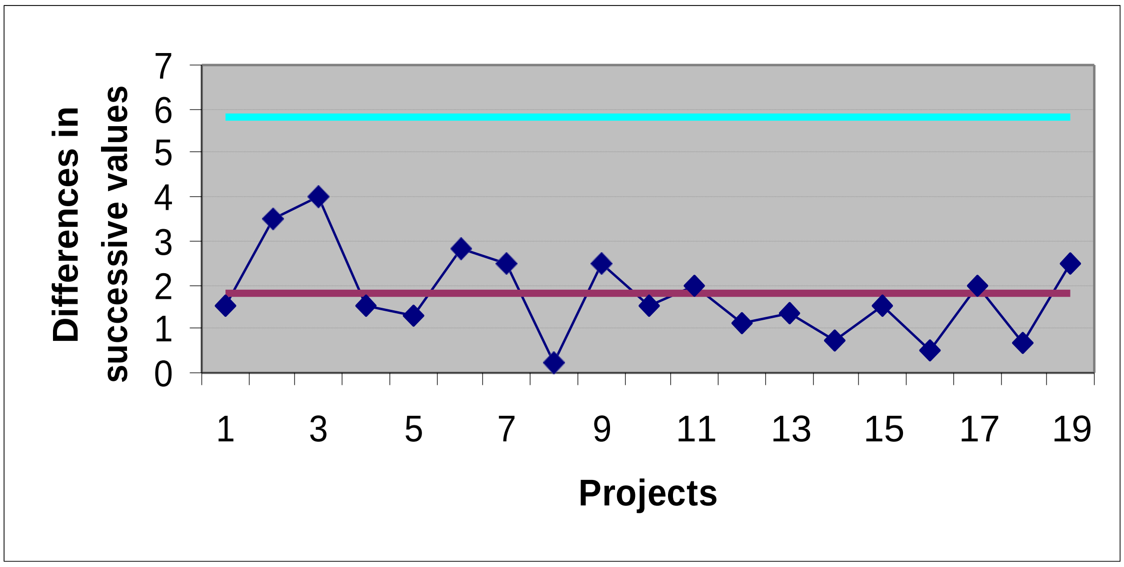
This data is first presented in the graph form as follows:



We now develop the Moving Range Control Chart as follows:

1. Calculate the moving ranges: the absolute values of the successive differences between each pair of data point. Plot these moving ranges on your chart. (the dark blue line)
2. Calculate the mean of the moving ranges. Plot this on the chart. (the red line)
3. Multiply the mean with 3.268. Plot this as the Upper Control Line (UCL). This line is 3 standard deviations above the line. (the light blue line)
4. To determine whether the process metrics description is stable, a simple question is asked: are the moving ranges values inside the UCL? If the answer is yes then the process is stable otherwise it is unstable.

This chart is shown in the following diagram:



This chart is then used to develop the individual control chart as follows:

1. Plot individual metric values
2. Compute the average value for the metrics values - A_m
3. Multiply the mean of moving average by 2.66 and add average computed in step 2 above. The result is Upper Natural Process Limit (UNPL) mcq
4. Multiply the mean of moving average by 2.66 and subtract average computed in step 2 above. The result is Lower Natural Process Limit (LNPL)
5. Plot UNPL and LNPL. If LNPL is less than zero than it need not be plotted unless the metric being evaluated takes on values that are less than 0.
6. Compute a standard deviation as (UNPL – A_m)/3.
7. Plot lines 1 and 2 standard deviations above and below A_m.
8. Applying these steps we derive an individual control chart.

This chart may be used to evaluate whether the changes represented by metrics indicate a process that is in control or out of contro . For this, the following 4 criteria zone rules are used.

1. A single metrics value lies outside UNPL
2. Two out of three successive values lay more than 2 standards deviations away from A_m.
3. Four out of five successive values lay more than one standard deviation away.
4. Eight consecutive values lie on one side of A_m.

If any of these tests passes, the process is out of control otherwise the process is within control. mc

Since none of the test passes for the data in our example, our process is in control and this data can be used for inference.

We now analyze our results. It can be seen that the variability decreased after project 10. By computing the mean value of the first 10 and last 10 projects, it can be shown that the remedial measure taken was successful and resulted in 29% improvement in efficiency of the process. Hence the process changes incorporated were useful and bore fruit.

Lecture No. 16

Interpreting Measurements

MC

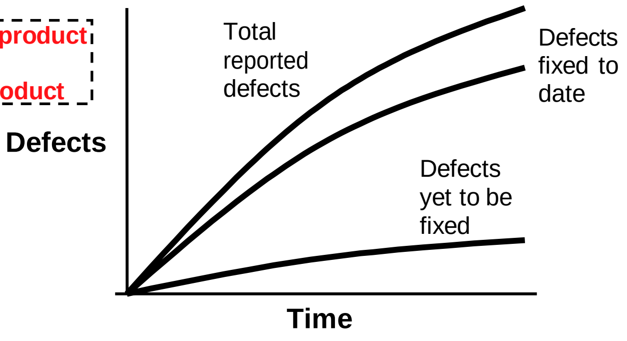
A good metric system is the one which is simple and cheap and at the same time adds a lot of value for the management. Following are some of the examples that can be used for effective project control and management.

We can collect data about the defects reported, and defects fixed and plot them in the following manner, with their difference showing the defects yet to be fixed. This can give us useful information about the state of the product. I the gap between the defects reported and defects fixed is increasing, then it means that the product is in unstable condition. On the other hand if this g p is decreasing then we can say that the product is in a stable condition and we can plan for shipment.

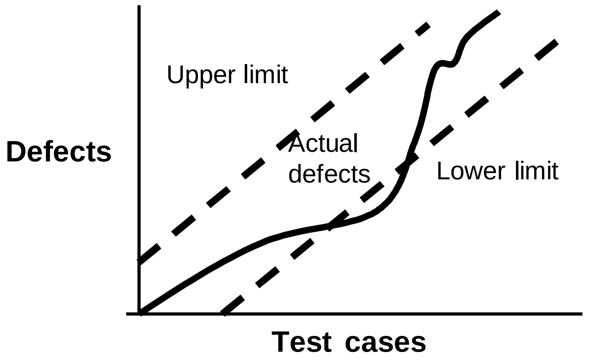
MCQS

gap between defects increases= unstable product

gap between defects decreseas= stable product

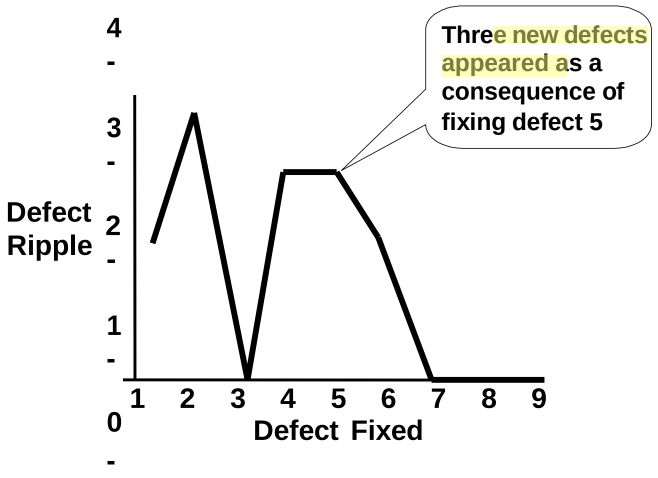


Similarly, we can gain useful information by plotting the defects reported against the number of use cases run. We can use control lines from our previous data and see if the actual defects are within those control limits. If the defects at any given point in time are less than the lower limit then it may mean that out testing team is not doing a good job and coverage is not adequate. On the other hand, if it crosses the upper line then it indicates that the design and coding is not up to mark and we perhaps need to check it.

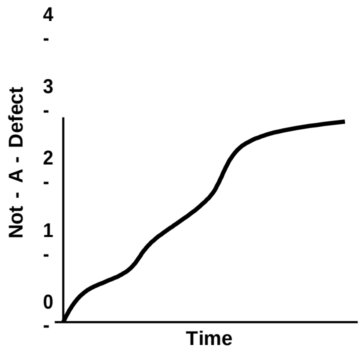


in 2023

Another very simple graph as shown below can give a lot of insight into the design quality. In this case, if the frequency of ripple defects is too large, then it means that then there is tight coupling and hence the design is not maintainable.



The following is yet another very simple and effective way of getting insight into the



quality of the requirements. If a number of defects that are reported by the testing team are ultimately resolved as not-a-defect then there may be a sever problem with the requirements document as two teams (development and testing) are interpreting it differently and hence coming to different conclusions.

Lecture No. 17

Software Project Planning

Software project planning is an activity carried out by the project manager to estimate and address the following points:

1. Software scope estimation
2. Resources requirements
3. Time requirements
4. Structural decomposition
5. Risk analysis and planning

Software scope estimation
Software scope describes the data and control to be processed, function, performance, constraints, interfaces, and reliability. Determination of the software scope is a pre-requisite of all sorts of estimates, including, resources, time, and budget.

MC

In order to understand the scope, a meeting with the client should be arranged. The analyst should start with developing a relationship with the client representative and start with context-free questions. An understanding of the project background should also be developed. This includes understanding:

- Who is behind the request (sponsor)?
- Who will use the solution (users)?
- What are the economic benefits (why)?

Now is the time to address the find out the more about the product. In this context, the following questions may be asked:

- How would you characterize good output?
- What problems will the solution address?
- Can you show me the environment in which the solution will be used?
- Will any special performance issues or constraints affect the way the solution is approached?
- Are you the right person to answer these questions? Are answers "official"?
- Are my questions relevant to the problem that you have?
- Am I asking too many questions?
- Can anyone else provide additional information?
- Should I be asking you anything else?

full form of fast MCQ in 2023

In this regards, a technique known as **Facilitated Application Specification Techniques** or simply **FAST** can be used. This is a team-oriented approach to requirement gathering that is used during early stages of analysis and specification. In this case joint team of customers and developers work together to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of requirements.

Feasibility
The purpose of the feasibility analysis is to determine can we build software to meet the scope? For this purpose, the project is analyzed on the following 4 dimensions:

Technology
- Is the project technically feasible?
- Is it within the state of the art?
- Can defects be reduced to a level matching the application needs?

Finance
- Is it financially feasible?
- Can development be completed at a cost that software organization, its client, or the market can afford?

Time
- Will the project time to market beat the competition?
- Can we complete the project in the given amount of time?

Resources
- Does the organization have resources needed to succeed? The resources include:
    - HW/SW tools
    - Reusable software components
    - People

Software Project Estimation
Once the project feasibility has been determined, the project manager starts the estimation activity. It is a relatively difficult exercise and has not become an exact science. It is influenced by human, technical, environmental, political factors.

For software project estimation, a project manager can use historic data about its organizations previous projects, decomposition techniques, and/or empirical models developed by different researchers.

Empirical Models

Empirical models are statistical models and are based upon historic data. Although there are many different models developed by different researchers, all of them share the following basic structure:

$E = A + B * (ev)^C$ MCQS

where

A, B, c are empirical constants,
'ev' is the effort in terms of lines of code or FP, and
'E' is the effort in terms of person months.

The most famous of these models is the COCOMO – MCQ COnstructive COst MOdel – model. It also has many different versions. The simplest of these versions is given below:

$$E = 3.2\,(KLOC)^{1.05}$$

Some of these models take into account the project adjustment components including problem complexity, staff experience, and development environment.

It is important to note that there are a number of models with each yielding a different result. This means that any model must be calibrated for local needs before it can be effectively used.

The Software Equation

The software equation shown below is dynamic multivariable estimation model. It assumes a specific distribution of effort over the life of the software development project and is derived from productivity data collected for over 4000 projects.

$$E = [LOC \times B^{0.333}/P]^3 \times (1/t^4)$$ MCQ

Where:

- E – effort in person months or person years
- t – project duration in months or years
- B – special skill factor
  - Increases slowly as the need for integration, testing, QA, documentation, and management skills grow
- P – productivity parameter
  - Overall process maturity and management practices
  - The extent to which good SE practices are used
  - The level of programming language used
  - The state of the software environment
  - The skills and experience of the software team
  - The complexity of the application

Buy versus build

It is often more cost-effective to acquire than to build. There may be several different options available. These include:

- Off-the-shelf licensed software
- Software components to be modified and integrated into the application
- Sub-contract

The final decision depends upon the criticality of the software to be purchased and the end cost. The buy versus build decision process involves the following steps:

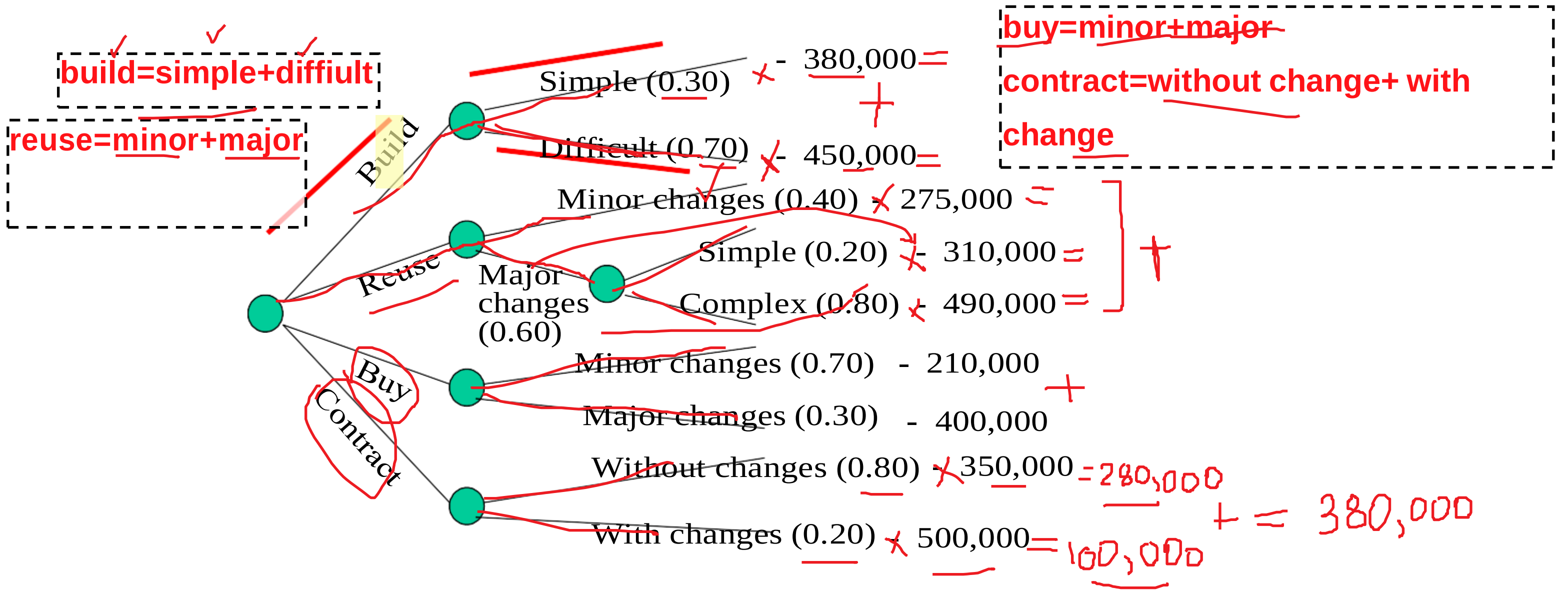
- Develop specification for function and performance of the desired software. Define measurable characteristics whenever possible.
- Estimate internal cost and time to develop
- Select 3-4 candidate applications that best meet your specifications
- Select reusable software components that will assist in constructing the required application
- Develop comparison matrix that presents a head-to-head comparison of key function. Alternatively, conduct benchmark tests to compare candidate software.

- Evaluate each software package or component based on past product quality, vendor support, product direction, reputation, etc.
- Contact other users of the software and ask for opinion.

The following key considerations must always be kept in the perspective:

- Delivery date
- Development Cost
    - Acquisition + customization
- Maintenance Cost

A decision tree can be built to analyze different options in this regards. As an example of this, let us consider the following. In this case, we have four different options, namely, build, reuse, buy, and contract. We analyze each one of these with the help of a decision tree. Each node of the tree is further partitioned a probability is assigned to each branch. At the end, cost for each path in the tree, from root to a leaf, is estimated and associated with that path. This process is shown in the following diagram.


build=simple+diffiult
reuse=minor+major
~~buy=minor+major~~
contract=without change+ with change
Build
Simple (0.30) - 380,000
Difficult (0.70) - 450,000
Reuse
Minor changes (0.40) - 275,000
Major changes (0.60)
Simple (0.20) - 310,000
Complex (0.80) - 490,000
Buy
Minor changes (0.70) - 210,000
Major changes (0.30) - 400,000
Contract
Without changes (0.80) - 350,000 = 280,000
With changes (0.20) - 500,000 = 100,000
+ = 380,000

Once the information in the tree is complete, it is used to determine the most viable option. For each option the expected cost is determined as follows:

Expected cost = ∑ (path probability)$_I$ x (estimated path cost)

Using this formula, we calculate the expected cost of each option as follows:

Build = 0.30*380 + 0.70*450000 = 429000

Reuse = 0.4*275000 + 0.6*0.2*310000 + 0.6*0.8*490000 = 382400

The expected cost of Buy and Contract can also be calculated in a similar fashion and comes out to be:

Buy	= 267000
Contract	= 380000

Therefore, according to this analysis, it is most viable to buy than any other option.

Lecture No. 18

Risk analysis and management

Analysis and management of project risks is also a very important activity that a project manager must perform in order to improve the chances for the project. Robert Charette defines risk as follows:

> *Risk concerns future happenings. Today and yesterday are beyond active concern. The question is, can we, therefore, by changing our action today create and opportunity for a different and hopefully better situation for ourselves tomorrow. This means, second, that risk involves change, such as changes in mind, opinion, action, or places. … [Third,] risks involve choice, and the uncertainty that choice itself entails.*

Q in 2022

Risk analysis and management involves addressing the following concerns:

1. Future - what risks might cause the project to go awry
2. Change – what change might cause the risk to strike
   - How changes in requirements, technology, personnel and other entities connected to the project affect the project
3. Choice – what options do we have for each risk

In Peter Drucker words:

> *while it is futile to try to eliminate risk, and questionable to try to minimize it, it is essential that the risk taken be the right risk.*

There are two basic risk management philosophies, reactive and proactive.

- Reactive – Indiana Jones school of risk management
  – Never worrying about problems until they happened, and then reacting in some heroic way – Indiana Jones style.

Q in

- Proactive
  – Begins long before technical work starts
  – Risks are identified, their probability and impact are analyzed, and they are ranked by importance.
  – Risk management plan it prepared
    - Primary objective is to avoid risk
    - Since all risks cannot be avoided, a contingency plan is prepared that will enable it to respond in a controlled and effective manner

Unfortunately, Indiana Jones style is more suitable for fiction and has a rare chance of success in real life situations. It is therefore imperative that we manage risk proactively.

A risk has two characteristics: Important

- Uncertainty – the risk may or may not happen
- Loss – if the risk becomes a reality, unwanted consequences or losses will occur.

A risk analysis involves quantifying degree of uncertainty of the risk and loss associate with it. In this regards, the PM tries to find answers to the following questions:

- What can go wrong?
- What is the likelihood of it going wrong?
- What will the damage be?
- What can we do about it?

Lecture No. 19

Types of Risks