Software Project Planning and Risk

1: Objectives ,scope, resources, project estimates, decomposition techniques
2:Empirical estimation models, risk management strategies
3:Software risks, risk identification, risk projection

Project planning

- Software planning involves estimating how **much time, effort, money, and resources** will be required to build a specific software system.
- After the project scope is determined and the problem is decomposed into smaller problems, software managers **use historical project data (as well as personal experience and intuition)** to determine estimates for each. The final estimates are typically adjusted by taking project complexity and risk into account. The resulting work product is called a project management plan.

Project planning objective:

- The **objective of software project planning** is to provide a framework that enables the manager **to make reasonable estimates of resources, cost and schedule.**
- **In addition , estimates should attempt to define best-case and worst-case scenarios.**

- Project plan must be adapted and updated as the project proceeds.

Task set for project planning:

1.establish project scope.
2.determine feasibility
3.analyze risk
4.define required resources
a. determine required human resources.
b. define reusable software resource
c. identify environmental resources
5.estimate cost and effort
a. decompose the problem
b. develop two or more estimates using
Size, function points, process tasks or use cases
c. reconcile the estimates
6.develop a project schedule.

Software scope and feasibility

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.

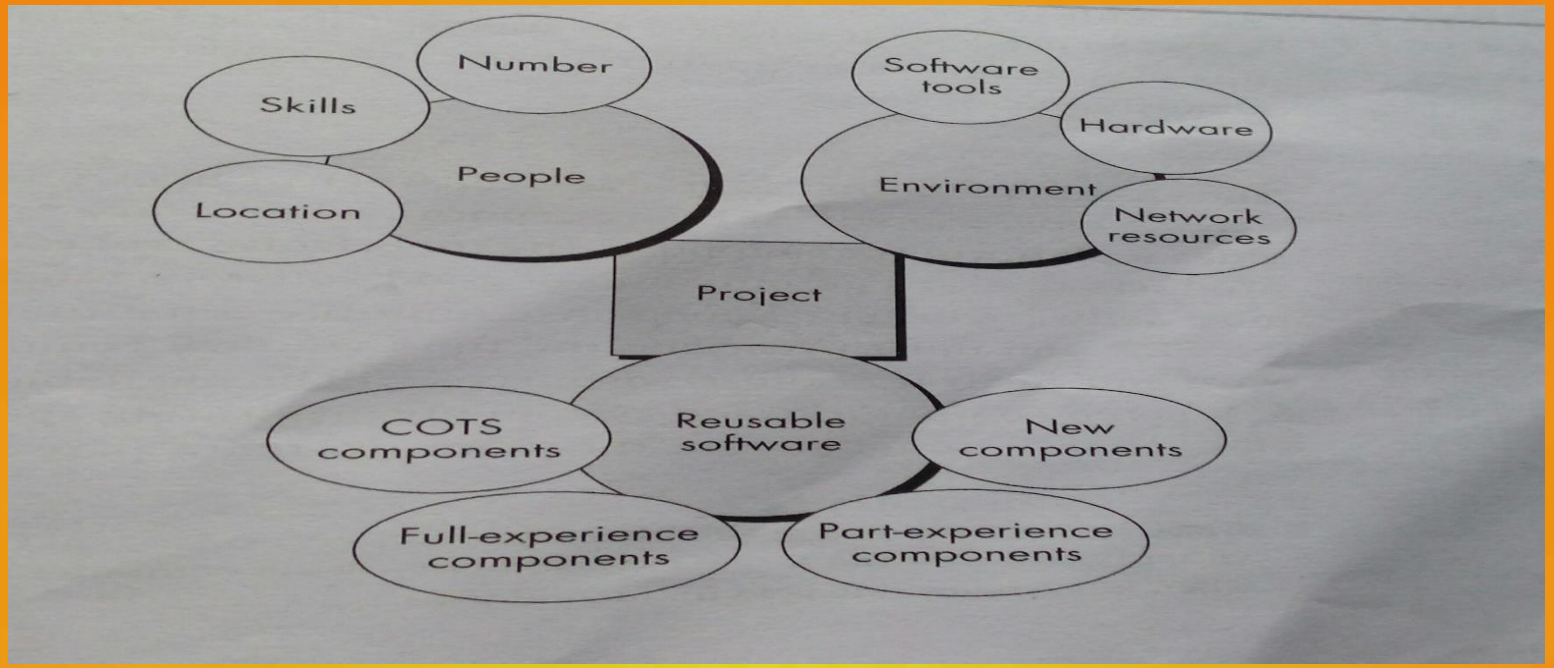
Scope is defined using one of two techniques:

1. A narrative description of software scope is developed after communication with all stakeholders.
2. A set of use cases is developed by end users.

- **Often functions described in the software scope statement are refined to allow for better estimates of cost and schedule(because the cost and schedule estimates is functionality oriented)**
- **Performance encompasses processing and response time requirements.**
- **Constraints identify limits placed on the software by external hardware, available memory, or other existing systems.**
- **Once the scope has been identified(with the concurrence of customer),it is reasonable to ask can we build software to meet this scope? Is the project feasible?**
- Once scope is understood, you must work to determine if it can be accomplished within the dimensions of available technology, cost , time and resources.
- This is a crucial although often overlooked part of the estimation process.

Resources

- The second planning task is estimation of the resources required to accomplish the software development effort.



- Each resources is specified with four characteristics:
  - Description of the resources.
  - A statement of availability .
  - Time when the resource will be required.
  - Duration of time that the resource will be applied.

Human Resources

- The planner begins by evaluating software scope and selecting the skills required to complete development.
- Both organizational position(e.g manager, senior software engineer) and specialty(e.g database) are specified.
- For small project a single individual may perform all software engineering tasks consulting with specialist as required.
- For larger projects, the software team may be geographically dispersed across number of different locations.
- The number of people required for a software project can be determined only after an estimate of development effort(e.g person-month) is made.

Reusable software resources

- Component based software engineering(CBSE) emphasizes reusability i.e the creation and use of software building blocks.
- There are four software resource categories that should be considered as planning proceeds:
- **Off-the-shelf components (**ready-made)

(existing software that can be acquired from third party or from the past project.

- Full experience components (similar past project parts) (existing specifications, designs, code or test data developed for past projects that are similar to the software to be built for the current project)

- **Partial-experience component (somewhat related, needs changes)**

(existing specification designs, code or test data developed for past projects that are related to the software to be built for the current project but require substantial modification)

- **New components (built from scratch)**

(components built by the software team specifically for the needs of the current project.)

Environmental resources

- The environment **that supports a software project**, often called the software engineering environment(SEE), incorporates **hardware and software**.
- Hardware provides a platform that supports the software required to produce the work products that are an outcome of good software engineering practice.
- When a computer based system is to be engineered, the software team may require access to hardware elements being developed by other engineering teams**. Example** *software for a robotic device use within a manufacturing cell may require a specific robot.*
- Each hardware element must be specified as part of planning.

Software project estimation

- Software cost and effort estimation will never be an exact science.
- Too many variables: **human, technology, environment, political environment** can affect the ultimate cost of software and effort applied to develop it.
- However, software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk.

1) Delay estimation until late in the project(obviously we can achieve 100% accurate estimates after the project is completed).
2) Base estimates on similar projects that have already been completed
3) Use relatively simple decomposition techniques to generate project cost and effort estimates.
4) Use one or more empirical models for software cost and effort estimation.

DECOMPOSITION TECHNIQUES

Why? (Used because estimating the entire project at once is too complex)

Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex **to be considered in one piece.**

For this reason we should decompose the problem, re-characterizing it as a set of smaller problems

1.Software sizing

Estimation accuracy

The degree to which the planner has properly estimated **the size of the product to be built;**

The ability to translate the size estimate **into human effort, calendar time, and dollars**

Team capability

The degree to which the project plan reflects **the abilities of the software team**

Requirements stability

The stability of product requirements **and the environment that supports the software engineering effort.**

Different approaches to the sizing problem:

"Fuzzy logic" sizing (Approximate, high-level guessing using ranges)
This approach uses the approximate reasoning techniques
The planner must identify the type of application**, establish its** magnitude **on a qualitative scale, and then** refine the magnitude **within the original range.**

Function point sizing.
The planner develops estimates of the information domain **characteristics**
Counts: User inputs, Outputs, Files, Inquiries, Interfaces

Standard component sizing:

Software is composed of a number **of different "standard components" that are generic to a particular application area.**
e.g. Information system are subsystems:
modules, screens, reports, interactive programs, batch programs, files,

Change sizing: Used when modifying existing software.
This approach is used when a project encompasses the use of existing software **that must be modified in some way as part of a project.**
The planner estimates the number and type
(e.g., reuse, adding code, changing code, deleting code)

2.Problem based estimation

LOC and FP data are used in two ways during software project estimation:

(1) as an estimation variable **to "size" each element of the software and**

(2) as baseline metrics **collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.**

3.process based estimation

Estimating a project based on **tasks** in the development process.

The process is decomposed **into a relatively small set of tasks and the effort required to accomplish each task is estimated.**

Effort = Sum of effort for each task

Cost = Effort × labor rate

Empirical estimation models

Uses empirically derived formulas **to predict effort as a function of LOC or FP.**

a. The Structure of Estimation Models

$$E = A + B \times (ev)^C$$

where *A*, *B*, and *C* are empirically derived constants, *E* is effort in person-months, and *ev* is the estimation variable (either LOC or FP).

LOC-oriented estimation models:

$E = 5.2 \times (KLOC)^{0.91}$ **Walston-Felix model**

$E = 5.5 + 0.73 \times (KLOC)^{1.16}$ **Bailey-Basili model**

$E = 3.2 \times (KLOC)^{1.05}$ **Boehm simple model**

$E = 5.288 \times (KLOC)^{1.047}$ **Doty model for KLOC > 9**

FP-oriented models:

$E = -13.39 + 0.0545 * FP$ **Albrecht and Gaffney model**

$E = 60.62 \times 7.728 \times 10^{-8} * FP$ **Kemerer model**

$E = 585.7 + 15.12 * FP$ **Matson, Barnett, and Mellichamp model**

These models are part of algorithmic cost estimation techniques, helping software managers predict resources (effort, time, cost) early in a project lifecycle, though their accuracy depends on applying them to similar projects and environments.

COCOMO MODEL (COst COnstructive MOdel)-Barry Bohem

- Most widely used software estimation model. COCOMO predicts the efforts and schedule of a software product.
- COCOMO is defined in terms of three different models:
- – the Basic model (focuses on size)
- – the Intermediate model (size along with other parameter of projects, acts as homogenous entity)
- and – the Detailed model(organic+semi detached+embedded)
- The more complex models account for more factors that influence software projects, and make more accurate estimates.

Basic COCOMO MODEL

COCOMO II is actually a hierarchy of estimation models that address the following areas:

Application composition model.

Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.

Early design stage model.

Used once requirements have been stabilized and basic software architecture has been established.

Post-architecture-stage model.

Used during the construction of the software.

Organic: (Small teams, familiar problems, and nominal experience)

A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past

Semi Detached:(A mix of characteristics, lying between Organic and Embedded) The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached type.

Aspects	Organic	Semidetached	Embedded
Project Size	2 to 50 KLOC	50-300 KLOC	300 and above KLOC
Complexity	Low	Medium	High
Team Experience	Highly experienced	Some experienced as well as inexperienced staff	Mixed experience, includes experts
Environment	Flexible, fewer constraints	Somewhat flexible, moderate constraints	Highly rigorous, strict requirements
Effort Equation	$E = 2.4(400)^{1.05}$	$E = 3.0(400)^{1.12}$	$E = 3.6(400)^{1.20}$
Example	Simple payroll system	New system interfacing with existing systems	Flight control software

Embedded: (Highest complexity, requiring large teams, and high creativity and experience) A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

Basic CoCoMO

Power

Effort = a_1 $(KLOC)^{a2}$ PM

Time = b_1 $(Effort)^{b2}$ Months

where a_1 a_2 b_1 b_2 are constants

Power

Basic CoCoMO

Effort = a_1 (KLOC) ^{a2} PM

Time = b_1 (Effort) ^{b2} Months

where a_1 a_2 b_1 b_2 are constants

Effort	a_1	a_2
Organic	2.4	1.05
Semidetached	3.0	1.12
Embedded	3.6	1.20

Effort	b_1	b_2
Organic	2.5	0.38
Semidetached	2.5	0.35
Embedded	2.5	0.32

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

$$\text{Effort} = 2.4 * (32)^{1.05} \text{ P-M}$$

$$\text{Time} = 2.5 * (91)^{0.38} \text{ Months}$$

$$\text{Cost} = 14 * 15000 = 210000$$

$$\text{Person} = E/D$$

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 1 5000 = Rs. 208480.85

- People required = (Effort applied) / (development time)
  = 6.57 = 7 persons

Question 6 Consider a software project with 55000 lines of code Find Intermediate Cocomo Model effort estimation, Development Time and Person Month? Given EAF = 1.31

Solution

For Intermediate Cocomo Model

Effort = a $(KLOC)^b$ EAF $= 3(55)^{1.12}$ x(1.31) = 349.6 Person-month

Development Time D = c $(E)^d$ $= 2.5(349.6)^{0.35}$ = 19.417 months

Team Size = E/T = 349.6 / 19.417 = 18 People

The Software Equation

The "Software Equation" primarily refers to models like the **Putnam Software Equation (E = [LOC × B^(1/3) / P]^3 × 1/t^4)** used for estimating effort (E) in software projects, relating it to Lines of Code (LOC), productivity (P), and time (t).

It highlights that effort grows non-linearly with project size and can decrease dramatically with more time due to team dynamics.

The Software Equation

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

Where,

E = effort in person-months or person-years

t = project duration in months or years

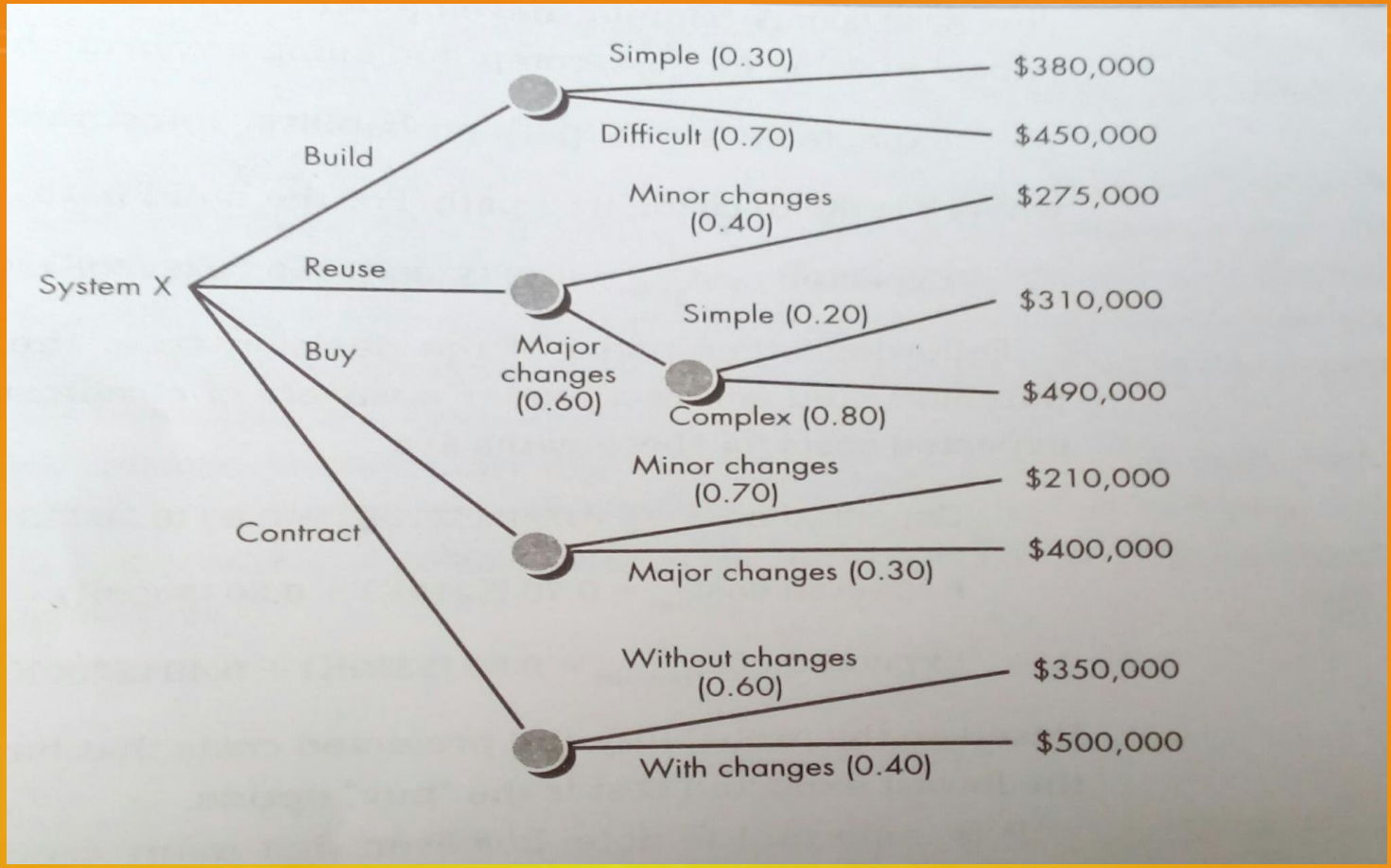
B = "special skills factor"(small project 0.16 – large project 0.39)

P = "productivity parameter" that reflects:

- Overall process maturity and management practices
- Extent to which good software engineering practices are used.
- The level of programming languages used
- The state of the software environment
- The skills and experience of the software team
- The complexity of the application

MAKE OR BUY DECUSION

the software engineering organization can
(1) build system *X from scratch*
(2) reuse existing "partial-experience" components to construct the System
(3) buy an available software product and modify it to meet local needs, or
(4) contract the software development to an outside vendor.


System X
Build
Reuse
Buy
Contract
Simple (0.30)
Difficult (0.70)
Minor changes (0.40)
Major changes (0.60)
Simple (0.20)
Complex (0.80)
Minor changes (0.70)
Major changes (0.30)
Without changes (0.60)
With changes (0.40)
$380,000
$450,000
$275,000
$310,000
$490,000
$210,000
$400,000
$350,000
$500,000

expected cost = Σ(path probability)*i x (estimated path cost)i*
where *i is the decision tree path.*

expected costbuild = 0.30 ($380K) + 0.70 ($450K) = $429K

Based on the probability and projected cost that have in noted the Lowest expected cost is the buy option.