UNIT V

SOFTWARE PROJECT MANAGEMENT

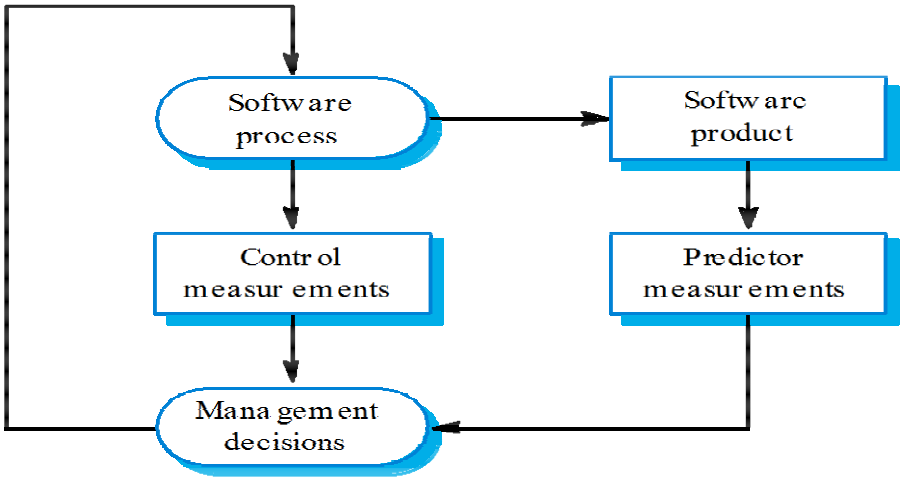
Measures and Measurements

- Software measurement is concerned with deriving a numeric value for an attribute of a software product or process.
- This allows for objective comparisons between techniques and processes.
- Although some companies have introduced measurement programmes, most organisations still don‘t make systematic use of software measurement.
- There are few established standards in this area.

Software metric

- Any type of measurement which relates to a software system, process or related documentation
    - Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allow the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.

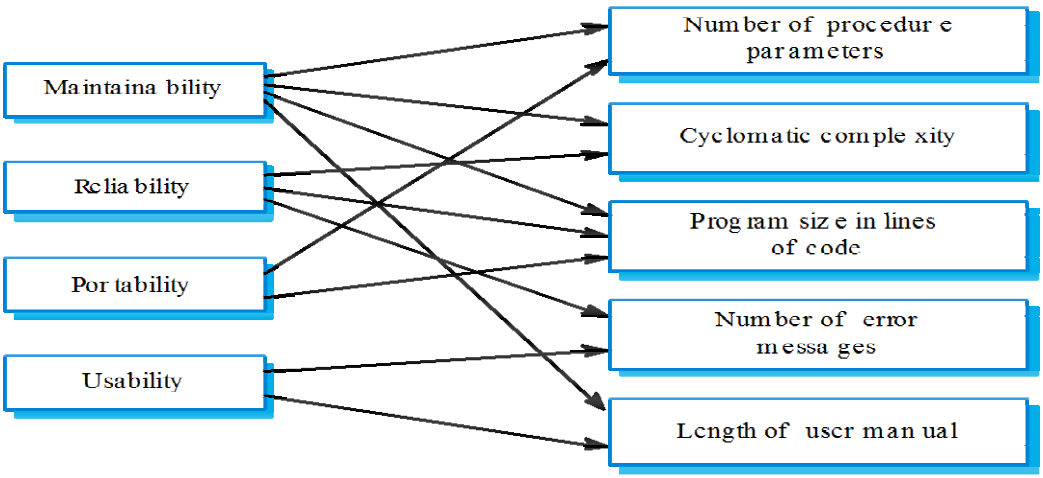
Predictor and control metrics



Metrics assumptions

- A software property can be measured.
- The relationship exists between what we can measure and what we want to know. We can only measure internal attributes but are often more interested in external software attributes.
- This relationship has been formalised and validated.
- It may be difficult to relate what can be measured to desirable external quality attributes.

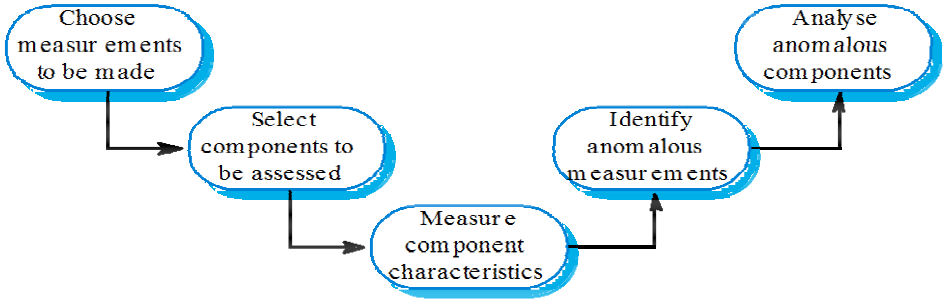
Internal and external attributes



The measurement process

- A software measurement process may be part of a quality control process.
- Data collected during this process should be maintained as an organisational resource.
- Once a measurement database has been established, comparisons across projects become possible.

Product measurement process



Data collection

- A metrics programme should be based on a set of product and process data.
- Data should be collected immediately (not in retrospect) and, if possible, automatically.
- Three types of automatic data collection
    - Static product analysis;
    - Dynamic product analysis;
    - Process data collation.

Data accuracy

- Don‘t collect unnecessary data
    - The questions to be answered should be decided in advance and the required data identified.
- Tell people why the data is being collected.
    - It should not be part of personnel evaluation.
- Don‘t rely on memory
    - Collect data when it is generated not after a project has finished.

Product metrics

- A quality metric should be a predictor of product quality.
- Classes of product metric
    - Dynamic metrics which are collected by measurements made of a program in execution;
    - Static metrics which are collected by measurements made of the system representations;
    - Dynamic metrics help assess efficiency and reliability; static metrics help assess complexity, understand ability and maintainability.

Dynamic and static metrics

- Dynamic metrics are closely related to software quality attributes
    - It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute).
- Static metrics have an indirect relationship with quality attributes
    - You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability.

Software product metrics

Software metric	Description
Fan in/Fan-out	Fan-in is a measure of the number of functions or methods that call some other function or method (say X). Fan-out is the number of functions that are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
Length of code	This is a measure of the size of a program. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.
Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability. I discuss how to compute cyclomatic complexity in Chapter 22.
Length of identifiers	This is a measure of the average length of distinct identifiers in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if statements are hard to understand and are potentially error-prone.
Fog index	This is a measure of the average length of words and sentences in documents. The higher the value for the Fog index, the more difficult the document is to understand.

Object-oriented metrics

Object-oriented metric	Description
Depth of inheritance tree	This represents the number of discrete levels in the inheritance tree where sub-classes inherit attributes and operations (methods) from super-classes. The deeper the inheritance tree, the more complex the design. Many different object classes may have to be understood to understand the object classes at the leaves of the tree.
Method fan-in/fan-out	This is directly related to fan-in and fan-out as described above and means essentially the same thing. However, it may be appropriate to make a distinction between calls from other methods within the object and calls from external methods.
Weighted methods per class	This is the number of methods that are included in a class weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1 and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be more difficult to understand. They may not be logically cohesive so cannot be reused effectively as super-classes in an inheritance tree.
Number of overriding operations	This is the number of operations in a super-class that are over-ridden in a sub-class. A high value for this metric indicates that the super-class used may not be an appropriate parent for the sub-class.

Measurement analysis

- It is not always obvious what data means
    - Analysing collected data is very difficult.
- Professional statisticians should be consulted if available.
- Data analysis must take local circumstances into account.

Measurement surprises

- Reducing the number of faults in a program leads to an increased number of help desk calls
    - The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total may increase;
    - A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls.

ZIPF's Law

- Zipf's Law as "the observation that frequency of occurrence of some event (*P*), as a function of the rank (*i*) when the rank is determined by the above frequency of occurrence, is a power-law function $P_i \sim 1/i^a$ with the exponent *a* close to unity (1)."

- Let *P* (a random variable) represented the frequency of occurrence of a keyword in a program listing.
- It applies to computer programs written in any modern computer language.
- Without empirical proof because it's an obvious finding, that any computer program written in any programming language has a power law distribution, i.e., some keywords are used more than others.
- Frequency of occurrence of events is inversely proportional to the rank in this frequency of occurrence.
- When both are plotted on a log scale, the graph is a straight line.
- we create entities that don't exist except in computer memory at run time; we create logic nodes that will never be tested because it's impossible to test every logic branch; we create information flows in quantities that are humanly impossible to analyze with a glance;
- Software application is the combination of keywords within the context of a solution and not their quantity used in a program; context is not a trivial task because the context of an application is attached to the problem being solved and every problem to solve is different and must have a specific program to solve it.
- Although a program could be syntactically correct, it doesn't mean that t he algorithms implemented solve the problem at hand. What's more, a correct program can solve the wrong problem. Let's say we have the simple requirement of printing "Hello, World!" A syntactically correct solution in Java looks as follows:
- Public class SayHello {

```
    public static void main(String[] args) {
        System.out.println("John Sena!");
    }
}
```

- This solution is obviously wrong because it doesn't solve the original requirement. This means that the context of the solution within the problem being solved needs to be determined to ensure its quality. In other words, we need to verify that the output matches the original requirement.
- Zip's Law can't even say too much about larger systems.

Software Cost Estimation

Software cost components

- Hardware and software costs.
- Travel and training costs.
- Effort costs (the dominant factor in most projects)
    - The salaries of engineers involved in the project;
    - Social and insurance costs.
- Effort costs must take overheads into account
    - Costs of building, heating, lighting.
    - Costs of networking and communications.
    - Costs of shared facilities (e.g library, staff restaurant, etc.).

Costing and pricing

- Estimates are made to discover the cost, to the developer, of producing a software system.

- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organisational, economic, political and business considerations influence the price charged.

Software productivity
- A measure of the rate at which individual engineers involved in software development produce software and associated documentation.
- Not quality-oriented although quality assurance is a factor in productivity assessment.
- Essentially, we want to measure useful functionality produced per time unit.

Productivity measures
- Size related measures based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- Function-related measures based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of measure.

Measurement problems
- Estimating the size of the measure (e.g. how many function points).
- Estimating the total number of programmer months that have elapsed.
- Estimating contractor productivity (e.g. documentation team) and incorporating this estimate in overall estimate.

Lines of code
- The measure was first proposed when programs were typed on cards with one line per card;
- How does this correspond to statements as in Java which can span several lines or where there can be several statements on one line.

Productivity comparisons
- The lower level the language, the more productive the programmer
    - The same functionality takes more code to implement in a lower-level language than in a high-level language.
- The more verbose the programmer, the higher the productivity
    - Measures of productivity based on lines of code suggest that programmers who write verbose code are more productive than programmers who write compact code.

Function Point model

Function points
- Based on a combination of program characteristics
    - external inputs and outputs;
    - user interactions;
    - external interfaces;
    - files used by the system.
- A weight is associated with each of these and the function point count is computed by multiplying each raw count by the weight and summing all values.

- The function point count is modified by complexity of the project
- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language
    - LOC = AVC * number of function points;
    - AVC is a language-dependent factor varying from 200-300 for assemble language to 2-40 for a 4GL;
- FPs are very subjective. They depend on the estimator
    - Automatic function-point counting is impossible.

COCOMO model

- An empirical model based on project experience.
- Well-documented, ‗independent' model which is not tied to a specific software vendor.
- Long history from initial version published in 1981 (COCOMO-81) through various instantiations to COCOMO 2.
- COCOMO 2 takes into account different approaches to software development, reuse, etc.

COCOMO 81

Project complexity	Formula	Description
Simple	$PM = 2.4\ (KDSI)^{1.05} \times M$	Well-understood applications developed by small teams.
Moderate	$PM = 3.0\ (KDSI)^{1.12} \times M$	More complex projects where team members may have limited experience of related systems.
Embedded	$PM = 3.6\ (KDSI)^{1.20} \times M$	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures.

COCOMO 2

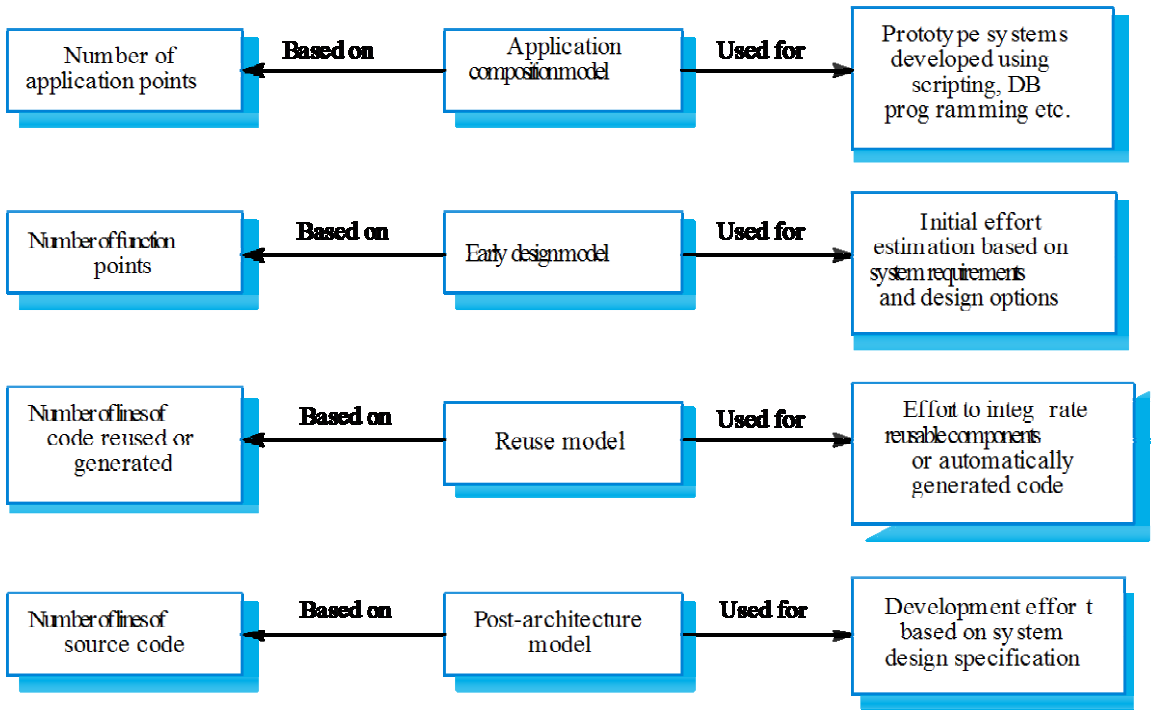
- COCOMO 81 was developed with the assumption that a waterfall process would be used and that all software would be developed from scratch.
- Since its formulation, there have been many changes in software engineering practice and COCOMO 2 is designed to accommodate different approaches to software development.

COCOMO 2 models

- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- The sub-models in COCOMO 2 are:
    - Application composition model. Used when software is composed from existing parts.
    - Early design model. Used when requirements are available but design has not yet started.
    - Reuse model. Used to compute the effort of integrating reusable components.

- Post-architecture model. Used once the system architecture has been designed and more information about the system is available.

Use of COCOMO 2 models



Application composition model

- Supports prototyping projects and projects where there is extensive reuse.
- Based on standard estimates of developer productivity in application (object) points/month.
- Takes CASE tool use into account.
- Formula is
  - PM = (NAP (1 - %reuse/100)) / PROD
  - PM is the effort in person-months, NAP is the number of application points and PROD is the productivity.

Early design model

- Estimates can be made after the requirements have been agreed.
- Based on a standard formula for algorithmic models
  - PM = A ´ $Size^B$ ´ M where
  - M = PERS ´ RCPX ´ RUSE ´ PDIF ´ PREX ´ FCIL ´ SCED;
  - A = 2.94 in initial calibration, Size in KLOC, B varies from 1.1 to 1.24 depending on novelty of the project, development flexibility, risk management approaches and the process maturity.

Multipliers

- Multipliers reflect the capability of the developers, the non-functional requirements, the familiarity with the development platform, etc.
  - RCPX - product reliability and complexity;

- RUSE - the reuse required;
- PDIF - platform difficulty;
- PREX - personnel experience;
- PERS - personnel capability;
- SCED - required schedule;
- FCIL - the team support facilities.

The reuse model

- Takes into account black-box code that is reused without change and code that has to be adapted to integrate it with new code.
- There are two versions:
  - Black-box reuse where code is not modified. An effort estimate (PM) is computed.
  - White-box reuse where code is modified. A size estimate equivalent to the number of lines of new source code is computed. This then adjusts the size estimate for new code.

Reuse model estimates

- For generated code:
  - PM = (ASLOC * AT/100)/ATPROD
  - ASLOC is the number of lines of generated code
  - AT is the percentage of code automatically generated.
  - ATPROD is the productivity of engineers in integrating this code.
- When code has to be understood and integrated:
  - ESLOC = ASLOC * (1-AT/100) * AAM.
  - ASLOC and AT as before.
  - AAM is the adaptation adjustment multiplier computed from the costs of changing the reused code, the costs of understanding how to integrate the code and the costs of reuse decision making.

Post-architecture level

- Uses the same formula as the early design model but with 17 rather than 7 associated multipliers.
- The code size is estimated as:
  - Number of lines of new code to be developed;
  - Estimate of equivalent number of lines of new code computed using the reuse model;
  - An estimate of the number of lines of code that have to be modified according to requirements changes.

The exponent term

- This depends on 5 scale factors (see next slide). Their sum/100 is added to 1.01
- A company takes on a project in a new domain. The client has not defined the process to be used and has not allowed time for risk analysis. The company has a CMM level 2 rating.
  - Precedenteness - new project (4)
  - Development flexibility - no client involvement - Very high (1)
  - Architecture/risk resolution - No risk analysis - V. Low .(5)
  - Team cohesion - new team - nominal (3)
  - Process maturity - some control - nominal (3)

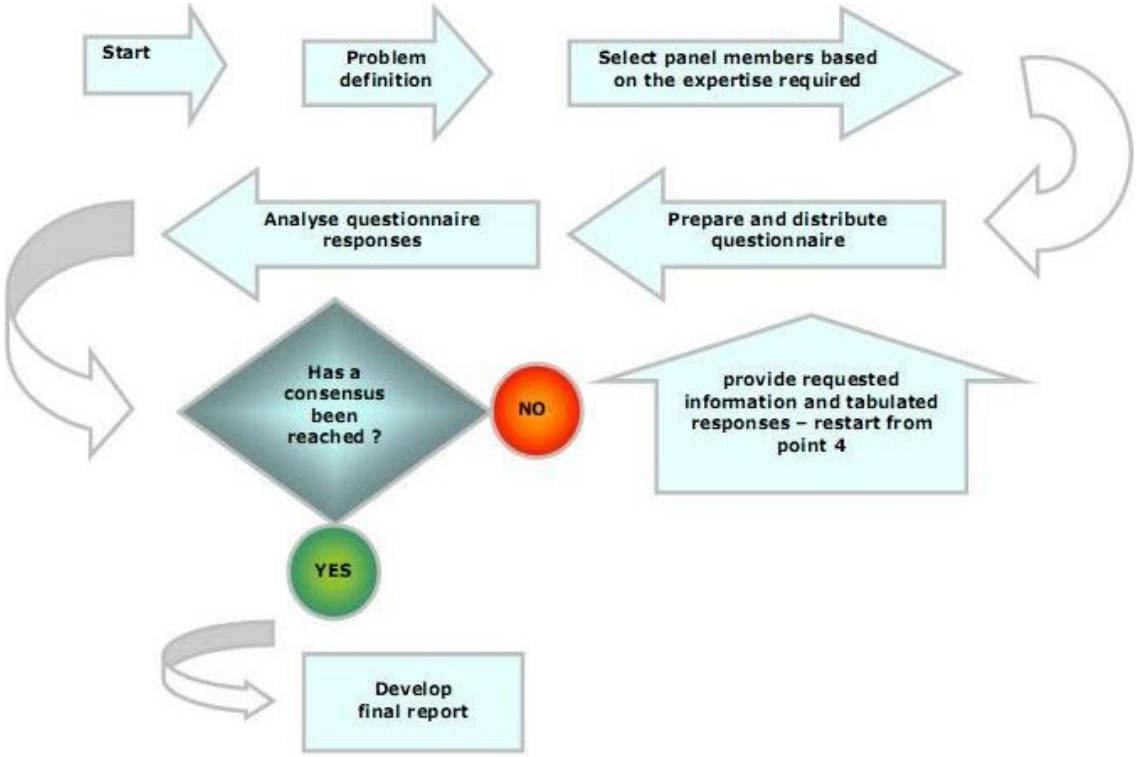
- Scale factor is therefore 1.17.

Multipliers

- Product attributes
  - Concerned with required characteristics of the software product being developed.
- Computer attributes
  - Constraints imposed on the software by the hardware platform.
- Personnel attributes
  - Multipliers that take the experience and capabilities of the people working on the project into account.
- Project attributes
  - Concerned with the particular characteristics of the software development project.

Delphi method

The Delphi method is a systematic, interactive forecasting method which relies on a panel of experts. The experts answer questionnaires in two or more rounds. After each round, a facilitator provides an anonymous summary of the experts‘ forecasts from the previous round as well as the reasons they provided for their judgments. Thus, experts are encouraged to revise their earlier answers in light of the replies of other members of their panel. It is believed that during this process the range of the answers will decrease and the group will converge towards the "correct" answer. Finally, the process is stopped after a pre-defined stop criterion (e.g. number of rounds, achievement of consensus, stability of results) and the mean or median scores of the final rounds determine the results.



The Delphi Technique is an essential project management technique that refers to an information gathering technique in which the opinions of those whose opinions are most valuable, traditionally industry experts, is solicited, with the ultimate hope and go al of attaining a consensus. Typically, the polling of these industry experts is done on an anonymous basis, in hopes of attaining opinions that are unfettered by fears or identifiability. The experts are presented with a series of questions in regards to the project, which is typically, but not always, presented to the expert by a third-party facilitator, in hopes of eliciting new ideas regarding specific project points. The responses from all experts are typically combined in the form of an overall summary, which is then provided to the experts for a review and for the opportunity to make further comments. This process typically results in consensus within a number of rounds, and this technique typically helps minimize bias, and minimizes the possibility t hat any one person can have too much influence on the outcomes.

Key characteristics

The following key characteristics of the Delphi method help the participants to focus on the issues at hand and separate Delphi from other methodologies:

- **Structuring of information flow**

The initial contributions from the experts are collected in the form of answers to questionnaires and their comments to these answers. The panel director controls the interactions among the participants by processing the information and filt ering out irrelevant content. This avoids the negative effects of face-to-face panel discussions and solves the usual problems of group dynamics.

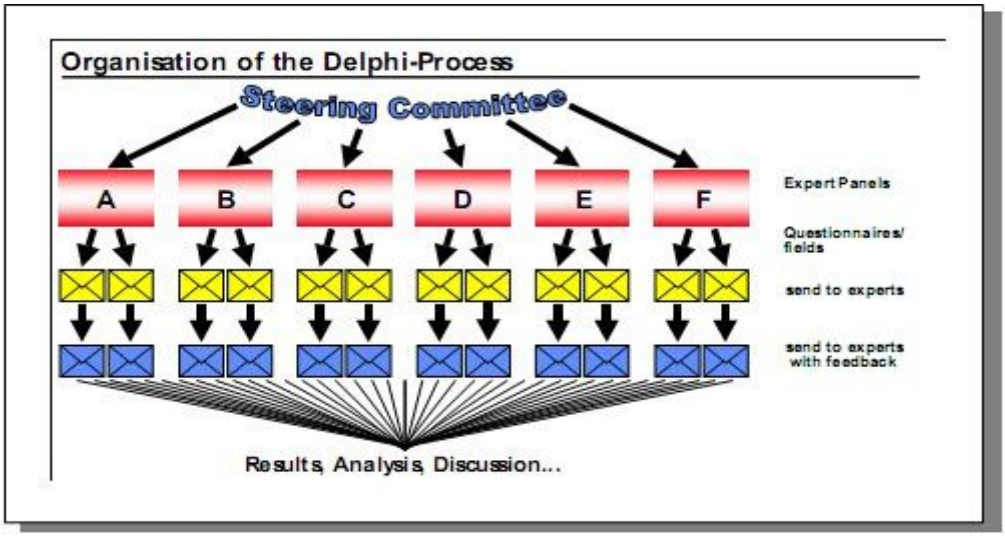
- **Regular feedback**

Participants comment on their own forecasts, the responses of others and on the progress of the panel as a whole. At any moment they can revise their earlier statements. While in regular group meetings participants tend to stick to previously stated opinions and often conform too much to group leader, the Delphi method prevents it.

- **Anonymity of the participants**

Usually all participants maintain anonymity. Their identity is not revealed even after the completion of the final report. This stops them from dominating others in the process using their authority or personality, frees them to some extent from their personal biases, minimizes the "bandwagon effect" or "halo effect", allows them to freely express their opinions, and encourages open critique and admitting errors by revising earlier judgments.

The first step is to found a steering committee (if you need one) and a management team with sufficient capacities for the process. Then expert panels to prepare and formulate the statements are helpful unless it is decided to let that be done by the management team. The whole procedure has to be fixed in advance: Do you need panel meetings or do the teams work virtually. Is the questionnaire an electronic or a paper one? This means, that logistics (from Internet programming to typing the results from the paper versions) have to be organised. Will there be follow-up work-shops,interviews, presentations? If yes, these also have to be organised and pre-pared. Printing of brochures, leaflets, questionnaire, reports have also be considered. The last organisational point is the interface with the financing organisation if this is different from the management team.

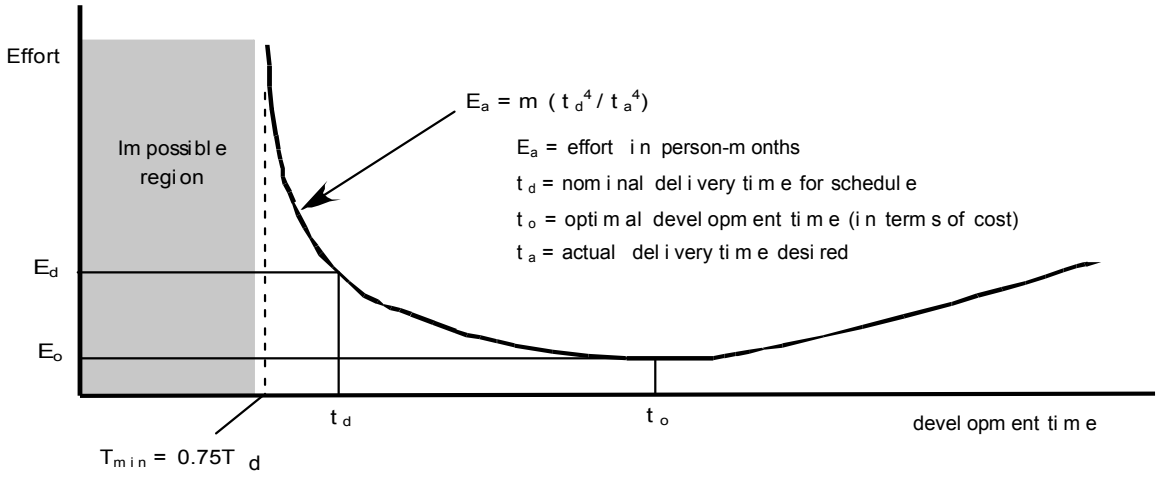


Scheduling

Scheduling Principles

- compartmentalization—define distinct tasks
- interdependency—indicate task interrelationship
- effort validation—be sure resources are available
- defined responsibilities—people must be assigned
- defined outcomes—each task must have an output
- defined milestones—review for quality

Effort and Delivery Time



Empirical Relationship: P vs E

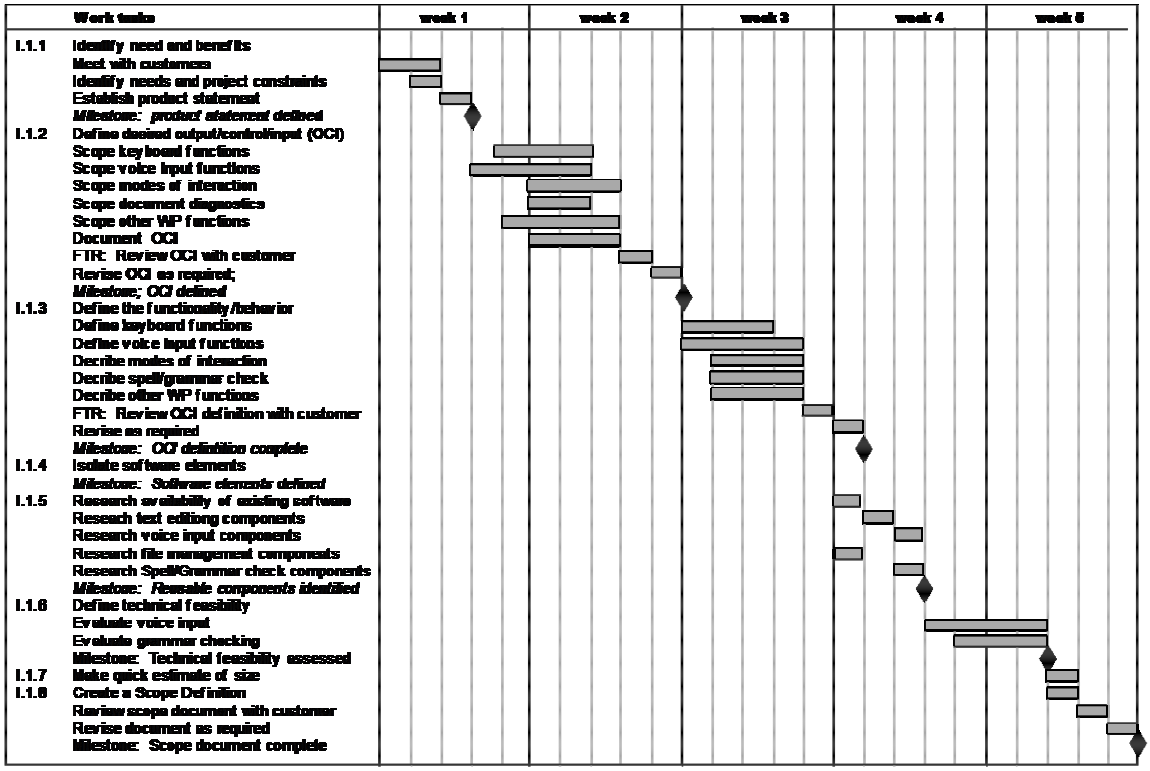
Given Putnam's Software Equation (5-3),

$$E = L^3 / (P^3 t^4)$$

Consider a project estimated at 33 KLOC, 12 person-years of effort, with a P of 10K, the completion time would be 1.3 years
If deadline can be extended to 1.75 years,

$E = L^3 / (P^3 t^4) \approx 3.8$ p-years vs 12 p-years

Timeline Charts



Effort Allocation

- –front endǁ activities
  - customer communication
  - analysis
  - design
  - review and modification
- construction activities
  - coding or code generation
- testing and installation
  - unit, integration
  - white-box, black box
  - regression

Defining Task Sets

- determine type of project
  - concept development, new application development, application enhancement, application maintenance, and reengineering projects

- assess the degree of rigor required
- identify adaptation criteria
- select appropriate software engineering tasks

Earned Value Analysis

- Earned value
  - is a measure of progress
  - enables us to assess the "percent of completeness" of a project using quantitative analysis rather than rely on a gut feeling
  - "provides accurate and reliable readings of performance from as early as 15 percent into the project."

Computing Earned Value

Budgeted cost of work scheduled (BCWS)

- The *budgeted cost of work scheduled* (BCWS) is determined for each work task represented in the schedule.
  - $BCWS_i$ is the effort planned for work task *i.*
  - To determine progress at a given point along the project schedule, the value of BCWS is the sum of the $BCWS_i$ values for all work tasks that should have been completed by that point in time on the project schedule.
- The BCWS values for all work tasks are summed to derive the *budget at completion,* BAC. Hence,
- $BAC = \sum (BCWS_k)$ for all tasks *k*

Budgeted cost of work performed (BCWP)

- Next, the value for *budgeted cost of work performed* (BCWP) is computed.
  - The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule.
- "the distinction between the BCWS and the BCWP is that the former represents the budget of the activities that were planned to be completed and the latter represents the budget of the activities that actually were completed."
- Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:
  - Schedule performance index, SPI = BCWP/BCWS
  - Schedule variance, SV = BCWP – BCWS
  - SPI is an indication of the efficiency with which the project is utilizing scheduled resources.

Actual cost of work performed, ACWP

- Percent scheduled for completion = BCWS/BAC
  - provides an indication of the percentage of work that should have been completed by time *t.*
- Percent complete = BCWP/BAC
  - provides a quantitative indication of the percent of completeness of the project at a given point in time, *t.*

- *Actual cost of work performed,* ACWP, is the sum of the effort actually expended on work tasks that have been completed by a point in time on the project schedule. It is then possible to compute
    - Cost performance index, CPI = BCWP/ACWP
    - Cost variance, CV = BCWP – ACWP

Problem

- Assume you are a software project manager and that you've been asked to computer earned value statistics for a small software project. The project has 56 planned work tasks that are estimated to require 582 person-days to complete. At the time that you've been asked to do the earned value analysis, 12 tasks have been completed. However, the project schedu le indicates that 15 tasks should have been completed. The following scheduling data (in person-days) are available:

Task	Planned Effort	Actual Effort
1	12	12.5
2	15	11
3	13	17
4	8	9.5
5	9.5	9.0
6	18	19
7	10	10
8	4	4.5
9	12	10
10	6	6.5
11	5	4
12	14	14.5
13	16	
14	6	
15	8	

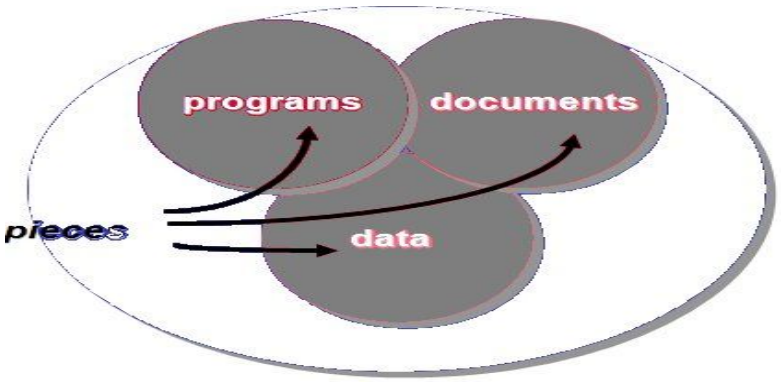
Error Tracking

- Schedule Tracking
    - conduct periodic project status meetings in which each team member reports progress and problems.
    - evaluate the results of all reviews conducted throughout the software engineering process.
    - determine whether formal project milestones (diamonds in previous slide) have been accomplished by the scheduled date.
    - compare actual start-date to planned start-date for each project task listed in the resource table
    - meet informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon.
    - use earned value analysis to assess progress quantitatively.
- Progress on an OO Project-I

- Technical milestone: OO analysis completed
  - All classes and the class hierarchy have been defined and reviewed.
  - Class attributes and operations associated with a class have been defined and reviewed.
  - Class relationships (Chapter 8) have been established and reviewed.
  - A behavioral model (Chapter 8) has been created and reviewed.
  - Reusable classes have been noted.
- Technical milestone: OO design completed
  - The set of subsystems (Chapter 9) has been defined and reviewed.
  - Classes are allocated to subsystems and reviewed.
  - Task allocation has been established and reviewed.
  - Responsibilities and collaborations (Chapter 9) have been identified.
  - Attributes and operations have been designed and reviewed.
  - The communication model has been created and reviewed.
- Progress on an OO Project-II
- Technical milestone: OO programming completed
  - Each new class has been implemented in code from the design model.
  - Extracted classes (from a reuse library) have been implemented.
  - Prototype or increment has been built.
- Technical milestone: OO testing
  - The correctness and completeness of OO analysis and design models has been reviewed.
  - A class-responsibility-collaboration network (Chapter 8) has been developed and reviewed.
  - Test cases are designed and class-level tests (Chapter 14) have been conducted for each class.
  - Test cases are designed and cluster testing (Chapter 14) is completed and the classes are integrated.
  - System level tests have been completed.

Software Configuration Management

- Configuration management is all about change control.
- Every software engineer has to be concerned with how changes made to work products are tracked and propagated throughout a project.
- To ensure quality is maintained the change process must be audited.



Software Configuration categories

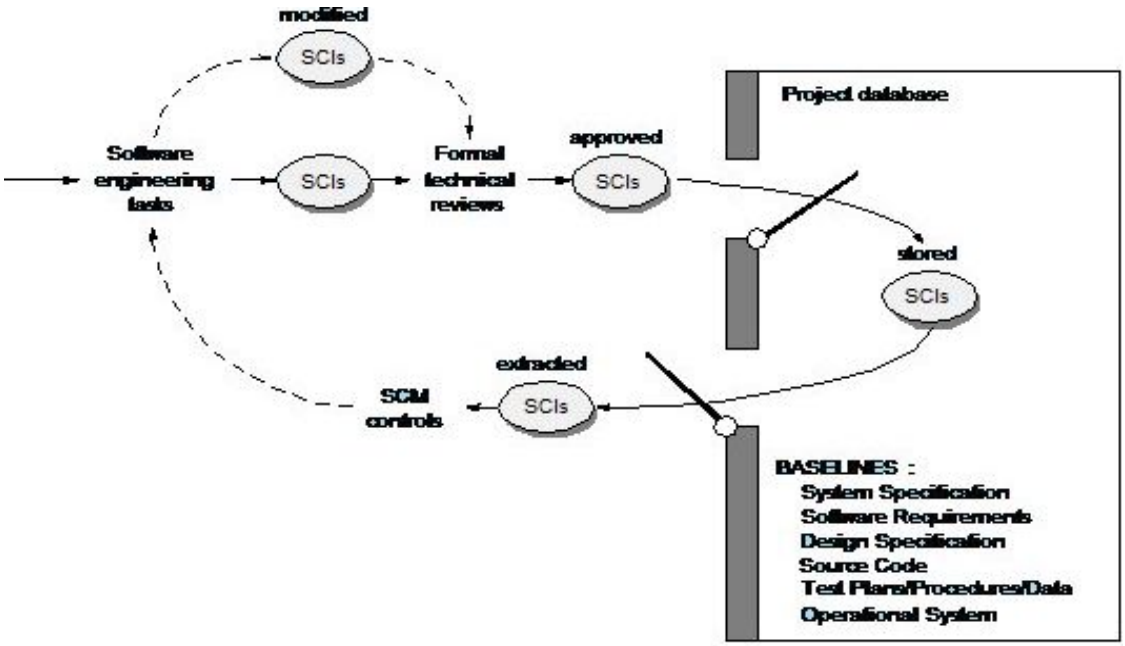
- Computer programs
  - source
  - executable
- Documentation
  - Technical / user
- Data
  - contained within the program
  - external data (e.g. files and databases)

Elements of SCM

- Component element
  - Tools coupled with file management
- Process element
  - Procedures define change management
- Construction element
  - Automate construction of software
- Human elements
  - Give guidance for activities and process features

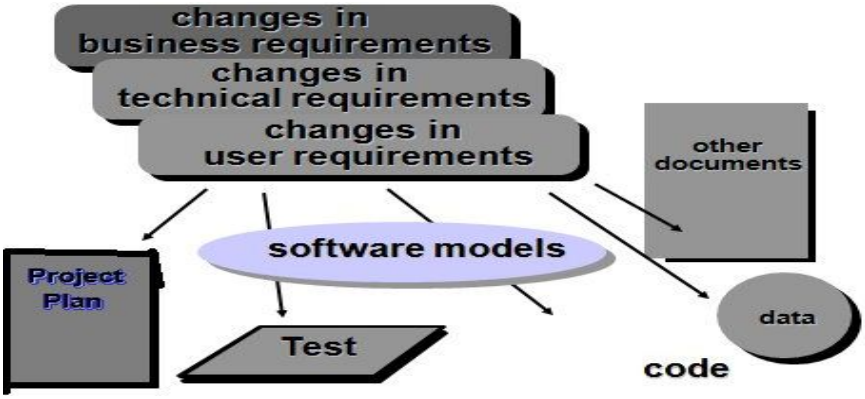
Baselines

- A work product becomes a baseline only after it is reviewed and approved.
- Before baseline – changes informal
- Once a baseline is established each change request must be evaluated and verified before it is processed.



Software Configuration Items

- SCI
- Document
- Test cases
- Program component
- Editors, compilers, browsers
  - Used to produce documentation.



Configuration Management process

- Identification
  - tracking changes to multiple SCI versions
- Version control
  - controlling changes before and after customer release
- Change control
  - authority to approve and prioritize changes
- Configuration auditing
  - ensure changes are made properly
- Reporting
  - tell others about changes made

Program evolution dynamics

- Program evolution dynamics is the study of the processes of system change.
- After major empirical studies, Lehman and Belady proposed that there were a number of ‗laws' which applied to all systems as they evolved.
- There are sensible observations rather than laws. They are applicable to large systems developed by large organisations. Perhaps less applicable in other cases.

Importance of evolution

- Organizations have huge investments in their software systems - they are critical business assets.
- To maintain the value of these assets to the business, they must be changed and updated.
- The majority of the software budget in large companies is devoted to evolving existing software rather than developing new software.

Software change

- Software change is inevitable
    - New requirements emerge when the software is used;
    - The business environment changes;
    - Errors must be repaired;
    - New computers and equipment is added to the system;
    - The performance or reliability of the system may have to be improved.
- A key problem for organisations is implementing and managing change to their existing software systems.

Lehman's laws

Law	Description
Continuing change	A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment.
Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases and the number of reported errors is approximately invariant for each system release.
Organisational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
Declining quality	The quality of systems will appear to be declining unless they are adapted to changes in their operational environment.
Feedback system	Evolution processes incorporate multi-agent, multi-loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

Applicability of Lehman's laws

- Lehman's laws seem to be generally applicable to large, tailored systems developed by large organisations.
    - Confirmed in more recent work by Lehman on the FEAST project (see further reading on book website).

- It is not clear how they should be modified for
    - Shrink-wrapped software products;
    - Systems that incorporate a significant number of COTS components;

- Small organisations;
- Medium sized systems.

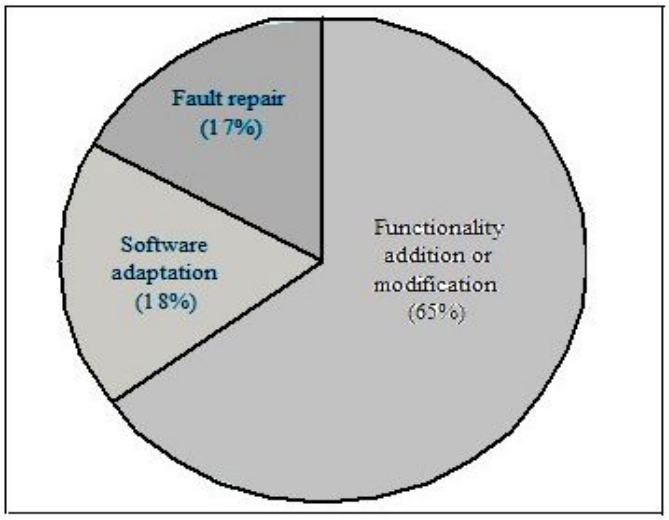
Software maintenance

- Modifying a program after it has been put into use or delivered.
- Maintenance does not normally involve major changes to the system‘s architecture.
- Changes are implemented by modifying existing components and adding new components to the system.
- Maintenance is inevitable
- The system requirements are likely to change while the system is being developed because the environment is changing. Therefore a delivered system won't meet its requirements!
- Systems are tightly coupled with their environment. When a system is installed in an environment it changes that environment and therefore changes the system requirements.
- Systems MUST be maintained therefore if they are to remain useful in an environment.

Types of maintenance

- Maintenance to repair software faults
  - Code ,design and requirement errors
  - Code & design cheap. Requirements most expensive.
- Maintenance to adapt software to a different operating environment
  - Changing a system‘s hardware and other support so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- Maintenance to add to or modify the system‘s functionality
  - Modifying the system to satisfy new requirements for org or business change.

Distribution of maintenance effort

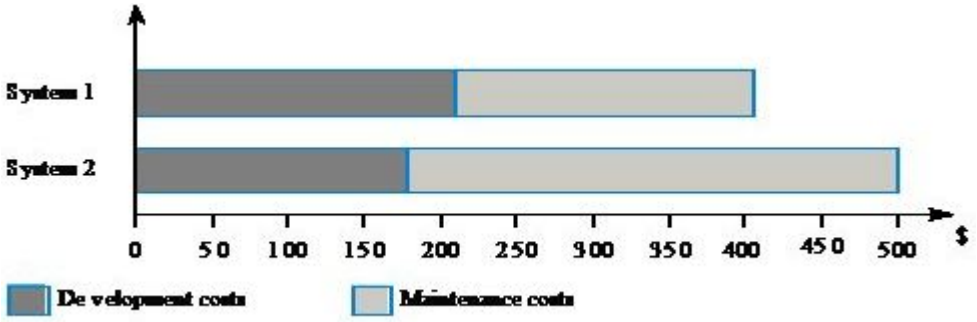


Maintenance costs

- Usually greater than development costs (2* to 100* depending on the application).

- Affected by both technical and non-technical factors.
- Increases as software is maintained. Maintenance corrupts the software structure so makes further maintenance more difficult.
- Ageing software can have high support costs (e.g. old languages, compilers etc.).

Development/maintenance costs



Maintenance cost factors

- Team stability
  - Maintenance costs are reduced if the same staff are involved with them for some time.
- Contractual responsibility
  - The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change.
- Staff skills
  - Maintenance staff are often inexperienced and have limited domain knowledge.
- Program age and structure
  - As programs age, their structure is degraded and they become harder to understand and change.

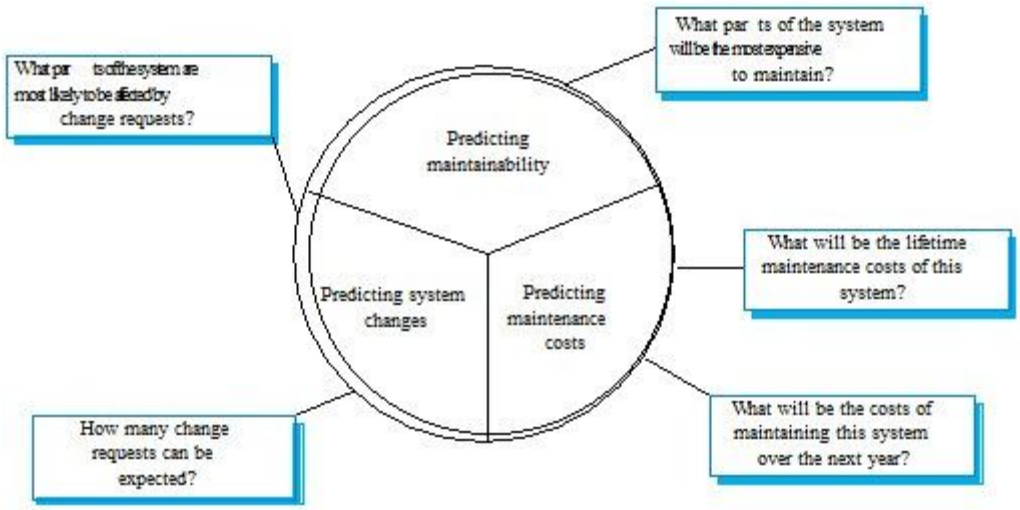
Maintenance prediction

- Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs
  - Change acceptance depends on the maintainability of the components affected by the change;
  - Implementing changes degrades the system structure and reduces its maintainability;
  - Maintenance costs depend on the number of changes and costs of change depend on maintainability.

Change prediction

- Predicting the number of changes requires and understanding of the relationships between a system and its environment.
- Tightly coupled systems require changes whenever the environment is changed.

- Factors influencing this relationship are
    - Number and complexity of system interfaces;
    - Number of inherently volatile system requirements;
    - The business processes where the system is used.



Complexity metrics

- Predictions of maintainability can be made by assessing the complexity of system components.
- Studies have shown that most maintenance effort is spent on a relatively small number of system components of complex system.
- Reduce maintenance cost – replace complex components with simple alternatives.
- Complexity depends on
    - Complexity of control structures;
    - Complexity of data structures;
    - Object, method (procedure) and module size.

Process metrics

- Process measurements may be used to assess maintainability
    - Number of requests for corrective maintenance;
    - Average time required for impact analysis;
    - Average time taken to implement a change request;
    - Number of outstanding change requests.
- If any or all of these is increasing, this may indicate a decline in maintainability.
- COCOMO2 model maintenance = understand existing code + develop new code.

Project management

Objectives

- To explain the main tasks undertaken by project managers
- To introduce software project management and to describe its distinctive characteristics
- To discuss project planning and the planning process

- To show how graphical schedule representations are used by project management
- To discuss the notion of risks and the risk management process Software project management
- Concerned with activities involved in ensuring that software is delivered on time and on schedule and in accordance with the requirements of the organisations develoing and procuring the software.
- Project management is needed because software development is always subject to budget and schedule constraints that are set by the organisation developing the software.

Project planning

- Probably the most time-consuming project management activity.
- Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
- Various different types of plan may be developed to support the main software project plan that is concerned with schedule and budget.

Types of project plan

Plan	Description
Quality plan	Describes the quality procedures and standards that will be used in a project.
Validation plan	Describes the approach, resources and schedule used for system validation.
Configuration management Plan	Describes the configuration management procedures and structures to be used.
Maintenance plan	Predicts the maintenance requirements of the system, maintenance costs and effort required.
Development plan.	Describes how the skills and experience of the project team members will be developed.

Project planning process

Establish the project constraints(delivery date, staff, budget)
Make initial assessments of the project parameters (structure, size)
Define project milestones and deliverables
while project has not been completed or cancelled loop
    Draw up project schedule
    Initiate activities according to schedule
    Wait (for a while)
    Review project progress
    Revise estimates of project parameters
    Update the project schedule
    Re-negotiate project constraints and deliverables
    if (problems arise) then
        Initiate technical review and possible revision
    end if
end loop

project plan

The project plan sets out:

- resources available to the project
- work breakdown
- schedule for the work.

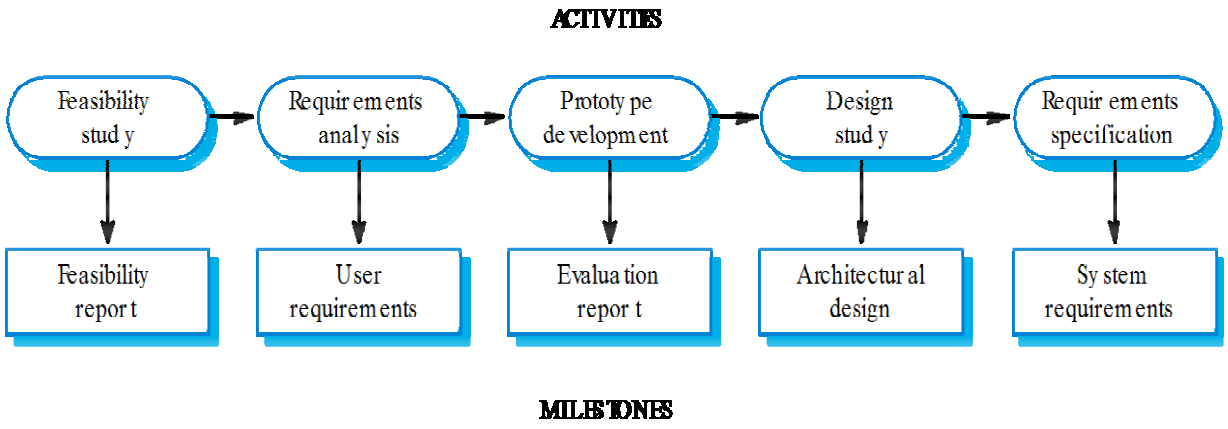
Project plan structure

- Introduction – objective, budget, time
- Project organisation. – roles of people
- Risk analysis. – arising, reduction
- Hardware and software resource requirements.
- Work breakdown. – break project to activity, milestone
- Project schedule. – time, allocation of people
- Monitoring and reporting mechanisms.

Milestones and deliverables

- Milestones are the end-point of a process activity.- report presented to management
- Deliverables are project results delivered to customers.
  - milestones need not be deliverables. May be used by project managers. – not to customers
- The waterfall process allows for the straight forward definition of progress milestones.

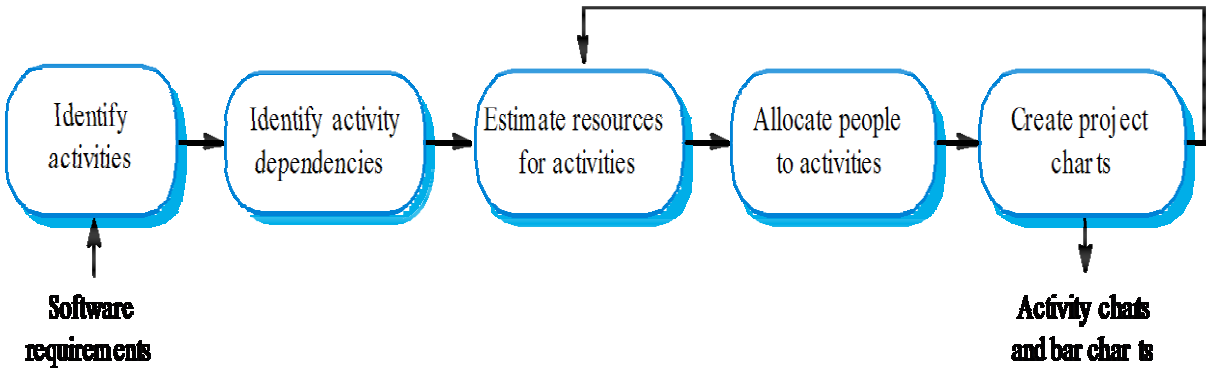
Milestones in requirement process



Project scheduling

- Split project into tasks and estimate time and resources required to complete each task.
- Organize tasks concurrently to make optimal use of workforce.
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
- Dependent on project managers intuition and experience.

The project scheduling process



Scheduling problems

- Estimating the difficulty of problems and hence the cost of developing a solution is hard.
- Productivity is not proportional to the number of people working on a task.
- Adding people to a late project makes it later because of communication overheads.
- The unexpected always happens. Always allow contingency in planning.

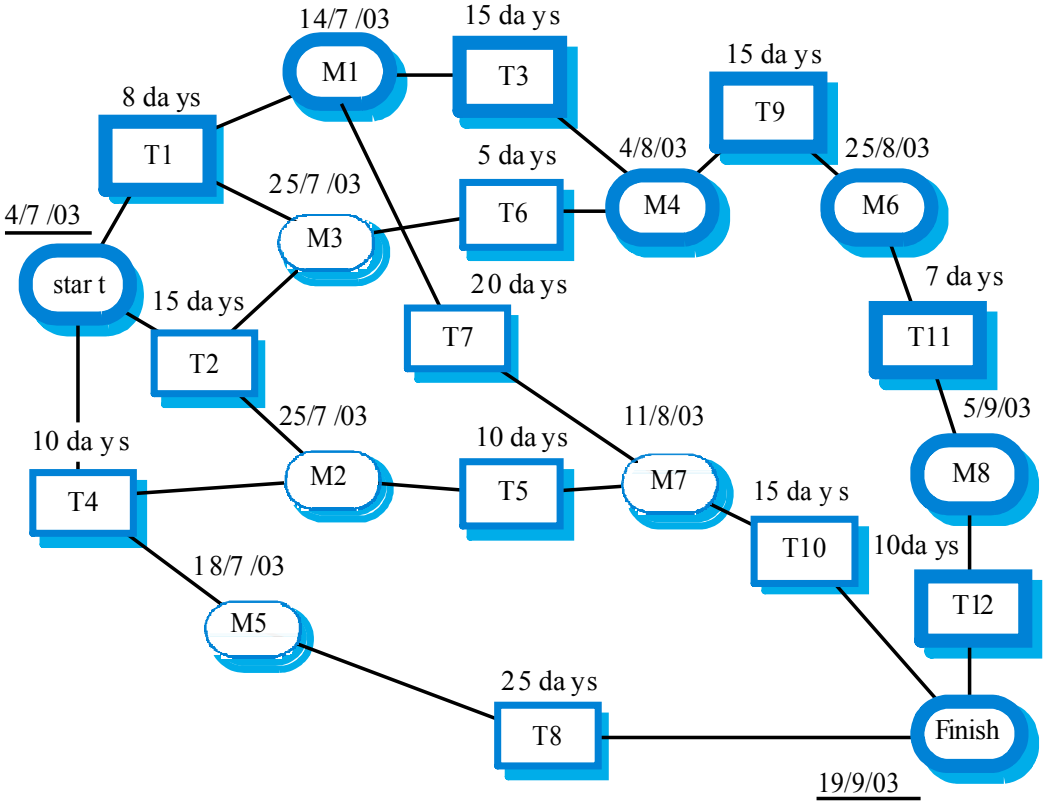
Bar charts and activity networks

- Graphical notations used to illustrate the project schedule.
- Show project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
- Activity charts show task dependencies and the critical path.
- Bar charts show schedule against calendar time.

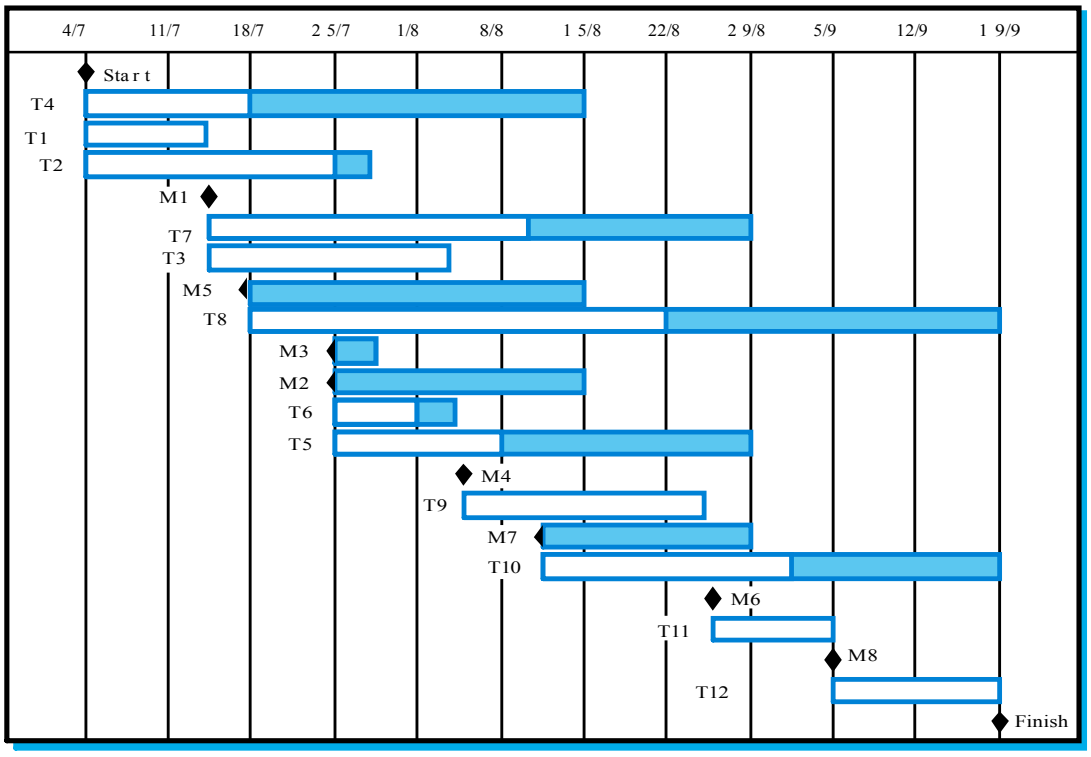
Task durations and dependencies

Activity	Duration (days)	Dependencies
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

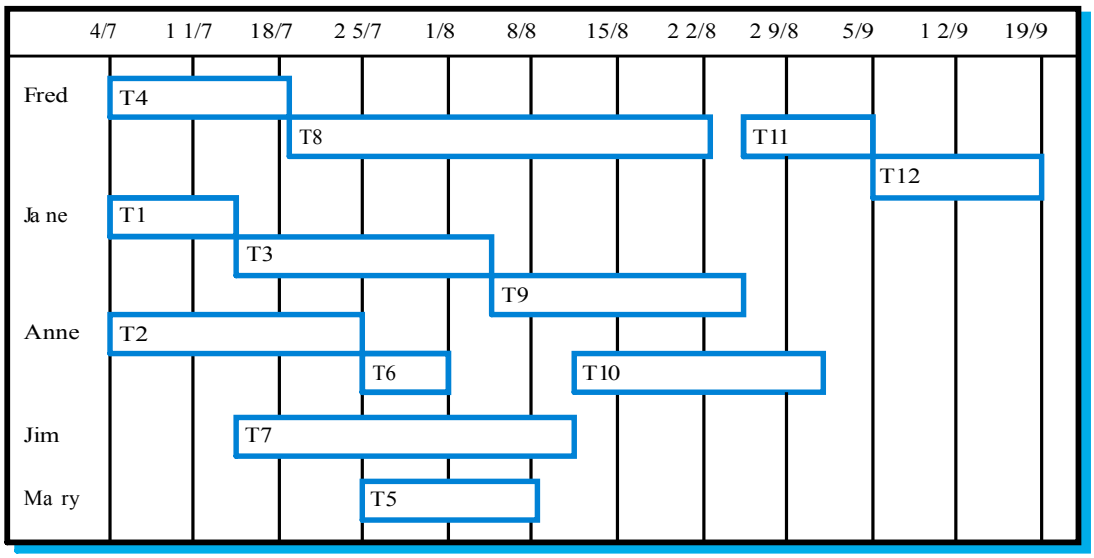
Activity network



Activity timeline



Staff allocation



Risk management

- Risk management - identifying risks and drawing up plans to minimise their effect on a project.
- A risk is a probability that some adverse circumstance will occur
    - Project risks : affect schedule or resources. eg: loss of experienced designer.
    - Product risks: affect the quality or performance of the software being developed. eg: failure of purchased component.
    - Business risks : affect organisation developing software. Eg: competitor introducing new product.

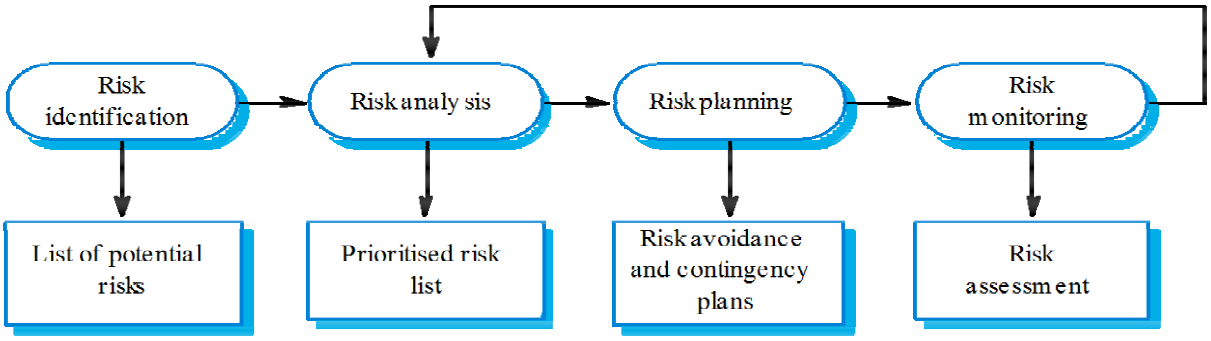
Software risks

Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organisational management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool under-performance	Product	CASE tools which support the project do not perform as anticipated
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

Risk management process

- Risk identification
  - Identify project, product and business risks;
- Risk analysis
  - Assess the likelihood and consequences of these risks;
- Risk planning
  - Draw up plans to avoid or minimise the effects of the risk;
- Risk monitoring
  - Constantly monitor risks & plans for risk mitigation.

Risk management process



Risk identification

- Discovering possible risk
- Technology risks.
- People risks.
- Organisational risks.
- Tool risk.
- Requirements risks.
- Estimation risks.

Risks and risk types

Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. Software components that should be reused contain defects that limit their functionality.
People	It is impossible to recruit staff with the skills required. Key staff are ill and unavailable at critical times. Required training for staff is not available.
Organisational	The organisation is restructured so that different management are responsible for the project. Organisational financial problems force reductions in the project budget.
Tools	The code generated by CASE tools is inefficient. CASE tools cannot be integrated.

Requirements	Changes to requirements that require major design rework are proposed. Customers fail to understand the impact of requirements changes.
Estimation	The time required to develop the software is underestimated. The rate of defect repair is underestimated. The size of the software is underestimated.

Risk analysis

- Make judgement about probability and seriousness of each identified risk.
- Made by experienced project managers
- Probability may be very low(<10%), low(10-25%), moderate(25-50%), high(50-75%) or very high(>75%). not precise value. Only range.
- Risk effects might be catastrophic, serious, tolerable or insignificant.

Risk	Probability	Effects
Organisational financial problems force reductions in the project budget.	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project.	High	Catastrophic
Key staff are ill at critical times in the project.	Moderate	Serious
Software components that should be reused contain defects which limit their functionality.	Moderate	Serious
Changes to requirements that require major design rework are proposed.	Moderate	Serious
The organisation is restructured so that different management are responsible for the project.	High	Serious
The database used in the system cannot process as many transactions per second as expected.	Moderate	Serious
The time required to develop the software is underestimated.	High	Serious
CASE tools cannot be integrated.	High	Tolerable
Customers fail to understand the impact of requirements changes.	Moderate	Tolerable
Required training for staff is not available.	Moderate	Tolerable
The rate of defect repair is underestimated.	Moderate	Tolerable
The size of the software is underestimated.	High	Tolerable
The code generated by CASE tools is inefficient.	Moderate	Insignificant

Risk planning

- Consider each identified risk and develop a **strategy** to manage that risk.
- categories

- Avoidance strategies
    - The probability that the risk will arise is reduced;
- Minimisation strategies
    - The impact of the risk on the project will be reduced;
- Contingency plans
    - If the risk arises, contingency plans are plans to deal with that risk. eg: financial problems

Risk management strategies

Risk	Strategy
Organisational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Recruitment problems	Alert customer of potential difficulties and the possibility of delays, investigate buying-in components.
Staff illness	Reorganise team so that there is more overlap of work and people therefore understand each other‘s jobs.
Defective components	Replace potentially defective components with bought-in components of known reliability.
Requirements changes	Derive traceability information to assess requirements change impact, maximise information hiding in the design.
Organisational restructuring	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Database performance	Investigate the possibility of buying a higher-performance database.
Underestimated development time	Investigate buying in components, investigate use of a program generator

Risk monitoring

- Assess each identified risks regularly to decide whether or not it is becoming less or more probable.
- Also assess whether the effects of the risk have changed.
- Cannot be observed directly. Factors affecting will give clues.
- Each key risk should be discussed at management progress meetings & review.

Risk indicators

Risk type	Potential indicators
Technology	Late delivery of hardware or support software, many reported technology problems
People	Poor staff morale, poor relationships amongst team member, job availability

Organisational	Organisational gossip, lack of action by senior management
Tools	Reluctance by team members to use tools, complaints about CASE tools, demands for higher-powered workstations
Requirements	Many requirements change requests, customer complaints
Estimation	Failure to meet agreed schedule, failure to clear reported defects