Software Project Estimation & Scheduling

❖ 3.1 RESPONSIBILITY OF SOFTWARE PROJECT MANAGER

Job responsibility

- Software project managers take the overall responsibility of project to success.
- The job responsibility of a project manager ranges from invisible activities like building up team morale to highly visible customer presentations.
- Most managers take responsibility for project proposal writing, project cost estimation, scheduling, project staffing, deciding software process, project monitoring and control, software configuration management, risk management, interfacing with clients, and presentations.
- These activities can be classified into project planning, and project monitoring and control activities.
- **Project planning**
  - ✓ Project planning involves estimating characteristics of the project and then planning the project activities based on the estimates made.
  - ✓ The project planning activity is undertaken after the feasibility study phase.
  - ✓ The initial project plans that are made are revised from time to time as the project progresses and more project data become available.
- **Project monitoring and control activities**
  - ✓ The project monitoring and control activities are undertaken once the development activities start with the aim of ensuring that the development proceeds as per plan and changing the plan whenever required.

Skills necessary for software project management

- A theoretical knowledge of different project management techniques is certainly necessary to become a successful project manager.
- In addition to having a good understand of the latest software project management techniques such as cost estimation, risk management, configuration management, project managers need good communication skills.
- However, some skills such as tracking and controlling the progress of the project, customer interaction, managerial presentations, and team building are largely acquired through experience.

❖ 3.2 METRICS FOR SIZE ESTIMATION

- Estimation of the problem size is estimation of effort, time duration and cost of a software project.
- The size of a problem is obviously not the number of bytes that the source code occupies. It is neither the byte size of the executable code.

- Currently two metrics are popularly being used widely to estimate size:
  - ✓ Lines of code (LOC)
  - ✓ Function point (FP)

LINES OF CODE (LOC)

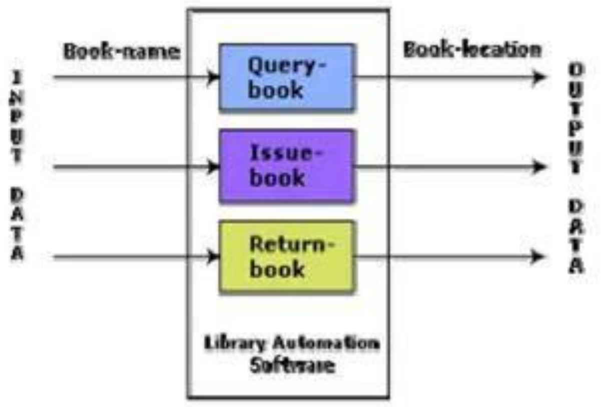
- LOC metric is very popular because it is the simplest to use. Using this metric, the project size is estimated by counting the number of source instructions in the developed program.
- Lines used for commenting the code and the header lines should be ignored.
- Determining the LOC count at the end of a project is a very simple job.
- To estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules and each module into sub modules and so on, until the sizes of the different leaf-level modules can be approximately predicted.

Shortcomings (or Disadvantages) of Lines of Code (LOC) metric

- LOC gives a numerical value of problem size that can vary with individual coding style – different programmers lay out their code in different ways. For example, one programmer might write several source instructions on a single line whereas another might split a single instruction across several lines. Of course, this problem can be easily overcome by counting the language tokens in the program rather than the lines of code.
- However, a more difficult problem arises because the length of a program depends on the choice of instructions used in writing the program. Therefore, even for the same problem, different programmers might come up with programs having different LOC counts.
- A good problem size measure should consider the overall complexity of the problem and the effort needed to solve it. That is, it should consider the effort needed to specify, design, code, test, etc. and not just the coding effort.
- LOC focuses on the coding activity alone; it only computes the number of source lines in final program.
- Larger code size does not necessarily imply better quality. Some programmers produce lengthy and complicated code as they do not make effective use of the available instruction set.
- If a programmer uses several library routines, then the LOC count will be lower. This would show up as smaller program size.
- It is very difficult to accurately estimate LOC in the final product from the problem specification. The LOC count can be accurately computed only after the code has been fully developed.
- Therefore, the LOC metric is little use to the project managers during project planning, since project planning is carried out even before any development activity has started. This is the biggest shortcoming of the LOC metric.

FUNCTION POINT (FP)

- Function Point metric overcomes many of the shortcomings of the LOC metric. One of the important advantages of using the function point metric is that it can be used to easily estimate the size of a software product directly from the problem specification.
- LOC metric, where the size can be accurately determined only after the product has fully been developed. The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different functions or features it supports.
- A software product supporting many features would certainly be of larger size than a product with less number of features. Each function when invoked reads some input data and transforms it to the corresponding output data.
- For example, the issue book feature (as shown in figure.) of a Library Automation Software takes the name of the book as input and displays its location and the number of copies available.



- The size of a product in function points (FP) can be expressed as the weighted sum of these five problem characteristics. The weights associated with the five characteristics were proposed empirically and validated by the observations over many projects. Function point is computed in two steps. The first step is to compute the unadjusted function point (UFP).

UFP = (Number of inputs)*4 + (Number of outputs)*5 + (Number of inquiries)*4 + (Number of files)*10 + (Number of interfaces)*10

Number of inputs

- Each data item input by the user is counted. It must be noted that individual data items input by the user are not considered in the calculation of the number of inputs, but a group of related inputs are considered as a single input.
- For example, while entering the data concerning employee to pay roll software; the data items name, age, address, phone number, etc. are together considered as a single input. All these data items can be considered to be related, since they count as to a single employee.

Number of outputs

- The outputs considered refer to reports printed, screen outputs, error messages produced, etc. While outputting the number of outputs the individual data items within a report are not considered, but a set of related data items is counted as one input.

Number of inquiries

- Inquiries are user commands such as print-account-balance. Inquiries are counted separately.

Number of files

- Each logical file is counted. A logical file means groups of logically related data.

Number of interfaces

- Here the interfaces considered are the interfaces used to exchange information with other systems. Examples of such interfaces are communication links with other systems etc.
- Once the unadjusted function point (UFP) is computed, the technical complexity factor (TCF) is computed next. TCF refines the UFP measure by considering 14 other factors such as high transaction rates, throughput, and response time requirements, etc.
- Each of these 14 factors is assigned from 0 (not present) to 6 (strong influence). The resulting numbers are summed, gives the total degree of influence (DI).
- Now, TCF = (0.65+0.01*DI). As DI can vary from 0 to 70, TCF can vary from 0.65 to 1.35.
- Finally, FP=UFP*TCF.

❖ 3.3 PROJECT ESTIMATION TECHNIQUES USING COCOMO MODEL

COCOMO

- COCOMO (Constructive Cost Estimation Model) was proposed by Boehm (1981). According to the Boehm any software development project can be classified into one of the following three categories based on the development complexity: organic, semidetached, and embedded.
- **Organic:** A development project can be considered of organic type, if the project deals with developing a well understood application program, the size of the development team is small, and the team members are experienced in developing similar types of projects.
- **Semidetached:** A development project can be considered of semidetached type, if the development consists of a mixture of experienced and inexperienced staff.
- **Embedded:** A development project is considered to be of embedded type, if the software being developed is strongly coupled to complex hardware.

- According to Boehm, software cost estimation should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

Basic COCOMO Model

- The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:
  Effort = a1 × (KLOC)$^{\mathbf{a2}}$ PM
  Tdev = b1 × (Effort)$^{\mathbf{b2}}$ Months
- Where
  - ✓ KLOC is the estimated size of the software product expressed in Kilo Lines of Code
  - ✓ a1, a2, b1, b2 are constants for each category of software products
  - ✓ Tdev is the estimated time to develop the software, expressed in months
  - ✓ Effort is the total effort required to develop the software product, expressed in person months (PMs)
- According to Boehm, every line of source text should be calculated LOC.
- The values of a1, a2, b1, b2 for different categories of products (i.e. organic, semidetached, and embedded) as given by Boehm [1981] are summarized below. He derived the above expressions by examining historical data collected from a large number of actual projects.
- **Estimation of development effort:** For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:
  - ✓ Organic : **Effort = 2.4(*KLOC*)$^{\mathbf{1.05}}$ PM**
  - ✓ Semi-detached : **Effort = 3.0(*KLOC*)$^{\mathbf{1.12}}$ PM**
  - ✓ Embedded : **Effort = 3.6(*KLOC*)$^{\mathbf{1.2}}$ PM**
- **Estimation of development time:** For the three classes of software products, the formulas for estimating the development time based on the effort are given below:
  - ✓ Organic : **Tdev = 2.5(*Effort*)$^{\mathbf{0.38}}$ Months**
  - ✓ Semi-detached : **Tdev = 2.5(*Effort*)$^{\mathbf{0.35}}$ Months**
  - ✓ Embedded : **Tdev = 2.5(*Effort*)$^{\mathbf{0.32}}$ Months**
- From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month.
- **Example:** Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.
- From the basic COCOMO estimation formula for organic
  software: Effort = **2.4 x (32)$^{\mathbf{1.05}}$** = **91 PM**
  Development time = **2.5 x (91)$^{\mathbf{0.38}}$** = **14 months**

Cost required to develop the product = **91 x 15000**

= **Rs. 1365000/-**

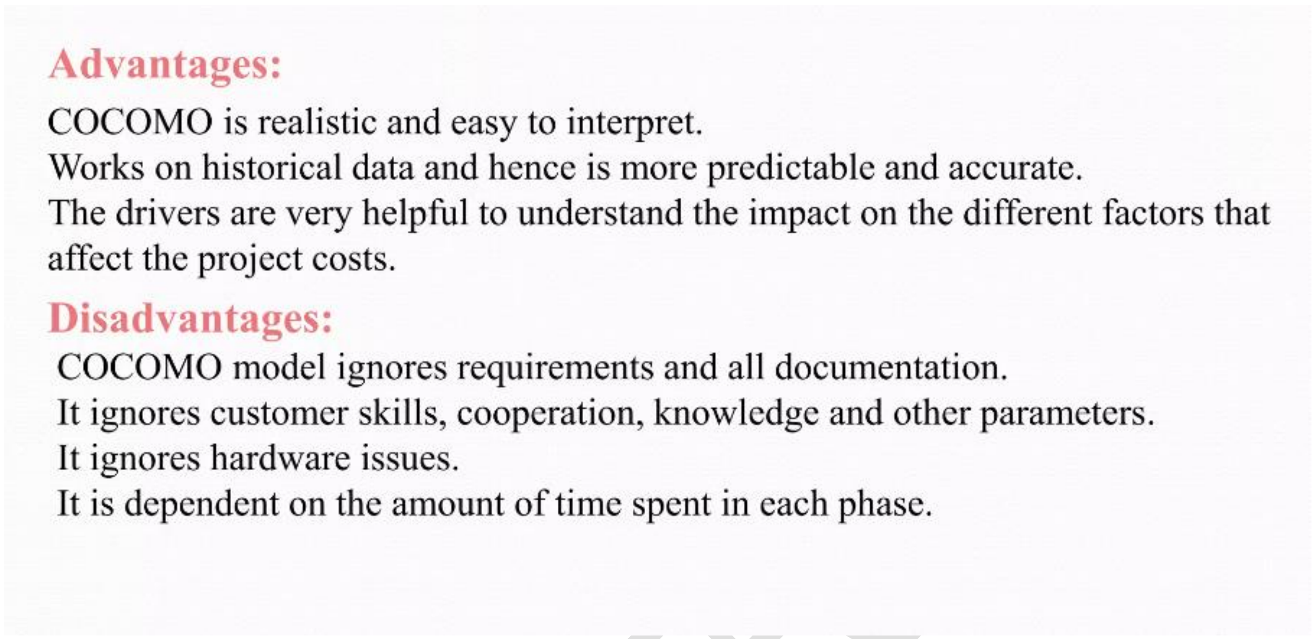
Intermediate COCOMO model

- Basic COCOMO model assumes that effort and development time are functions of the product size.
- Therefore, in order to obtain an accurate estimation of the effort and project duration, the effect of all relevant parameters must be taken into account.
- The intermediate COCOMO model recognizes this fact and refines the initial estimate obtained using the basic COCOMO expressions by using a set of 15 cost drivers (multipliers) based on various attributes of software development.
- For example, if modern programming practices are used, the initial estimates are scaled downward.
- If there are reliability requirements on the software product, this initial estimate is scaled upward.
- In general, the cost drivers can be classified as being attributes of the following items:
- **Product:** The characteristics of the product like reliability requirements of the product.
- **Computer:** Characteristics of the computer like execution speed required, storage space required etc.
- **Personnel:** experience level of personnel, programming capability, analysis capability, etc.
- **Development Environment:** use of the automation (CASE) tools used for software development.

Complete COCOMO model

- A major shortcoming of both the basic and intermediate COCOMO models is that they consider a software product as a single entity.
- However, most large systems are made up several smaller sub-systems. These subsystems may have widely different characteristics.
- For example, some subsystems may be considered as organic type, some semidetached, and some embedded. Not only that the development complexity of the subsystems may be different, but also for some subsystems the reliability requirements may be high, for some the development team might have no previous experience of similar development, and so on.
- The complete COCOMO model considers these differences in characteristics of the subsystems and estimates the effort and development time as the sum of the estimates for the individual subsystems.
- The following development project can be considered as an example of application of the complete COCOMO model.
- A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:
  - ✓ Database part
  - ✓ Graphical User Interface (GUI) part
  - ✓ Communication part
- Of these, the communication part can be considered as embedded software. The database part could be semi-detached software, and the GUI part organic software. The costs for these three

components can be estimated separately, and summed up to give the overall cost of the system.



Advantages:

COCOMO is realistic and easy to interpret.
Works on historical data and hence is more predictable and accurate.
The drivers are very helpful to understand the impact on the different factors that affect the project costs.

Disadvantages:

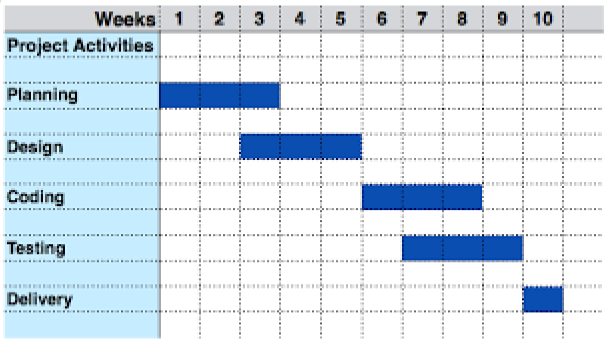
COCOMO model ignores requirements and all documentation.
It ignores customer skills, cooperation, knowledge and other parameters.
It ignores hardware issues.
It is dependent on the amount of time spent in each phase.

❖ 3.4 PROJECT SCHEDULING

i. Gantt Chart
ii. Flow Chart
iii. Sprint burn down chart for agile model

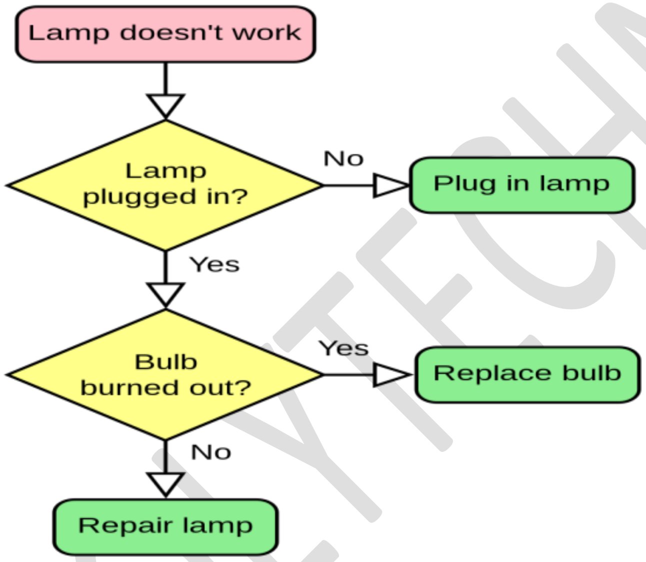
i. Gantt chart

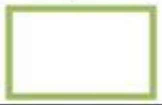
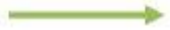
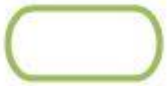
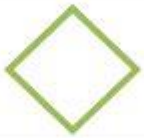
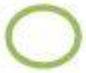
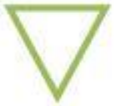
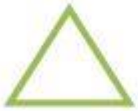
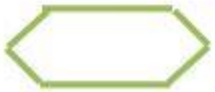
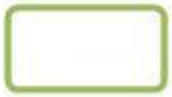
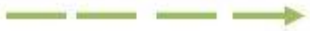
- Gantt charts are used to allocate resources to activities. The resources allocated to activities include staff, hardware, and software. Gantt charts are useful for resource planning.
- A Gantt chart is a special type of bar chart where each bar represents an activity. The bars are drawn along a time line. The length of each bar is proportional to the duration of time planned for the corresponding activity.
- This allows you to see: various activities, when each activity begins and ends, how long each activity is scheduled, start and end date of the whole project.
- A Gantt chart representation for the MIS problem is shown in the figure.

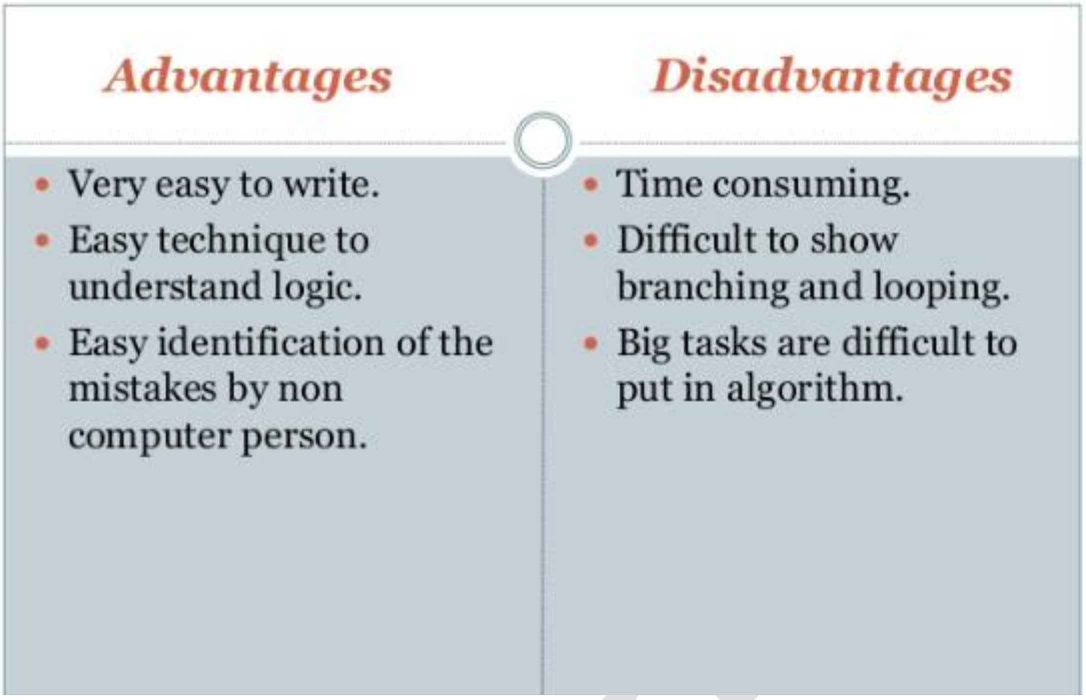


ii. Flow Chart

- A flowchart is a picture of the separate steps of a process in sequential order. It is a generic tool that can be adapted for a wide variety of purposes, and can be used to describe various processes, such as a manufacturing process, an administrative or service process, or a project plan.



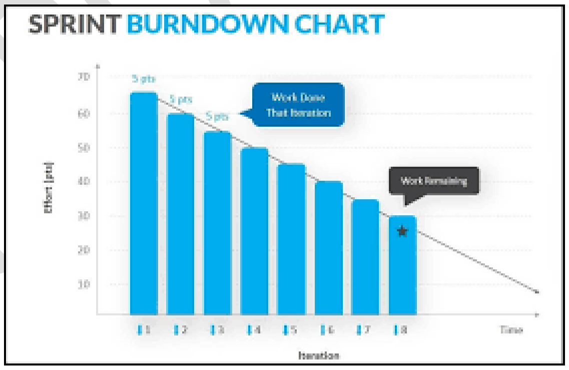
Name	Symbol	Description
Process		Process or action step
Flow line		Direction of process flow
Start/ terminator		Start or end point of process flow
Decision		Represents a decision making point
Connector		Inspection point
Inventory		Raw material storage
Inventory		Finished goods storage
Preparation		Initial setup and other preparation steps before start of process flow
Alternate process		Shows a flow which is an alternative to normal flow
Flow line(dashed)		Alternate flow direction of information flow



Advantages	Disadvantages
• Very easy to write. • Easy technique to understand logic. • Easy identification of the mistakes by non computer person.	• Time consuming. • Difficult to show branching and looping. • Big tasks are difficult to put in algorithm.

iii. Sprint burn down chart for agile model

- A sprint burn down chart is a visual comparison of how much work has been completed during a sprint and the total amount of work remaining. It helps measure a Scrum team's progress, and it provides an easy view of whether the team needs to make any adjustments to complete its work for the current sprint iteration.



❖ 3.5 Risk Management

I. Risk Identification
II. Risk Assessment
III. Risk Control

RISK IDENTIFICATION

- A software project can be affected by a large variety of risks. In order to be able to systematically identify the important risks which might affect a software project, it is necessary to categorize risks into different classes.

- The project manager can then examine which risks from each class are relevant to the project. There are three main categories of risks which can affect a software project:

Project risks

- Project risks concern varies forms of budgetary, schedule, personnel, resource, and customer-related problems. An important project risk is schedule. It is very difficult to monitor and control a software project.
- It is very difficult to control something which cannot be seen. For any manufacturing project, such as manufacturing of cars, the project manager can see the product taking shape. He can for instance, see that the engine is fitted, after that the doors are fitted, and the car is getting painted, etc. Thus he can easily assess the progress of the work and control it.
- The invisibility of the product being developed is an important reason why many software projects suffer from the risk of schedule.

Technical risks

- Technical risks concern design, implementation, interfacing, testing, and maintenance problems.
- Technical risks also include ambiguous specification, incomplete specification, changing specification, technical uncertainty. Most technical risks occur due to the development team's insufficient knowledge about the project.

Business risks

- This type of risks include risks of building an excellent product that no one wants, losing budgetary or personnel commitments, etc.

Example

- Let us consider the satellite based mobile communication product. The project manager can identify several risks in this project. We can classify them appropriately as:
  - ✓ What if the project cost extend to a large extent than what was estimated? – project risk
  - ✓ What if the mobile phones become too large for people to conveniently carry? – Business risk

RISK ASSESSMENT

- Risk assessment involves identifying risk, analyzing them and then assigns priority to them on the basis of the analysis.
- The objective of risk assessment is to rank the risks in terms of their damage. For risk assessment, first each risk should be rated in two ways:
  - ✓ The probability of a risk coming true (denoted as r).
  - ✓ The result of the problems associated with that risk (denoted as s).

- Based on these two factors, the priority of each risk can be computed:
  p = r * s
- Where, p is the priority with which the risk must be handled, r is the probability of the risk becoming true, and s is the result of damage caused due to the risk becoming true. If all identified risks are prioritized, then the most likely and damaging risks can be handled first and reject procedures can be designed for these risks.

RISK CONTROL

- After all the identified risks of a project are assessed, plans must be made to containment the most damaging and the most likely risks.
- Different risks require different containment procedures. In fact, most risks require expertness on the part of the project manager in handling the risk.
- There are three main strategies to plan for risk containment:
  - ✓ **Avoid the risk:** This may take several forms such as discussing with the customer to change the requirements to reduce the scope of the work.
  - ✓ **Transfer the risk:** This strategy involves getting the risky component developed by a third party.
  - ✓ **Risk reduction:** This involves planning ways to containment the damage due to a risk.
- To choose between the different strategies of handling a risk, the project manager must consider the cost of handling the risk and the corresponding reduction in risk.
- For this we may compute the risk leverage of the different risks. Risk leverage is the difference in risk divided by the cost of reducing the risk.

Risk leverage = (Risk before reducing - Risk after reducing) / cost of reducing