Software Engineering Fundamentals

Bachelors in Software Engineering

III semester

Chapter two

Software metrics, Software Project Planning and Risk

OUTLINES:

2.1 basic terminologies of software metrics(Measures, Metrics, and Indicators)
2.2 software metrics guidelines
2.3 product, process and project metrics
2.4 Objectives, Scope, Resources, Project Estimation, Decomposition Techniques
2.5 Empirical Estimation Models, Risk Management Strategies
2.6 Software Risks, risk Identification, Risk Projection

Basic Terminologies

- **MEASURE: A quantitative indication of the extent, amount, dimension, or size of some attribute** of a product or process (e.g. number of errors)
- **METRICS: The degree to which a system, component, or process possesses a given attribute. It r**elates several measures (e.g. average number of errors found per person hour).
- **INDICATORS: An indicator is a metric or combination of metrics that provide insight into the** software process, a software project, or the product itself.
- **DIRECT METRICS: Immediately measurable attributes (e.g. line of code, execution speed,** defects reported).

- **INDIRECT METRICS: Aspects that are not immediately quantifiable (e.g. functionality,** quantity, reliability).
- **MEASUREMENT is the process by which numbers or symbols are assigned to the attributes of** the entities in the real world in such a way as to define them accordingly to clearly defined rules.
- **FAULTS:**

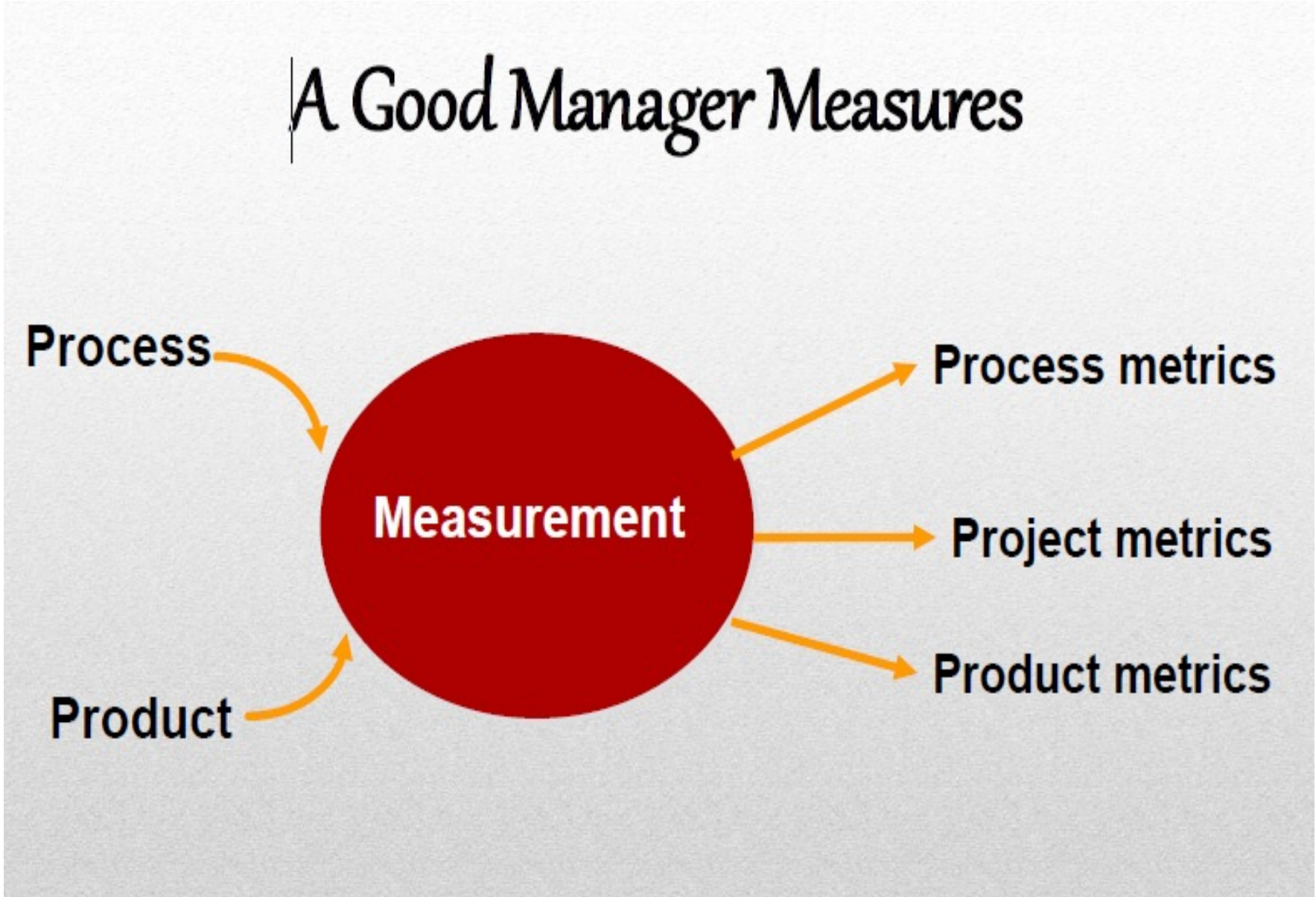
  Errors: Faults found by the developers during software development.

  Defects: Faults found by the customers after release

WHY MEASURE SOFTWARE?

- Establish baselines for comparisons with future assessments.
- To evaluate the status with respect to plans.
- Predict qualities of a product or a process by gaining understandings of relationships among process and products.
- Improve product quality and process performance by identifying roadblocks and inefficiencies.

A Good Manager Measures



METRICS GUIDELINES

- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who collect measures and metrics.
- Don’t use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them.
- Never use metrics to threaten individuals or teams.
- Metrics data that indicate a problem area should not be considered “negative.” These data are merely an indicator for process improvement.
- Don’t obsess on a single metric to the exclusion of other important metrics.

Product metrics

- Product metrics are quantifiable measurements used to assess and track the performance, usage, and success of a product, aiding in data-driven decision-making and optimization.
- Product metrics help the software engineers gain insight into the design and construction of the software they build by focusing on specific and measurable attributes of the work products.
- These metrics examine the requirements model with the intent of predicting the size and complexity of the resulting system.

Function-oriented metrics

- The Function point (FP) metric can be used effectively as a means for measuring the functionality delivered by a system.
- The FP metric can be used to -

a. Estimate the cost or effort required to design, code and test the software.

b. Predict the number of errors that will be encountered during testing

c. Forecast the number of components in the system

SIZE-ORIENTED METRICS

- Size oriented metrics are derived by normalizing quality or productivity measures by considering the size of the software that has been produced.
- A simple set of size-oriented metrics can be developed for each project :

  Errors per KLOC

  Defects per KLOC

  Cost per KLOC

  Pages of documentation per KLOC

PROCESS METRICS

- Process metrics are used for strategic purposes.
- Process metrics are collected across all projects and over long period of time. Their intent is to provide a set of process indicators that lead to long-term software process improvement.
- The only way to improve any process is to measure specific attributes of the process, develop a set of meaningful metrics based on the outcomes and then use the metrics to provide indicators that will lead to strategic improvements.

PROCESS METRICS contd…

- Quality-related
  - focus on quality of work products and deliverables
- Productivity-related
  - Production of work-products related to effort expended
- Statistical SQA data
  - error categorization & analysis
- Defect removal efficiency
  - propagation of errors from process activity to activity
- Reuse data
  - The number of components produced and their degree of reusability

PROJECT METRICS

- used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.
- every project should measure:
  - *inputs*—measures of the resources (e.g., people, tools) required to do the work.
  - *outputs*—measures of the deliverables or work products created during the software engineering process.
  - *results*—measures that indicate the effectiveness of the deliverables.

Typical Project Metrics

- Effort/time per software engineering task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Changes (number) and their characteristics
- Distribution of effort on software engineering tasks

Software Scope

- Describes:
  - the functions and features that are to be delivered to end users;
  - the data that are input and output;
  - the “content” that is presented to users as a consequence of using the software; and
  - the performance, constraints, interfaces, and reliability that *bound* the system.
- Scope is defined using one of two techniques:
  - A narrative description of software scope is developed after communication with all stakeholders.
  - A set of use cases is developed by end users.
- In general sense, the software scope indicates the acceptance limit of software by end user or environment.
- must be unambiguous and understandable at the management and technical level.
- Can be answered: Is the project feasible?

Resources

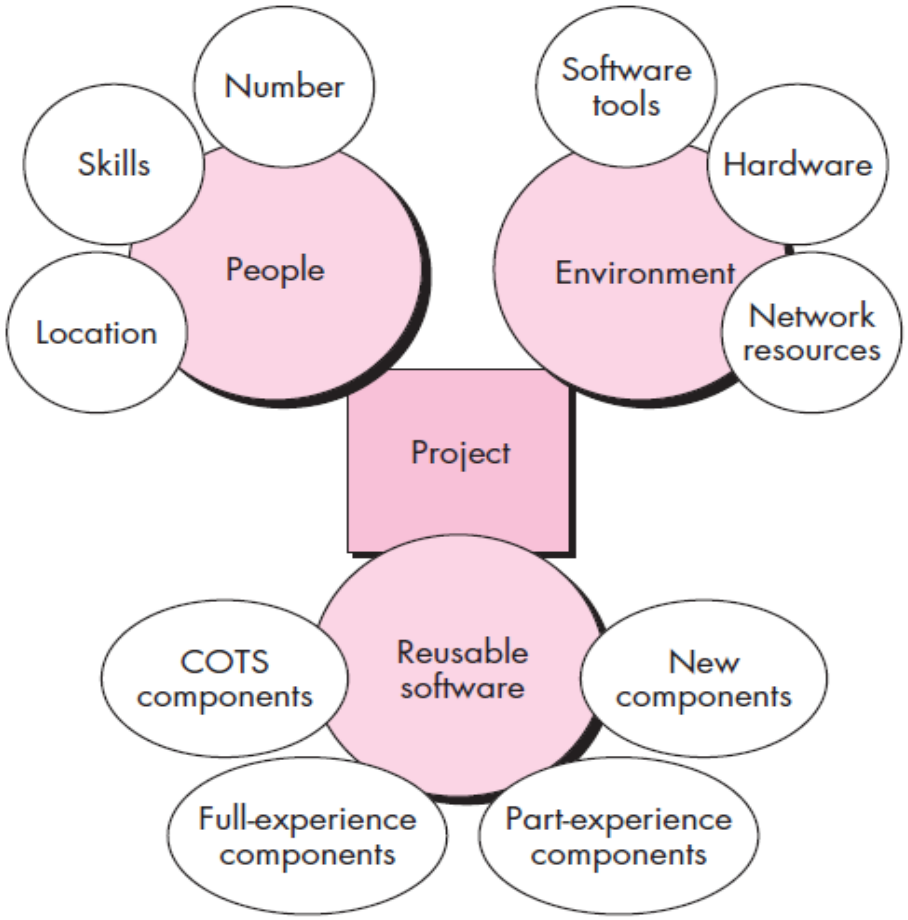
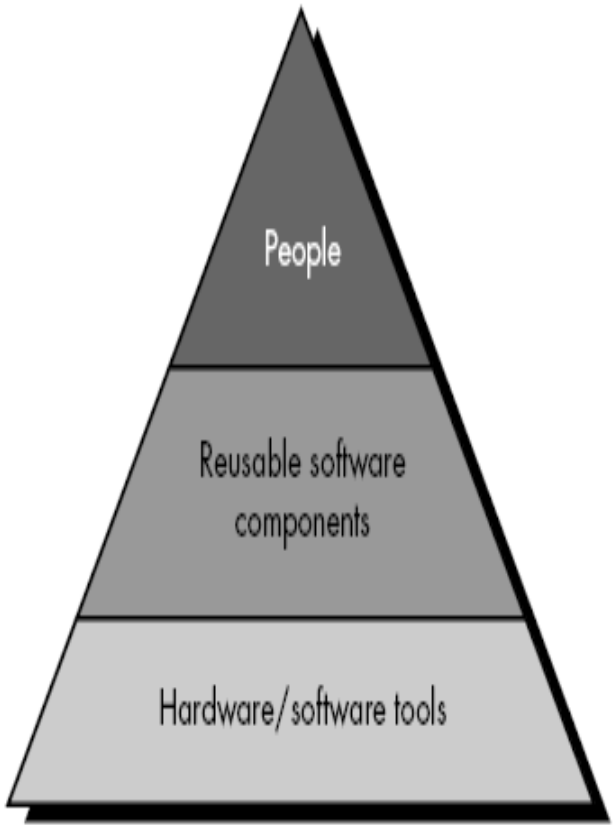


Fig. Project resources



1. Human Resources

- Human are the primary resources that evaluate software scope and select the skills required to complete development.
- They specify both organizational position (e.g., manager, senior software engineer) and specialty (e.g., telecommunications, database, and client-server).
- The number of people required for a software project can be determined only after an estimate of development effort (e.g., person-months) is made.
- For relatively small projects (a few person-months), a single individual may perform all software engineering tasks, consulting with specialists as required.
- For larger projects, the software team may be geographically dispersed across a number of different locations. Hence, the location of each human resource is specified.

2. Reusable Software Resources

1. Off-the-shelf components
2. Full-experience components
3. Partial-experience components
4. New components

2. Reusable Software Resources

1. **Off-the-shelf components**
   - Existing software that can be acquired from a third party or from a past project.
   - COTS (commercial off-the-shelf) components are purchased from a third party, are ready for use on the current project, and have been fully validated.

2. **Full-experience components**
   - Existing specifications, designs, code, or test data developed for past projects that are similar to the software to be built for the current project.
   - Members of the current software team have had full experience in the application area represented by these components.
   - Therefore, modifications required for full-experience components will be relatively low risk.

2. Reusable Software Resources

3. Partial-experience components

- Existing specifications, designs, code, or test data developed for past projects that are related to the software to be built for the current project, **but will require substantial modification.**
- Members of the current software team have only limited experience in the application area represented by these components.
- Therefore, modifications required for partial-experience components have a fair degree of risk

4. New components

- Software components must be built by the software team specifically for the need of the current project.

3. Environmental Resources

- incorporates hardware and software.
- Hardware provides a platform that supports software required to produce the work products
- When a computer-based system (incorporating specialized hardware and software) is to be engineered, the software team may require access to hardware elements being developed by other engineering teams.
- For example, software for a robotic device may require a specific robot as part of the validation test step.

Software Project Estimation

- Software is the most expensive element of all computer system.
- Inappropriate estimation can make a big difference between the profit and loss.
- Software cost and effort estimation will never be an exact science.
- Too many variables- human, technical, environmental, political- can affect the cost and effort for software development.
- However, software project estimation can be done after considering these aspects:

Software Project Estimation

- A general project estimation can be done by:
  i. Delaying estimation until late in the project.
  ii. Estimation projects based on similar projects in the past.
  iii. Using simple relatively decomposition technique.
  iv. Using one or more empirical models for software cost and effort estimation.

- *First two techniques are not efficient.*

Software Project Estimation

Decomposition Technique:

- works on “Divide and Conquer” approach
- project is divided into different components and related software engineering activities.
- Cost and effort estimation can be performed step by step on each component.
- Software cost estimation is a form to solve the problems.
- Most of the times problem to be solved is too complex to be solve in a single step.
- Then the problem is decomposed into number of components in order to achieve an accurate cost estimate.
- There are two approaches in decomposition technique:
    i. Problem based estimation: Problem decomposed. LOC and FP
    ii. Process based estimation: Process is decomposed into a relatively small set of tasks and effort required to accomplish each task is estimated.

2.5.1 LOC Based Estimation

- LOC is a software metric used to measure the size of a software program by counting the no. of lines in the text of the program's source code.
- In LOC, lines used for commenting the code and the header lines should be ignored.
- The software is divided into different modules and the LOC is estimated for each module.
- The LOC count cannot be accurately computed before the actual coding. Hence, this method is less used.
- For eg:

```
#include <stdio.h>
int main()
{
    printf("Hello World");
    return 0;
}
```

Here, the line of code (LOC) =5

Logical line of code (LLOC) =2

Advantages	Disadvantages
• Simple to measure	• Cannot measure the size of specification. • Takes no account of functionality or complexity. • Bad software design may cause more LOC. • Language dependent.

For example: If

- Estimated total LOC on any project = 33,200
- Organizational average productivity = 620 LOC/pm
- Labor rate per month = $8000

Then,

- Cost per LOC = ($8000/m)/(620LOC/pm) = $13
- Total project cost = Total LOC x Cost per LOC = 33,200 x $13 = $4,31,600
- Estimated effort = Total LOC / Average productivity

  = (33,200 LOC) / (620 LOC/person month) = 54 persons

FP based estimation: COMPUTING FUNCTIONAL POINTS

- Function point is a unit of measurement to express the amount of business functionality that a software provides to a user.
- It measures functionality from user's point of view.
- The software product is directly dependent on the number of functions or features it supports.
- Information domain values are defined in the following manner:
  - Number of external inputs (EIs)
  - Number of external outputs (EOs)
  - Number of external inquiries (EQs)
  - Number of internal logical files (ILFs)
  - Number of external interface files (EIFs)

Analyzing the Information Domain

Information Domain Value	Count		Weighting factor simple	average	complex		
External Inputs (EIs)		X	3	4	6	=	
External Outputs (EOs)		X	4	5	7	=	
External Inquiries (EQs)			3	4	6	=	
Internal Logical Files (ILFs)		X	7	10	15	=	
External Interface Files (EIFs)		X	5	7	10	=	
Count total	→			→			

Computation of FP:

Mathematically;

FP= Count Total x CAF

Where, Count Total= sum of all FP entries

$CAF = 0.65 + 0.01 \times \sum F_i$

CAF → Complexity Adjustment Factor

The F_i (i=1 to 14) is the value adjusted factor based on response to 14 questions.

The Fi (i=1 to 14) are ***value adjustment factors (VAF) based on responses to the*** following questions.

```
string aspects[14] = {
    "reliable backup and recovery required ?",
    "data communication required ?",
    "are there distributed processing functions ?",
    "is performance critical ?",
    "will the system run in an existing heavily utilized operatior
    "on line data entry required ?",
    "does the on line data entry require the input transaction to
    "are the master files updated on line ?",
    "is the inputs, outputs, files or inquiries complex ?",
    "is the internal processing complex ?",
    "is the code designed to be reusable ?",
    "are the conversion and installation included in the design ?'
    "is the system designed for multiple installations in differer
    "is the application designed to facilitate change and ease of
};
```

- 0 - No Influence
- 1 - Incidental
- 2 - Moderate
- 3 - Average
- 4 - Significant
- 5 - Essential

FACTOR	VALUE (Fi)
1. Back-up and Recovery ?	4
2. Data Communication ?	2
3. Distributed Processing ?	0
4. Performance Critical ?	4
5. Existing Operational Environment ?	3
6. On-line Data Entry ?	4
7. Input transactions over multiple Screens?	5
8. Online Updates ?	3
9. Information Domain Values Complex ?	5
10. Internal Processing Complex?	5
11 Code Designed for reuse?	4
12. Conversion / installation in Design?	3
13. Multiple Installations?	5
14. Application Designed for change ?	5
$\sum$ (Fi)	52

Numerical: Given the following values, Compute FP when all complexity adjustment factors and weighting factors are average.

Information Domain Value	Count	Weighting Factor		
		Simple	Average	Complex
External Inputs (EI)	50	3	4	6
External Outputs (EO)	40	4	5	7
External Inquiries (EQ)	35	3	4	6
Internal Logical Files (IFL)	6	7	10	15
External Interface Files (EIF)	4	5	7	10

Solution: FP is given by :

$$FP = \text{Count Total} \times CAF$$

Now , Count Total $= 50*4 + 40*5 + 35*4 + 6*10 + 4*7 = 628$

$$CAF = 0.65 + 0.01 \times \sum F_i$$

$$= 0.65 + 0.01 \times (14 \times 3) = 1.07$$ *(Here 14 is taken for 14 questions to be answered)*

Hence $FP = 628 \times 1.07 = 672$

For example:

If FP = 375 FP, average productivity = 6.5 FP/PM and labor rate = $8000 per month then

- Cost per FP = ($8000) / (6.5 FP/PM) = $1230
- Total project cost = 375 x $1230 = $461250
- Estimated effort = Total FP / Average productivity = 375 FP / (6.5 FP/PM) = 58 persons

The COCOMO I

- It is the previous version of COCOMO II.
- COCOMO consists of a hierarchy of three increasingly detailed and accurate forms namely: Basic COCOMO, Intermediate COCOMO, and Detailed COCOMO
- The COCOMO model assumes every project is developed in one of the three models:
  a) *Organic mode:* The project requires little innovation
  b) *Semi-organic mode:* Intermediate between organic and embedded mode.
  c) *Embedded mode:* Requires a great deal of innovation

i) The Basic COCOMO:

The BASIC COCOCMO model computes software development effort and cost as a function of program size expressed in Estimated Lines of Code (KLOC). It is good for quick and rough estimation of software costs. Its accuracy is limited due to it does not consider the cost drivers.

- It take the form:

Effort(E) = a_b * (KLOC)$^{b}_{b}$(in Person-months)

DevelopmentTime(D) = c_b * (E) $^{d}_{b}$ (in month)

Average staff size(SS) = E/D (in Person)

Productivity(P) = KLOC / E (in KLOC/Person-month)

Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Example:

The size of organic software is estimated to be 32,000 LOC. The average salary for software engineering is Rs. 15000/- per month. What will be effort and time for the completion of the project?

Solution:

- Effort applied = 2.4 x (32)1.05 PM = 91.33 PM (Since: 32000 LOC = 32KLOC)
- Time = 2.5 x (91.33)0.38 Month = 13.899 Months
- Cost = Time x Average salary per month = 13.899 x 15000 = Rs. 208480.85
- People required = (Effort applied) / (development time) = 6.57 = 7 persons

Practise:

- A project size of 200KLOC is to be developed. S/W development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the effort and development time of the project.

ii) Intermediate COCOMO

It is an extension of Basic COCOMO model that computes software development effort by adding a set of "Cost Drivers" that will determine the effort and duration of the project, such as assessment of personnel and hardware.

- Multiply all 15 Cost Drivers to get **Effort Adjustment Factor(EAF)**
- **E(Effort) = $a_b(KLOC)^{b_b}$ * EAF** (in Person-Month)
- **D(Development Time) = $c_b(E)^{d_b}$** (in month)
- **SS (Avg Staff Size) = E/D** (in persons)
- **P (Productivity) = KLOC/E** (in KLOC/Person-month)

Project	a_b	b_b	c_b	d_b
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Table: Intermediate COCOCMO coefficients

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Table: Intermediate COCOCMO EAF coefficients

Example:

A new project with estimated 400 KLOC embedded system has to be developed. Project manager hires developers of low quality but a lot of experience in programming language. Calculate the Effort, Development time, Staff size & Productivity.

- EAF = 1.29 * 0.95 = 1.22
- 400 KLOC implies Embedded System
- Effort = 2.8*(400)1.20 * 1.225 = 3712 * 1.22 = 4528 person-months
- Development Time = 2.5 * (4528)0.32 = 2.5 * 14.78 = 36.9 months
- Avg. Staff Size = E/D = 4528/36.9 = 122 persons
- Productivity = KLOC/Effort = 400/4528 = 0.0884 KLOC/person-month.

iii) Detailed COCOMO

It is an extension of the Intermediate model that adds effort multipliers for each phase of the project to determine the cost-drivers impact on each step. Detailed COCOMO used the same equation for estimation as the Intermediate model, but for each development phases.

Detailed COCOMO = Intermediate COCOMO+ assessment of Cost Drivers impact on each phase.

Phases:

1)Plans and requirements
2)System Design
3) Detailed Design
4)Module code and test
5)Integrate and test

Cost of each subsystem is estimated separately. This reduces the margin of error.

- Multiply all 15 Cost Drivers to get **Effort Adjustment Factor(EAF)**
- **$E(\text{Effort}) = a_b(\text{KLOC})^{b_b} * \text{EAF}$** (in Person-Month)
- **$D(\text{Development Time}) = c_b(E)^{d_b}$** (in month)
- **E_p (Total Effort) = $\mu_p * E$** (in Person-Month)
- **D_p (Total Development Time) = $\tau_p * D$** (in month)

RISK?

- a potential problem that may or may not occur in the future.
- It is the uncertainty, a irrelevant future occurrence or happening.

EFFECTS of RISK

- Project diversion from its actual performance
- Unable to meet proper time-lines and thus overall success is affected.
- Quality degradation

• Since risks are the factors for project failure and quality degradation, measures should be taken to avoid them. Thus, the overall process of identifying its impact and making plans in advance is called the risk analysis and management.

RISK HANDLING STRATEGIES

1. Reactive Risk Strategy
2. Proactive Risk Strategy

RISK HANDLING STRATEGIES

1. Reactive Risk Strategy

- concerned with the process of finding solutions once the problem exists,
- the software team does nothing about risk until something goes wrong.
- The software engineer or team never worry about the problems until they happen,
- This is often called a *fire-fighting mode*.

RISK HANDLING STRATEGIES

2. Pro-active Risk Strategy

- thinking about possible risks in a project before they occur.
- begins long before technical work is initiated and
  - potential risks are identified,
  - their probability and impact are assessed,
  - and they are ranked by importance.
- Then, the software team establishes a plan for managing risk.
- The primary objective is to avoid risk, but all risks cannot be avoided, so exercise to make them minimal.

SOFTWARE RISKS

1. Project Risk
2. Technical Risk
3. Business Risk
4. Known Risk
5. Predictable Risk
6. Unpredictable Risk

- Risk concerns the future happening that always involves two characteristics:
  - (a) *uncertainty*—the risk may or may not happen; that is, and
  - (b) *loss*—if the risk becomes a reality, unwanted losses will occur.
- When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk. To accomplish this, different categories of risks are considered.

1) Project risk: *threaten the project plan.*

- Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.

2) Technical risks: *threaten the quality and timeliness*

- Technical risks identify potential design, implementation, interface, verification, and maintenance problems.
- In addition, specification ambiguity, technical uncertainty, technical obsolescence, and “leading-edge” technology are also risk factors.
- Technical risks occur because the problem is harder to solve than expected.

3) **Business risks:** *threaten the feasibility*

- It includes the top five business risks which are
  - *Market risk:* system that no one really wants
  - *Strategic risk:* product no longer fits into the overall business strategy
  - *Sales risk:* the sales force doesn't understand how to sell the product
  - *Management risk:* losing the support of senior management
  - *Budget risks:* losing budgetary or personnel commitment

4) Known risks

- Risk is known and its effects are largely known.
- A known risk is where there is a clear indication, or enough information or history available, to establish that a risk exists.
- Known risks are the risks we know about and we also know how big they are.
- For example, an organization may know that there is a risk of them losing some of their customers to a new competitor, and that they risk losing 10% of their customers. The organization knows the risk exists and can quantify it as well. These are the risks that are the easiest to manage because all the required information is present.
- To manage known known risks, the organization simply has to ensure that it is ready for the expected impact. One feature can be integrating a risk management methodology, combined with business process workflows and integrated management change.

5) Predictable risks

These risks are identified from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced).

6) Unpredictable risks

They can and do occur, but they are extremely difficult to identify in advance.

Risk Identification

- Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.).
- Risks can be removed or avoided only if they can be identified.
- By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.
- The different categories of risk (project, technical, business, known, predictable, and unpredictable) can be further divided as:

    1. Generic Risk: *risk common to all*
    2. Product-specific Risk : *associated to only specific product*

One method for identifying risks is to create a **risk item checklist**; we need to identify the following area from where risk will appear:

1. *Product size*—risks associated with the overall size of the software to be built or modified.
2. *Business impact*—risks associated with constraints imposed by management or the marketplace.
3. *Stakeholder characteristics*—risks associated with the sophistication of the stakeholders and the developer's ability to communicate with stakeholders in a timely manner.
4. *Process definition*—risks associated with the degree to which the software process has been defined and is followed by the development organization.
5. *Development environment*—risks associated with the availability and quality of the tools to be used to build the product.
6. *Technology to be built*—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
7. *Staff size and experience*—risks associated with the overall technical and project experience of the software engineers who will do the work.

Risk components:

The risk components defined by U.S. air force are defined in the following manner:

1. *Performance risk*—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
2. *Cost risk*—the degree of uncertainty that the project budget will be maintained.
3. *Support risk*—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.
4. *Schedule risk*—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

The impact of each risk component is divided into one of four categories—

1. negligible,
2. marginal,
3. critical, or
4. catastrophic.

RISK PROJECTION

- Risk Projection (***aka Risk Estimation)***
- is the process of rating the risk based upon its likelihood or probability of occurrence and the consequence of the problem associated with the risk.
- It is carried out by the project planner along with other managers and technical staffs.
- During risk estimation, the first step is the prioritizing risks and hence we can allocate resources where they will have the most impact.
- The process of categorization of various possible risks that may appears in the project is called ***risk assessment***.
- The analysis of nature, scope and time are main component of risk assessment.

RISK PROJECTION (Contd.)

This process involves different activities like:

- Establishing a scale for the likelihood of a risk.
- Determining a scale for the likelihood (probability) of a risk.
- Determining and listing out the consequences of the risk.

RISK TABLE

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
Σ				
Σ				
Σ				

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

Risk Category:
PS = Project size risk
BU = Business risk
TE = Technology risk
DE = Development risk
ST = Stakeholder risk
CU =Customer risk

Fig. RISK TABLE

- A risk table provides a simple technique for risk projection.
- The risk table can be implemented as a spreadsheet model.
- This enables easy manipulation and sorting of the entries.
- Here, at first all risks are listed in the first column of the table.
- This can be accomplished with the help of the risk item checklists reference.
- Each risk is categorized in the second column.
- The probability of occurrence of each risk is entered in the next column of the table.
- The probability value for each risk can be estimated by team members individually.

RISK TABLE (Contd.)

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
Σ				
Σ				
Σ				

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

Risk Category:
PS = Project size risk
BU = Business risk
TE = Technology risk
DE = Development risk
ST = Stakeholder risk
CU =Customer risk

Fig. RISK TABLE

- Finally, the table is sorted by probability and by impact.
- High-probability, high-impact risks percolate to the top of the table, and low-probability risks drop to the bottom.
- This accomplishes first-order risk prioritization.
- After that the table is sorted and defines a cutoff line.
- The *cutoff line* (drawn horizontally at some point in the table) implies that only risks that lie above the line will be given further attention.
- Risks that fall below the line are reevaluated to accomplish second-order prioritization.
- The column labeled RMMM contains a pointer into a *risk mitigation, monitoring, and management plan.*

RISK REFINEMENT

- During early stages of project planning, a risk may be stated quite generally.
- But as time passes, we become more familiar learn about the project and risk. It might be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor and manage.
- One way to do this is to represent the risk in *condition-transition-consequence* (CTC) format. That is, the risk is stated in the following form:

  Given that <condition> then there is concern that (possibly) <consequence>.

Example: Insufficient testing of critical components-> Increased likelihood of defects or failures -> Delays in project schedule, increased rework, and customer dissatisfaction.

RMMM and RMMM PLAN

- RMMM: Risk Mitigation, Monitoring and Management Plan
- All of the risk analysis activities assist the project team in developing a strategy for dealing with risk.
- An effective strategy must consider three issues: risk avoidance, risk monitoring, and risk management and contingency planning.
- Risk Mitigation is a problem avoidance activity,
- Risk Monitoring is a project tracking activity,
- Risk Management includes contingency plans that risk will occur. The goal of the risk mitigation, monitoring and management plan is to identify as many potential risks as possible

1. Risk Mitigation

- The risk mitigation is proactive approach to adopting always the best strategy to reduce future possibilities of risk.
- Related to risk planning, through risk mitigation, the team develops strategies to reduce the possibility or the loss impact of a risk.
- Risk mitigation produces a situation in which the risk items are eliminated or otherwise resolved.

2. Risk Monitoring

- After risks are identified, analyzed, and prioritized, and actions are established, it is essential that the team regularly monitor the progress of the product and the resolution of the risk items, taking corrective action when necessary.
- This monitoring can be done as part of the team project management activities or via explicit risk management activities.
- Often teams regularly monitor their “Top 10 risks.”
- Risks need to be revisited at regular intervals for the team to reevaluate each risk to determine when new circumstances caused its probability and/or impact to change.
- At each interval, some risks may be added to the list and others taken away. Risks need to be reprioritized to see which are moved “above the line” and need to have action plans and which move “below the line” and no longer need action plans.
- A key to successful risk management is that proactive actions are owned by individuals and are monitored.

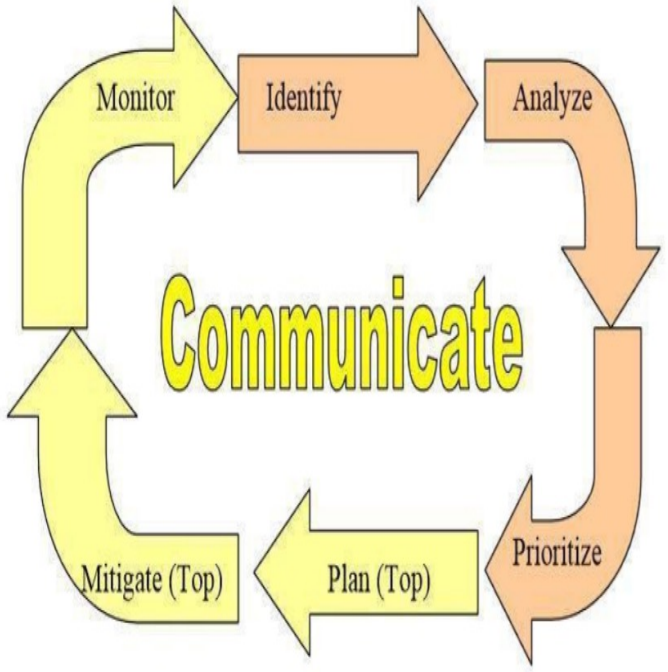
In the case of **high staff turnover**, the following factors must be monitored:

- The general attitude of team members based on project pressures,
- The degree to which the team has jelled (i.e. begun to work well)
- Interpersonal relationships among team members,
- Potential problems with compensation and benefits, and
- The availability of jobs within the company and outside

In addition to monitoring these, other risk monitoring parameters are effectiveness of risk mitigation steps, documentation and if a newcomer were forced to join the software team somewhere in the middle of the project.

3. Risk Management

- The risk management process can be broken down into two interrelated phases, risk assessment and risk control.
- These phases are further broken down.
- Risk assessment involves risk identification, risk analysis, and risk prioritization.
- Risk control involves risk planning, risk mitigation, and risk monitoring.
- It is essential that risk management be done iteratively, throughout the project, as a part of the team's project management routine.



RMMM PLAN

- The RMMM plan is a document in which all the risk analysis activities are described.
- The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.
- Some software teams do not develop a formal RMMM document.
- Rather, each risk is documented individually using a *risk information sheet* (RIS).
- In most cases, the RIS is maintained using a database system so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily.
- The format of the RIS is illustrated in figure.
- Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence.

Risk information sheet

Risk ID: P02-4-32	Date: 5/9/09	Prob: 80%	Impact: high

Description:
Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Refinement/context:
Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards.
Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.

Mitigation/monitoring:
1. Contact third party to determine conformance with design standards.
2. Press for interface standards completion; consider component structure when deciding on interface protocol.
3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.

Management/contingency plan/trigger:
RE computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly.
Trigger: Mitigation steps unproductive as of 7/1/09.

Current status:
5/12/09: Mitigation steps initiated.

Originator: D. Gagne	Assigned: B. Laster

Thank You