# SOFTWARE PROJECT MANAGEMENT

- Overview
- Software Project Metrics
- Software Project Estimation
- Software Project Planning

# TOPICS

Overview

Metrics

Estimation

Planning

# Overview

**To successfully manage software development, the project leader must determine:**

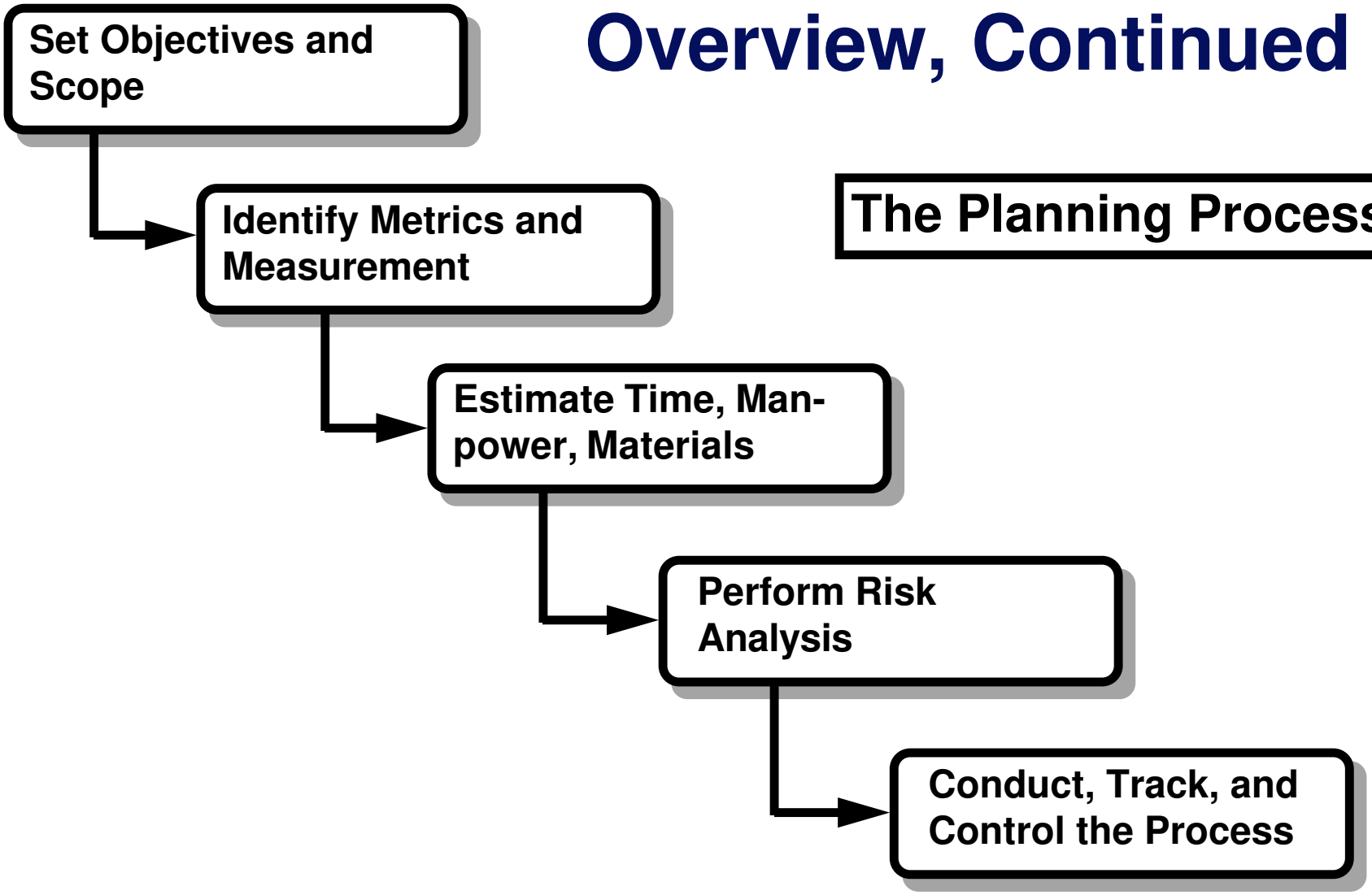
1. ***Scope*** **of work to be done**
2. ***Risks*** **to be incurred**
3. ***Resources*** **that will be required**
4. ***Tasks*** **to be accomplished**
5. ***Effort*** **(cost) that will be expended**
6. ***Schedule*** **to be followed**

**Software project management begins before the technical work starts.**

**Software project management ends when the software is retired.**

# Overview, Continued

**The Planning Process**



# SOFTWARE METRICS

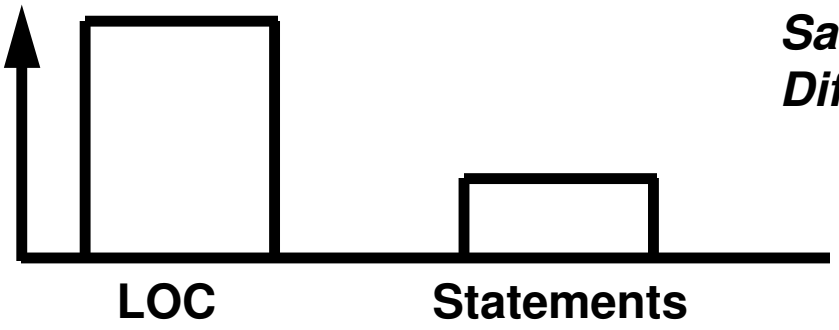
- Measuring Software
- Why Measure Software?
- Two Types of Measurements
- Categories of Metrics
- Size-Oriented Metrics
- Function Points
- Feature Points
- Function-Oriented Metrics
- Measuring Software Quality
- Relationship of LOC to FP
- Use of Productivity Data
- Integrating Metrics into the Software Engineering Process
- Collecting Software Metrics

# Measuring Software

- **Objectively measuring software is difficult.**
  - **Most projects use only "lines of code" (LOC) for metrics.**
  - **Much disagreement exists on what and how much to measure.**

**but**

- **Accurately measuring software is vitally important to tracking and controlling software development.**




***Same Project***
***Different Metrics***

# Why Measure Software?

To --

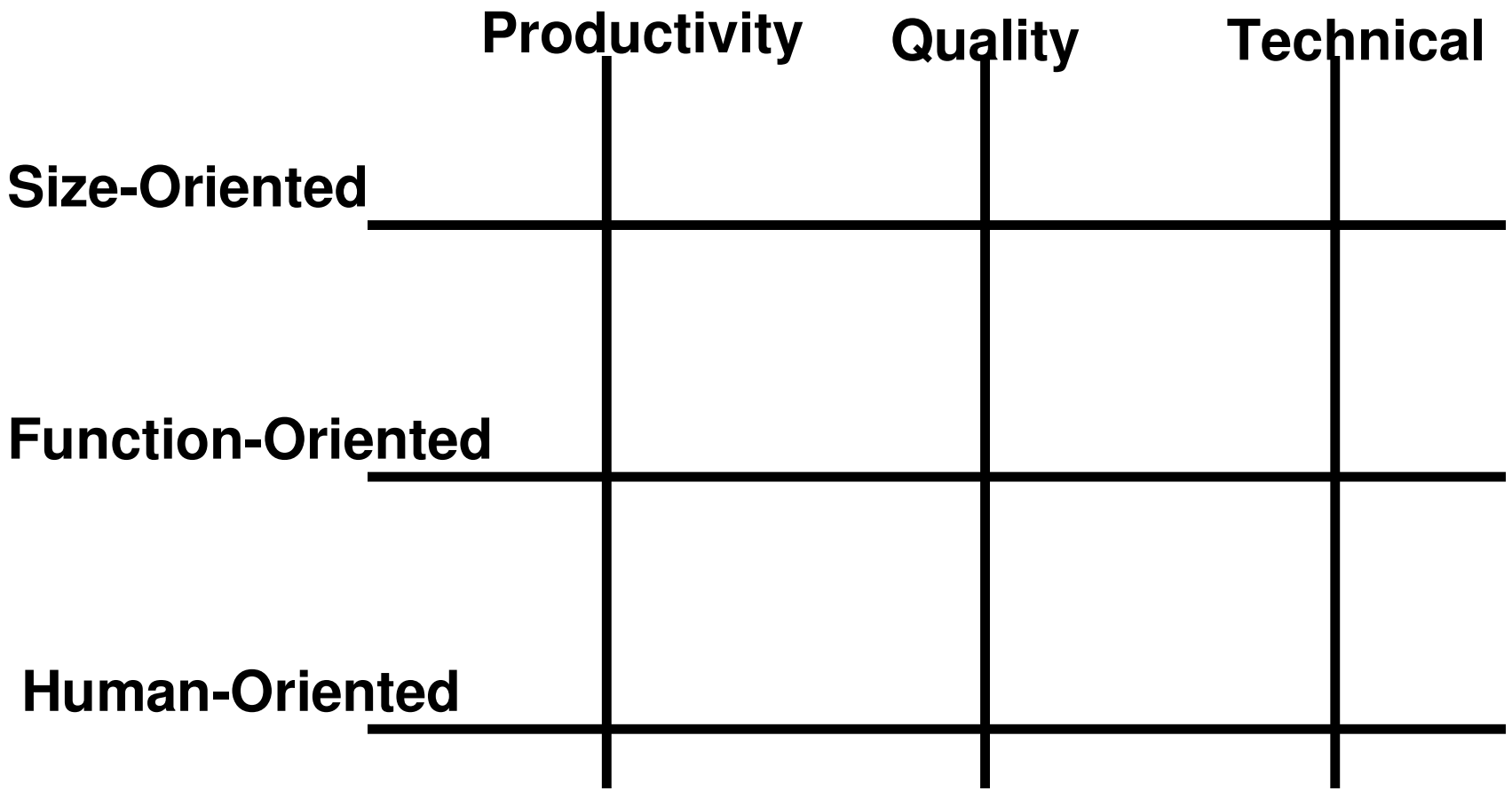
1. identify quality of the software product
2. assess productivity of the software developers
3. assess benefits of using development processes and tools
4. form a baseline for estimation
5. justify requests for tools and training

# Two Types of Measurements

- **Direct**
  - -- cost
  - -- LOC
  - -- execution speed
  - -- binary code size
  - -- memory used
- ♦ easy to make

- **Indirect**
  - -- functionality
  - -- quality
  - -- "-ilities"
- ♦ not easy to make

# Categories of Metrics



# Size-Oriented Metrics

**Let *KLOC* = "thousand lines of code"**

**Then we can define**

- ***productivity = KLOC / person-months***
- ***quality = defects in code / KLOC***
- ***cost = dollars / KLOC***
- ***documentation = pages of documents / KLOC***

**Efforts and costs include all elements of software development (analysis, design, code, test, *etc.*).**

# Size-Oriented Metrics - Examples

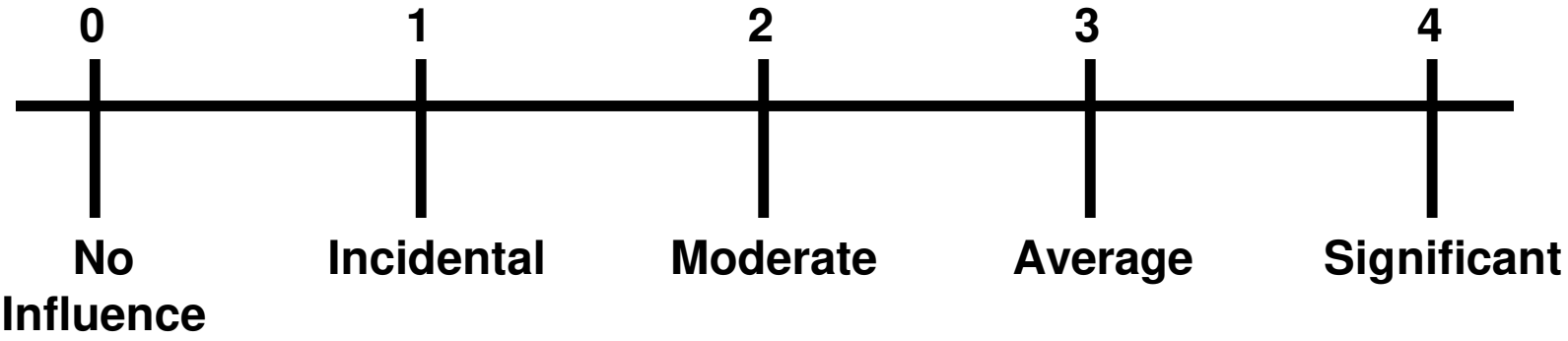
| Project | Person-Months | Cost | KLOC | Pages of Doc | Errors |
|---|---|---|---|---|---|
| A | 24 | $168,000 | 12.1 | 365 | 29 |
| B | 62 | $440,000 | 27.2 | 1224 | 86 |
| C | 43 | $314,000 | 20.2 | 1050 | 64 |

| Project | Productivity (KLOC/p-months) | Quality (errors/KLOC) | Cost ($/LOC) | Documents (pages/KLOC) |
|---|---|---|---|---|
| A | 0.504 | 2.40 | $13.88 | 30.17 |
| B | 0.439 | 3.55 | $16.18 | 45.00 |
| C | 0.470 | 3.67 | $15.54 | 51.98 |

# Problems with Size-Oriented Metrics

- Definition of "lines of code"
  - Programming language dependent
  - Penalize well-designed shorter programs
  - Cannot easily accommodate non-procedural languages
  - Difficult to assess LOC before a program is written
- Only known errors can be counted
- Types, skill levels, and productivity of personnel varies

# Function Points - Fi Values




1. Does the system require reliable backup?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use?

# Function Points - Computation

| Measurement Parameter | Count | Weighting Factor Simple | Average | Complex | Product |
|---|---|---|---|---|---|
| Number of user inputs | | x 3 | 4 | 6 = | |
| Number of user outputs | | x 4 | 5 | 7 = | |
| Number of user inquiries | | x 3 | 4 | 6 = | |
| Number of files | | x 7 | 10 | 15 = | |
| Number of external interfaces | | x 5 | 7 | 10 = | |

Count - Total →

$$FP = count - total(0.65 + 0.01 \sum F_i)$$

# Feature Points

## Function Point Extensions for Technical Software

- Function points were originally designed for business information systems applications.
- Extensions called *feature points* apply to technical software applications.
- Algorithms are a bounded computational problem that is included within a specific computer program.

# Feature Points - Computation

| Measurement Parameter | Count | Weight | | Product |
|---|---|---|---|---|
| Number of user inputs | | x 4 | = | |
| Number of user outputs | | x 5 | = | |
| Number of user inquiries | | x 4 | = | |
| Number of files | | x 7 | = | |
| Number of external interfaces | | x 7 | = | |
| Algorithms | | x 3 | = | |

Count - Total →

$$FP = count - total(0.65 + 0.01 \sum F_i)$$

# Problems with Function Points and Feature Points

1. These metrics are based on subjective data.
2. Parameters can be difficult to obtain after-the-fact.
3. Function and Feature Points have no direct physical meaning.

# Function-Oriented Metrics

- Focus is on "functionality" or "utility"
- Both Function Points and Feature Points support the derivation of potentially useful data for the comparison of one project to another:

  - Productivity = FP / person-month
  - Quality = defects / FP
  - Cost = $ / FP
  - Documentation = pages / FP

# Measuring Software Quality

Before Delivery

- Program complexity
- Effective modularity
- Program size

After Delivery (most widely used)

- Number of defects uncovered in the field
- Maintainability of the system

# "After Delivery" Quality Metrics

- *Correctness* - defects/KLOC or defects/FP over a one-year period
- *Maintainability* - mean-time-to-change (MTTC), which is the time required to:
  - analyze the change request,
  - design a modification to the software,
  - implement the change,
  - test the changed software and the system as a whole, and
  - distribute the changed system to the users

# "After Delivery" Quality Metrics, Continued

- ***Integrity*** **- based on threats and security**
  - ***Threat*** **- probability that a specific attack will take place within a given period of time**
  - ***Security*** **- probability that the attack of a specific type will be repelled**

$$\text{Integrity} = \sum_{\text{allthreats}} (1 - \text{threat}\,(1 - \text{security}\,))$$

- ***Useability*** **- based on several perceptions of the users:**
  - **skill required to use the program**
  - **time required to learn the use of the program**
  - **the increase in productivity from using the program**
  - **the user's attitude towards the program**

# Relationship of LOC to FP

- The relationship of lines of code to feature points is a function of the programming language used and the quality of the design.
- Rough estimates of the number of lines of code to create on feature point are:

| *Language* | *LOC/FP* |
|---|---|
| Assembly | 300 |
| COBOL | 100 |
| FORTRAN | 100 |
| Pascal | 90 |
| Ada | 70 |
| Object-Oriented Languages | 30 |
| Fourth Generation Languages | 20 |
| Automatic Code Generators | 15 |

# Use of Software Productivity Data

- Do not use LOC/person-month or FP/person-month to:
  - Compare one group of developers to another
  - Rate the performance of an individual
- Many factors affect productivity:

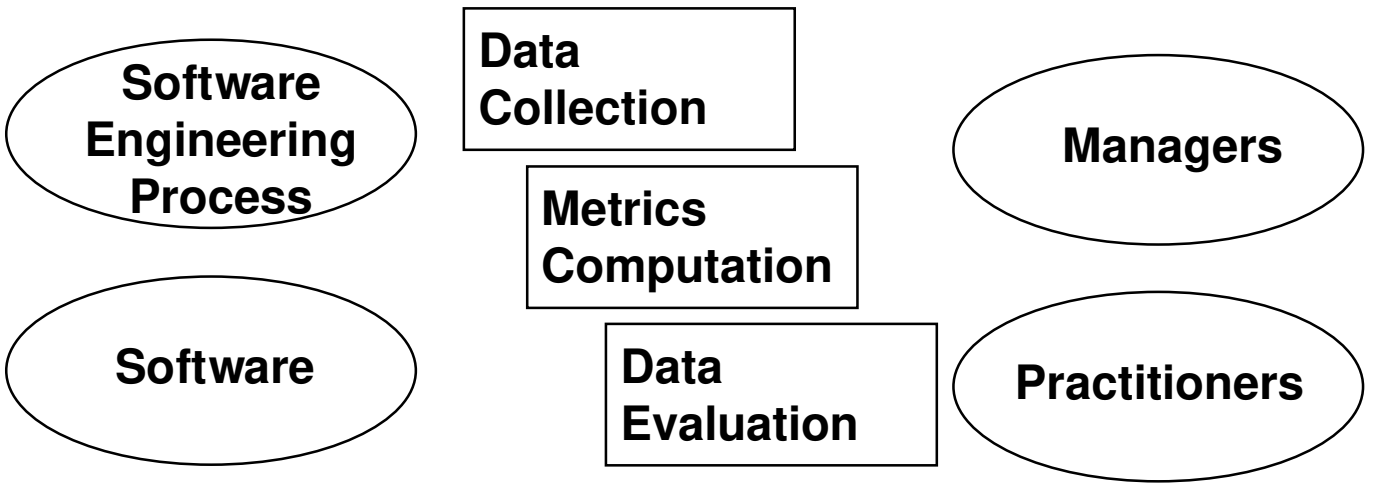
| *Factor* | *Approximate % Variation in Productivity* |
|---|---|
| People (number, experience) | 90% |
| Problem (complexity, number of changes) | 40% |
| Process (language, CASE) | 50% |
| Product (reliability, environment) | 140% |
| Resources (CASE, hardware, software) | 40% |

# Integrating Metrics into the Software Engineering Process

- A historical baseline of metrics data is needed:
  - Company, department, or unit should be identified in the scope of this data.
  - Resistance to data collection should be expected in many corporate cultures.
- At least three years of accurate, standardized metric data collection is needed to produce accurate planning estimates.

# Collecting Software Metrics

- The process of collecting and using software metrics includes the following steps:
  1. data collection
  2. metrics computation
  3. data evaluation
- The following slides show a spreadsheet model for the collection and computation of historical software baseline data.



# Spreadsheet Data Collection Model

| *Description* | *Units* | *Sample Data* |
|---|---|---|
| ● Cost Data Input | | |
| Labor cost | $/person-month | $7,744 |
| Labor year | hours/year | 1560 |
| ● Data for Metrics Computation | | |
| Release type | alphanumeric | maintenance |
| Number of staff members | people | 3 |
| Effort | person-hours | 4800 |
| Elapsed time to complete | hours | 2000 |
| Source code | KLOC | |
| Newly developed | | 11.5 |
| Modified | | 0.4 |
| Reused | | 0.8 |
| Delivered | | 33.4 |

# Spreadsheet Data Collection Model

- Data for Metrics Computation, Continued

| *Description* | *Units* | *Sample Data* |
|---|---|---|
| Documentation | pages | |
| Technical | | 265 |
| User | | 122 |
| Number of errors to date | numeric | |
| Critical errors | | 0 |
| Level 1 errors | | 12 |
| Level 2 errors | | 14 |
| Documentation errors | | 40 |
| Maintenance to date | person-hours | |
| Modifications | | 3550 |
| Error correction | | 1970 |

# Spreadsheet Data Collection Model

| *Description* | *Units* | *Sample Data* |
|---|---|---|
| ● **Project Data** | **% of total** | |
| **Analysis and specification** | | **18%** |
| **Design** | | **20%** |
| **Coding** | | **23%** |
| **Testing** | | **25%** |
| **Other - Describe** | | **14%** |

# Spreadsheet Data Collection Model

- **Function-Oriented Data**

| *Description* | *Units* | *Sample Data* |
|---|---|---|
| **Information Domain** | | |
| **1. No. of user inputs** | **inputs** | **24** |
| **2. No. of user outputs** | **outputs** | **46** |
| **3. No. of user inquiries** | **inquiries** | **8** |
| **4. No. of files** | **files** | **4** |
| **5. No. of ext. interfaces** | **interfaces** | **2** |
| **Weights** | | |
| **1. No. of user inputs** | **3, 4, 6** | **4** |
| **2. No. of user outputs** | **4, 5, 7** | **4** |
| **3. No. of user inquiries** | **3, 4, 6** | **6** |
| **4. No. of files** | **7, 10, 15** | **10** |
| **5. No. of ext. interfaces** | **5, 7, 10** | **5** |

# Spreadsheet Data Collection Model

| *Description* | *Units* | *Sample Data* |
|---|---|---|
| ● **Function-Oriented Data, Continued** | | |
| **Processing Complexity Factors** | **0-5** | |
| 1. backup and recovery required | | 4 |
| 2. data communication required | | 1 |
| 3. distributed processing function | | 0 |
| 4. performance critical | | 3 |
| 5. heavily utilized operating environment | | 3 |
| 6. online data entry | | 5 |
| 7. input transaction with multiple screens | | 4 |
| 8. master files updated online | | 4 |
| 9. input, output, files, queries complex | | 3 |
| 10. internal processing complex | | 3 |
| 11. code designed to be reusable | | 2 |
| 12. conversion/installation included in design | | 2 |
| 13. system design for multiple installation | | 4 |
| 14. maintainability/ease of use | | 5 |

# Spreadsheet Data Collection Model

- Size-Oriented Metrics

| *Description* | *Units* | *Sample Data* |
|---|---|---|
| **Productivity and Cost** | | |
| Output | KLOC/p-month | 0.905 |
| Cost - all code | $/KLOC | $22,514 |
| Cost - exclude reuse | $/KLOC | $24,028 |
| Elapsed time | months/KLOC | 1.0 |
| Documentation | pages/KLOC | 30 |
| Documentation | pages/p-month | 10 |
| Documentation | $/page | $739 |
| **Quality** | | |
| Defects | errors/KLOC | 2.0 |
| Cost of errors | $/error | $376 |

# Spreadsheet Data Collection Model

| *Description* | *Units* | *Sample Data* |
|---|---|---|
| ● **Function-Oriented Metrics** | | |
| **Productivity and Cost** | | |
| **Output** | **FP/p-month** | **378** |
| **Cost - all code** | **$/FP** | **$700** |
| **Elapsed time** | **FP/month** | **31.4** |
| **Documentation** | **pages/FP** | **0.9** |
| **Quality** | | |
| **Defects** | **errors/FP** | **0.064** |

# SOFTWARE PROJECT ESTIMATION

- Overview
- Resources
- Decomposition Techniques
- Using LOC or FP to Estimate Effort
- Effort Estimation by Function
- Effort Estimation by Task
- Empirical Estimation Models
- COCOMO
- Putman Estimation Model

# Overview

**Estimation of:**

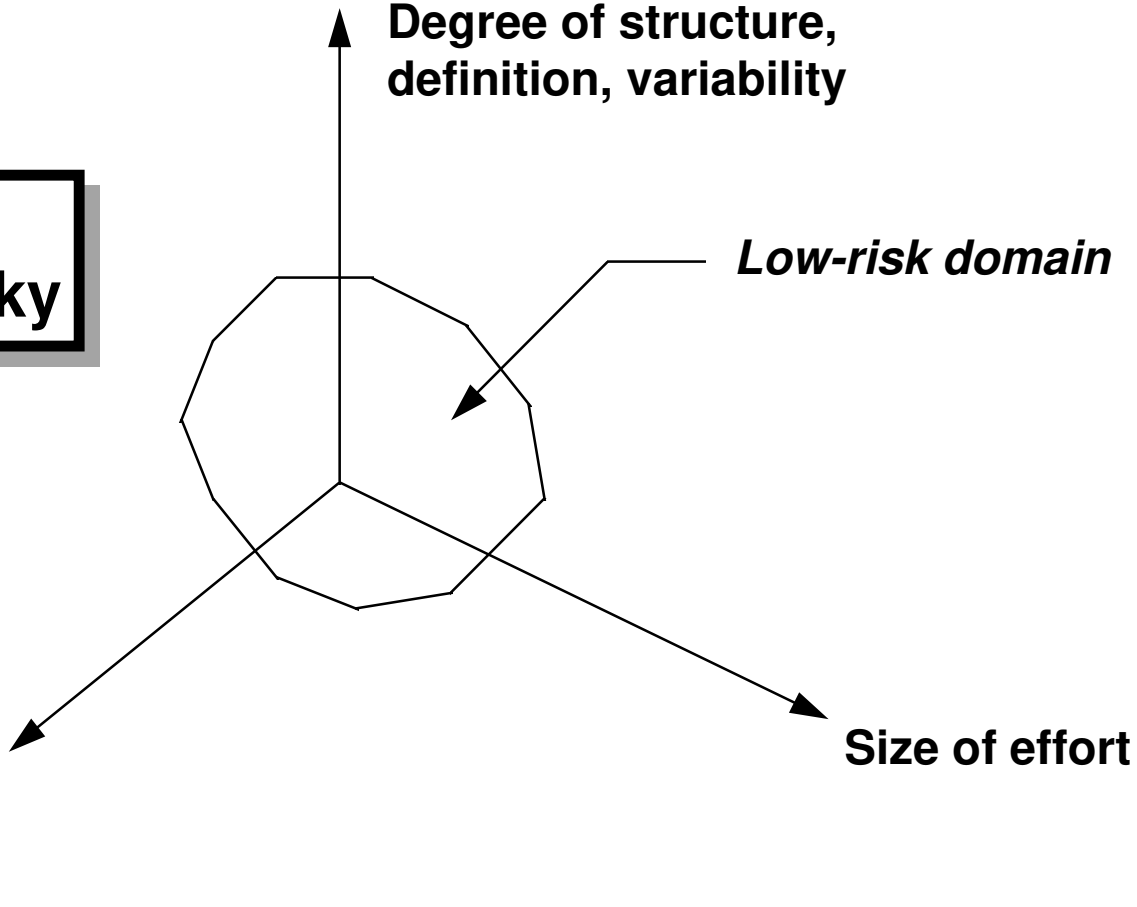
- **resources**
- **costs**
- **schedules**

**Requires:**

- **experience**
- **historical information**
- **quantitative measures of qualitative data**

# Overview, Continued

**Estimation can be Risky**



# Resources

**Planning Task 1: Software Scope**

1. Statement of software scope must be bounded
2. Software scope describes:
   - function
   - performance
   - constraints
   - interfaces
   - reliability

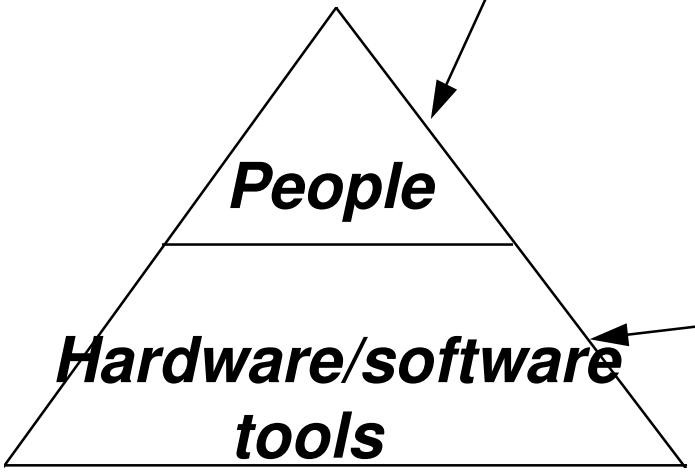
*evaluated together*

# Resources, Continued

**Planning Task 2: Estimation of Needed Resources**

**Specify:**

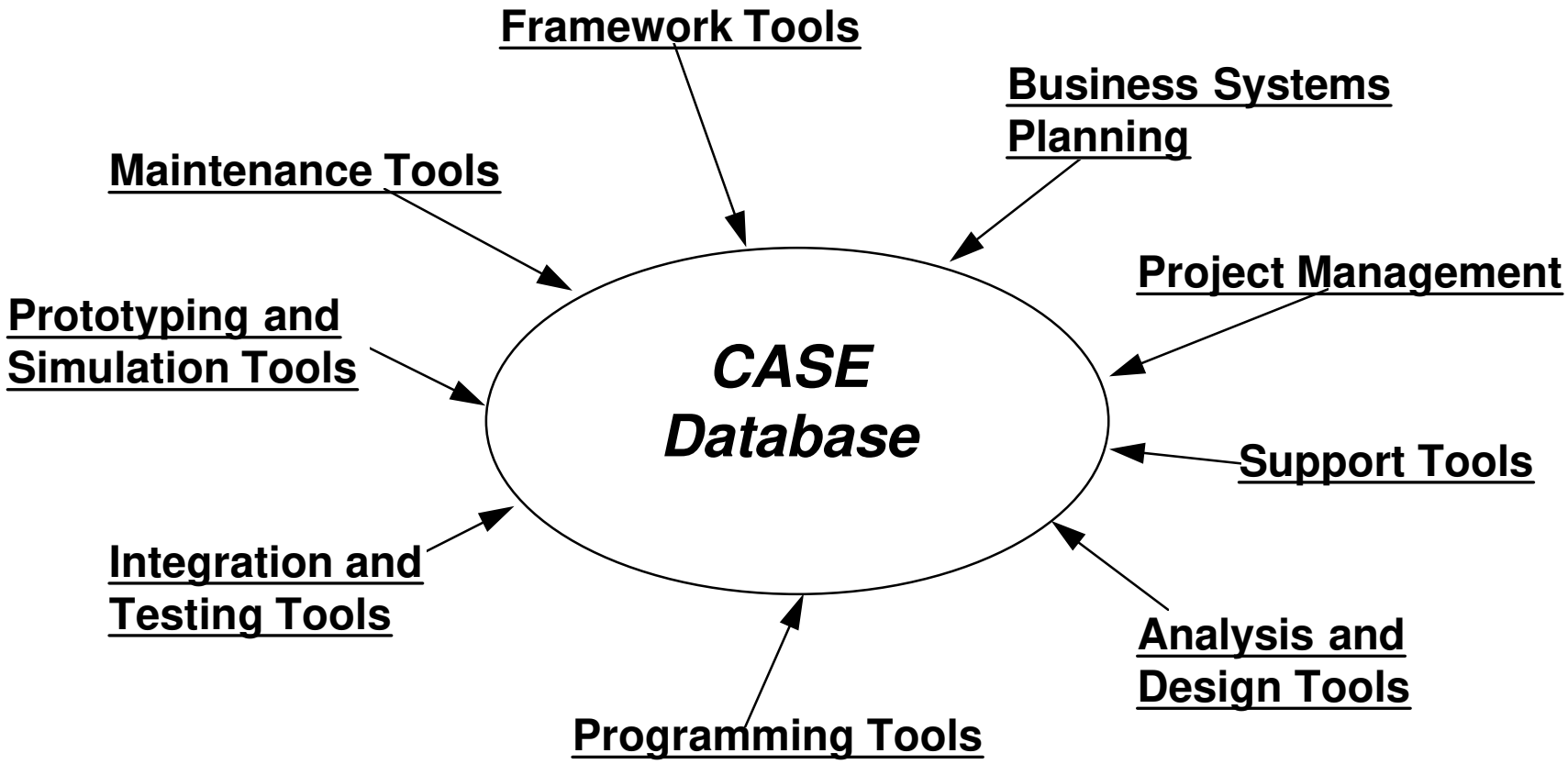
- Required skills
- Availability
- Duration of tasks
- Start date




**Specify:**

- Description
- Availability
- Duration of use
- Delivery date

# Resources, Continued




**CASE - Computer-Aided Software Engineering**

# Resources, Continued

## Reuse - A Resource

**Two rules:**

1. **If existing software meets requirements, then**

   **acquire and use it!**

2. **If existing software can meet requirements with some modification, then**

   **be careful!**

**The cost of modification can exceed the cost of new development!**

# Decomposition Techniques

- LOC and FP Estimation
- Effort Estimation

# Decomposition Techniques, Continued

## LOC and FP Estimation

The idea is that the person planning the software project:

- creates a bounded statement of the scope of the software
- decomposes the scope into smaller subfunctions
- estimates LOC or FP for each subfunction
- applies baseline productivity metrics (e.g., LOC/person-month) to LOC or FP estimates to produce a cost or effort estimate for each subfunction
- combines estimates for each subfunction to derive estimates for the entire project

# Decomposition Techniques, Continued

## Differences Between LOC and FP

- FP estimation techniques require less detail than LOC
- LOC is estimate directly while FP is estimated indirectly

# Using LOC or FP to Estimate Effort

1. Estimate LOC or FP values for each subfunction
   - Use historical data (or intuition, if necessary)
   - Three estimates: optimistic (o), most likely (m), and pessimistic (b)
2. Calculate expected value for each subfunction $E = \frac{a + 4m + b}{6}$
3. Apply productivity data to get effort to be expended; two ways:
   1. Total expected LOC or FP for all subfunctions and divide this by the expected LOC or FP completed per person-month (estimated from past projects); example:

*Effort = 310 expected FP for project/5.5 expected FP per person-month*

*= 56 person-months*

   2. Multiply each subfunction LOC or FP by the adjusted productivity value (based on the estimated complexity of the function) and sum the results for all subfunctions in the project

# Effort Estimation by Function

## CAD Program Example

| *Function* | *Optimistic* | *Most Likely* | *Pessimistic* | *Expected* | *$/Line* | *Line/Month* | *Cost* | *Months* |
|---|---|---|---|---|---|---|---|---|
| User interface control | 1800 | 2400 | 2650 | 2,340 | $14 | 315 | $ 32,760 | 7.4 |
| 2-D geometric analysis | 4100 | 5200 | 7400 | 5,380 | $20 | 220 | $107,600 | 24.4 |
| 3-D geometric analysis | 4600 | 6900 | 8600 | 6,800 | $20 | 220 | $136,000 | 30.9 |
| Data structure mgmt | 2950 | 3400 | 3600 | 3,350 | $18 | 240 | $ 60,300 | 13.9 |
| Graphics display | 4050 | 4900 | 6200 | 4,950 | $22 | 200 | $108,900 | 24.7 |
| Peripheral control | 2000 | 2100 | 2450 | 2,140 | $28 | 140 | $ 59,920 | 15.2 |
| Design analysis | 6600 | 8500 | 9800 | 8,400 | $18 | 300 | $151,200 | 28.0 |
| Estimated Effort | | | | 33,360 | | | $656,680 | 144.5 |

**Estimated Cost: $ 656,680**
**Estimated Effort: 144.5 person-months**

# Effort Estimation by Task

## CAD Program Example

| *Function* | *RA* | *Design* | *Code* | *Test* | *Total* |
|---|---|---|---|---|---|
| User interface control | 1.0 | 2.0 | 0.5 | 3.5 | 7.0 |
| 2-D geometric analysis | 2.0 | 10.0 | 4.5 | 9.5 | 26.0 |
| 3-D geometric analysis | 2.5 | 12.0 | 6.0 | 11.0 | 31.5 |
| Data structure mgmt | 2.0 | 6.0 | 3.0 | 4.0 | 15.0 |
| Graphics display | 1.5 | 11.0 | 4.0 | 10.5 | 27.0 |
| Peripheral control | 1.5 | 6.0 | 3.5 | 5.0 | 16.0 |
| Design analysis | 4.0 | 14.0 | 5.0 | 7.0 | 30.0 |
| | | | | | |
| Total | 14.5 | 61.0 | 26.5 | 50.5 | 152.5 |
| Rate ($) | 5200 | 4800 | 4250 | 4500 | |
| Cost ($) | 75,400 | 292,800 | 112,625 | 227,250 | 708,075 |

**Estimated Cost: $ 708,075**
**Estimated Effort: 152.5 person-months**

# Empirical Estimation Models

- **Static single-variable model (example: COCOMO)**

$$\text{Resource} = cx^{d}$$

**where**

**x is the estimated characteristic (LOC, FP, effort, etc.)**

**c and d are constants derived from data collected from past projects**

- **Static multivariable model**

$$\text{Resource} = cx + dy + \ldots$$

**where**

**x, y, ... and c, d, ... are as above**

- **Dynamic multivariable model**

  **Project resource requirements are determined over a series of time steps**

- **Theoretical (example: Putman Estimation Model)**

  **Uses equations derived from hypothesized expenditure curves**

# COCOMO

- Involves basic, intermediate, and advanced models
- Basic model:

$$\text{Effort} = a(b)KLOC^{b(b)}\text{person} - \text{months}$$

$$\text{Development_Time} = c(b)Effort^{d(b)}\text{months}$$

a(b), b(b), c(b), and d(b) are determined from the table:

| *Software Project* | *a(b)* | *b(b)* | *c(b)* | *d(b)* |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

# COCOMO, Continued

Example of COCOMO basic model on the CAD program:

Effort = 3.0 (LOC) ^ 1.12

= 3.0 (33.3) ^ 1.12

= 152 person-months

Development Time = 2.5 (Effort) ^ 0.35

= 2.5 (152) ^ 0.35

= 14.5 months

Thus, estimated number of people N is:

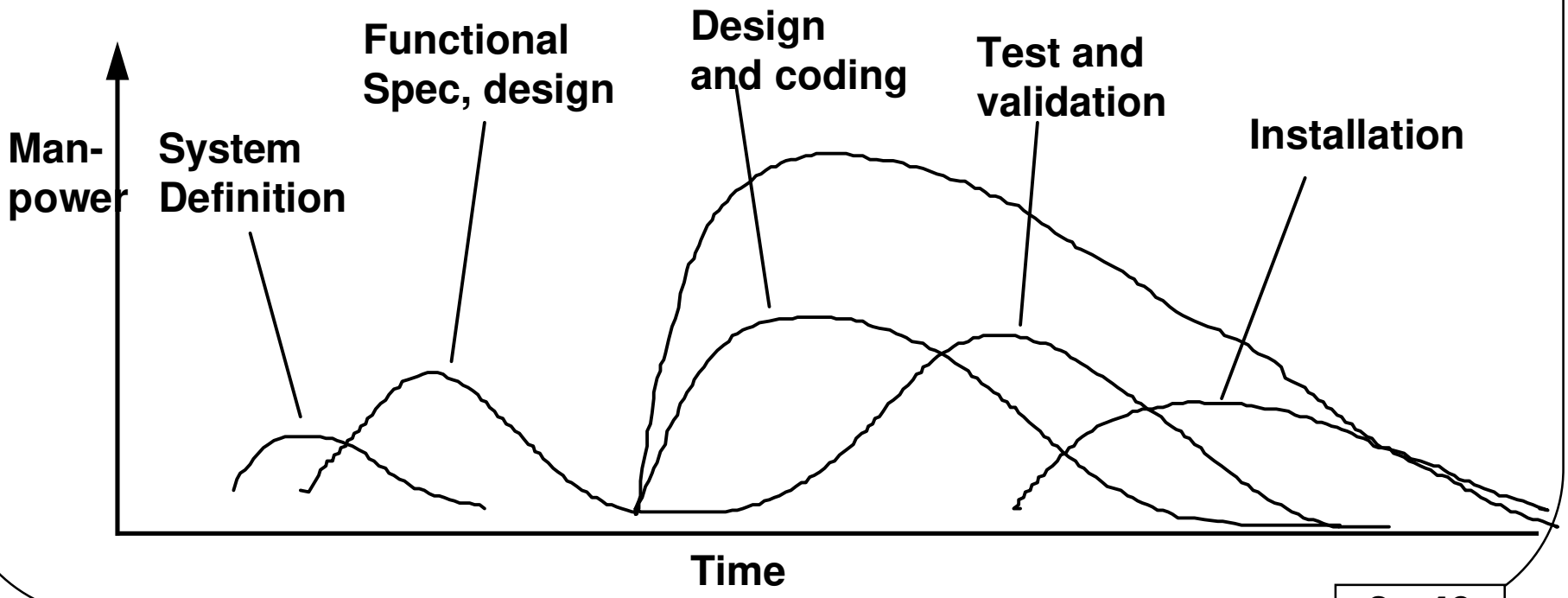
N = Effort / Development Time

= 152 / 14.5

= 11 people

# Putman Estimation Model

- Data is derived from large projects
- Model is applicable to smaller projects as well
- The distribution of effort is described by the Rayleigh-Norden curve



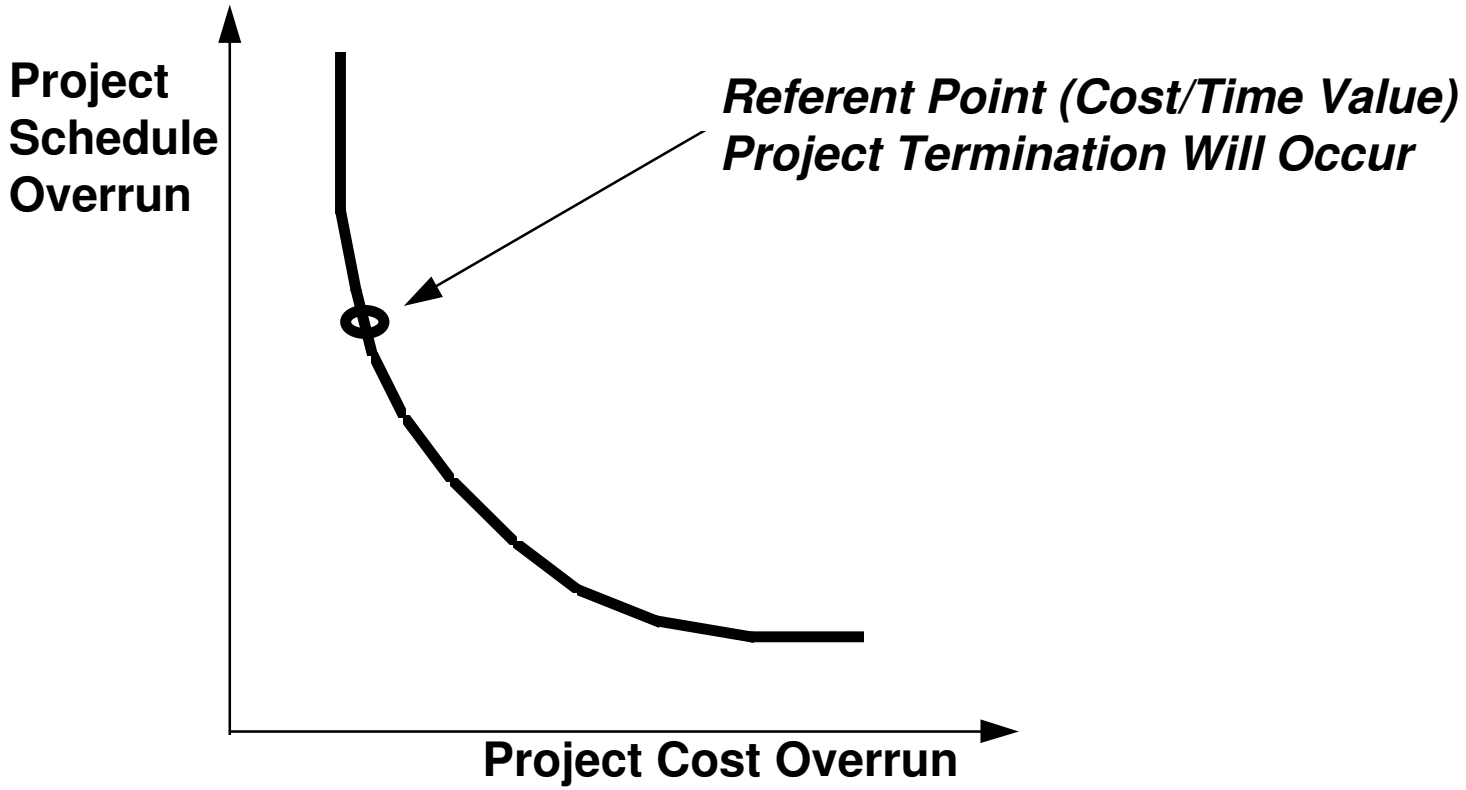
# SOFTWARE PROJECT PLANNING

- What Software Project Planning Involves
- Risk Analysis
- Risk Management
- Risk Monitoring - Project Tracking
- Software Project Scheduling
- Typical Task Network
- Approaches to Project Tracking
- Software Acquisition
- Software Acquisition Decision Tree
- Software Re-Engineering
- Organizational Planning
- Enhancements to a Good Organization
- The Software Project Plan (SPP)

# What Software Project Planning Involves

1. *Estimation*
2. **Risk Analysis**
3. **Scheduling**
4. **Acquisition Decision Making**
5. **Re-Engineering**
6. **Organizational Planning**

# Risk Analysis



# Risk Management

- Create risk management and monitoring plan
- For each risk triplet, define the risk management steps
- Risk management incurs additional project cost
- For larger projects, there may be 30-40 risks identified

## Example

Assume:

Risk = High staff turnover

Likelihood of occurrence = 70%

Impact = Increase project time by 15%, project cost by 12%

Risk Management steps may be:

1. Identify high turnover causes
2. Reduce causes before project starts
3. Develop techniques to assure work continuity in light of turnover

# Risk Monitoring - Project Tracking

1. Determine if predicted risk occurs
2. Properly apply risk aversion steps
3. Collect info for future risk analysis

# Software Project Scheduling

- People-work relationships
- Task definition and parallelism
- Effort distribution
- Scheduling methods
- An example

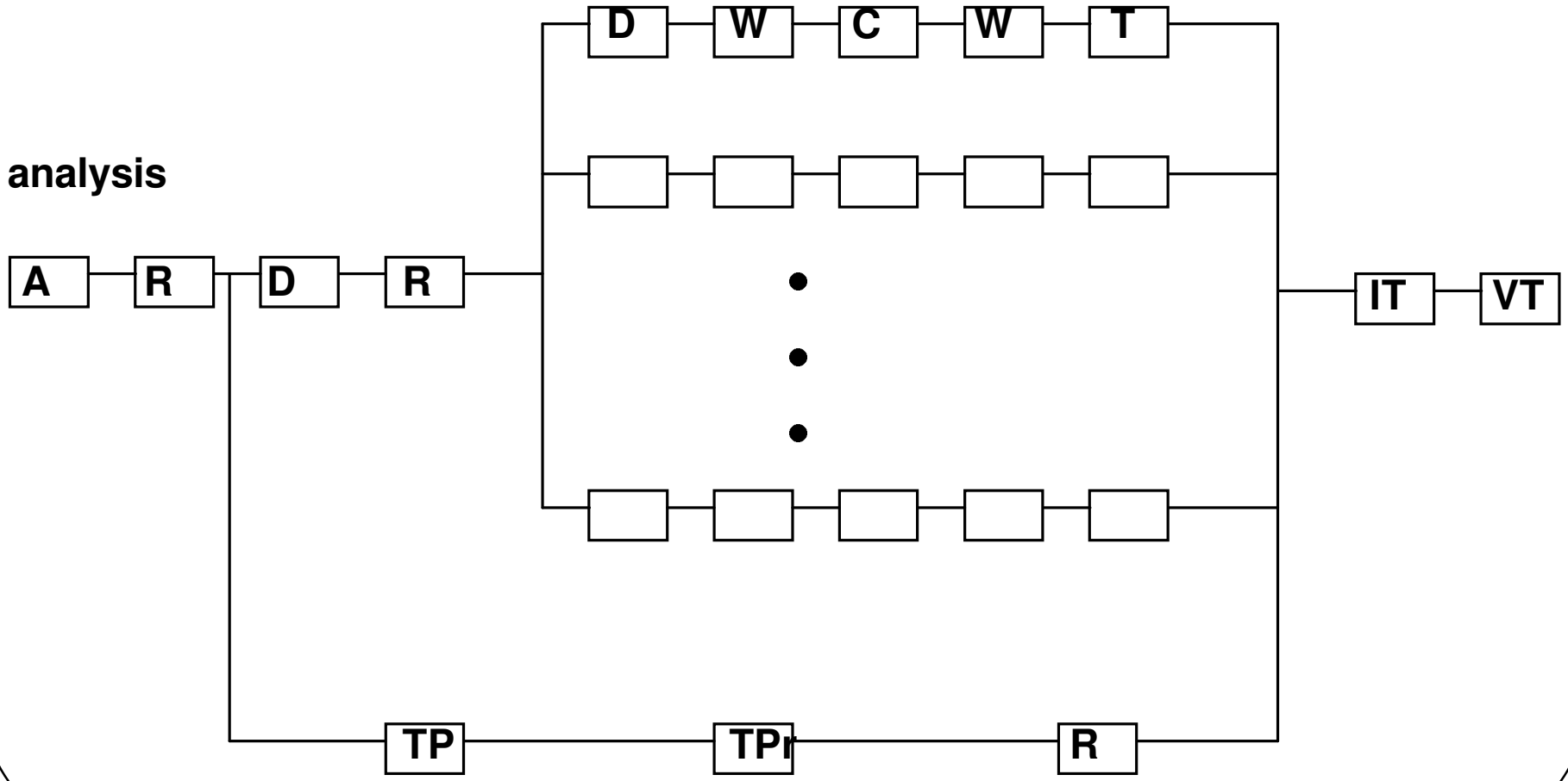
# Software Project Scheduling

## *People-Work Relationships*

- Adding people to a project when behind schedule is counterproductive (*adding people to a late project makes it later*)
- Using fewer people over a longer period of time is more beneficial than lots of people for a shorter period of time
- Use of small, tightly-knit teams is productive
- Inspire creativity and self-motivation within the structure of the project

# Software Project Scheduling

## *Task Definition and Parallelism*



# Software Project Scheduling

## *Task Definition and Parallelism*

### *Initial Sequential Events*

**Milestone 1 Occurs After --**

- System analysis and specification
- System requirements review

**Milestone 2 Occurs After --**

- System architecture and data design
- System preliminary design review

# Software Project Scheduling

## *Task Definition and Parallelism*

### *Parallel Events for Each Subfunction*

**Milestone P1 Occurs After --**

- **Procedural design**
- **Design walkthrough**

**Milestone P2 Occurs After --**

- **Coding**
- **Code walkthrough**

**Milestone P3 Occurs After --**

- **Unit testing**

# Software Project Scheduling

## *Task Definition and Parallelism*

### *System Testing Activities Can Be Performed In Parallel*

**Testing Milestone (After Unit Testing) --**

- **System test planning**
- **System test procedure**
- **System test review**

# Software Project Scheduling

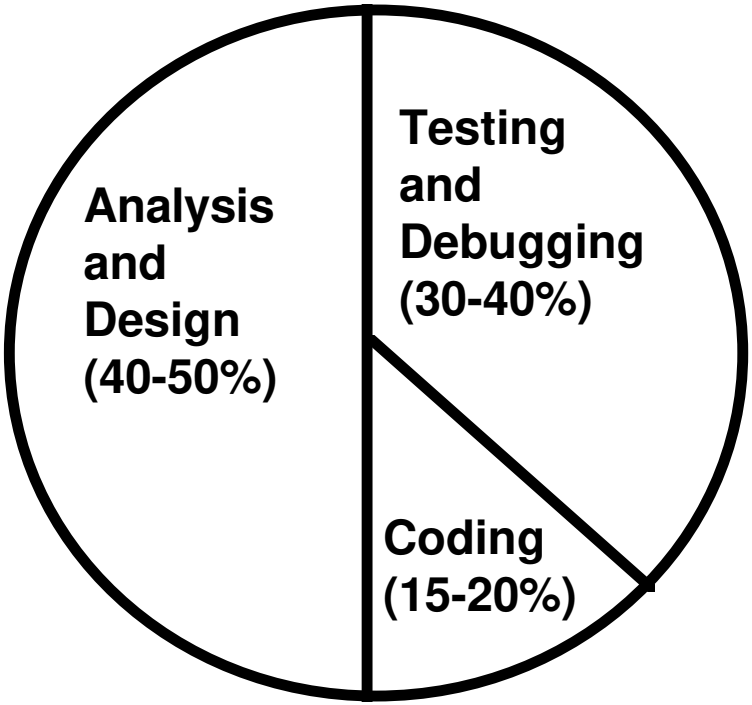
## *Task Definition and Parallelism*

**Integration Test Milestone - completed after system is assembled**

**Validation Test Milestone - completed last**

# Software Project Scheduling

## *Effort Distribution*



# Software Project Scheduling

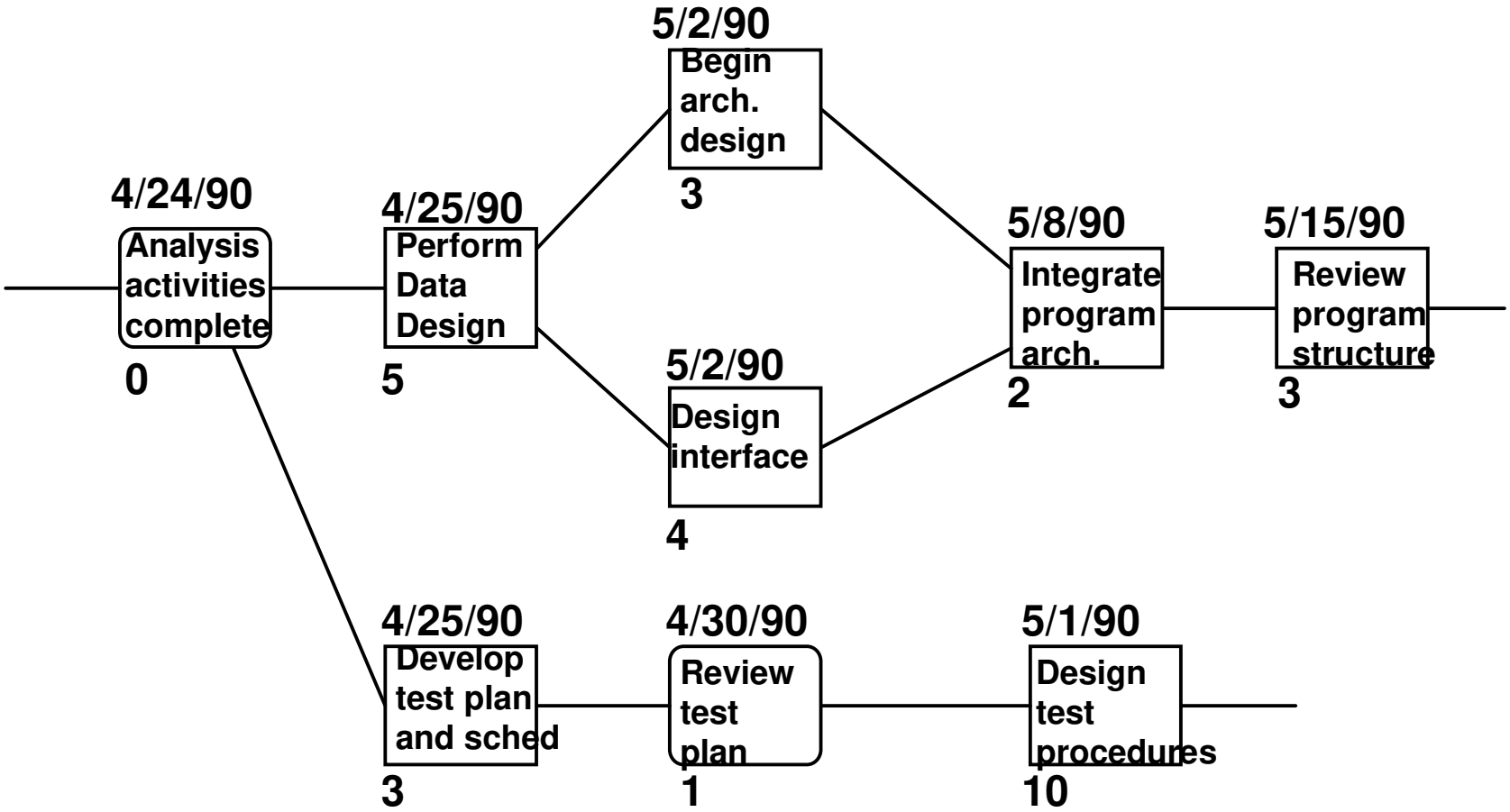
## *Scheduling Methods*

- PERT - *Program Evaluation and Review Technique*
- CPM - *Critical Path Method*

PERT and CPM are:

- Usually presented pictorially
- Quantitative tools for the planner to determine:
  - Critical path
  - Most likely time estimates
  - Boundary times (earliest task start time, latest task start time, earliest task finish time, latest task finish time, total float time)

# Typical Task Network



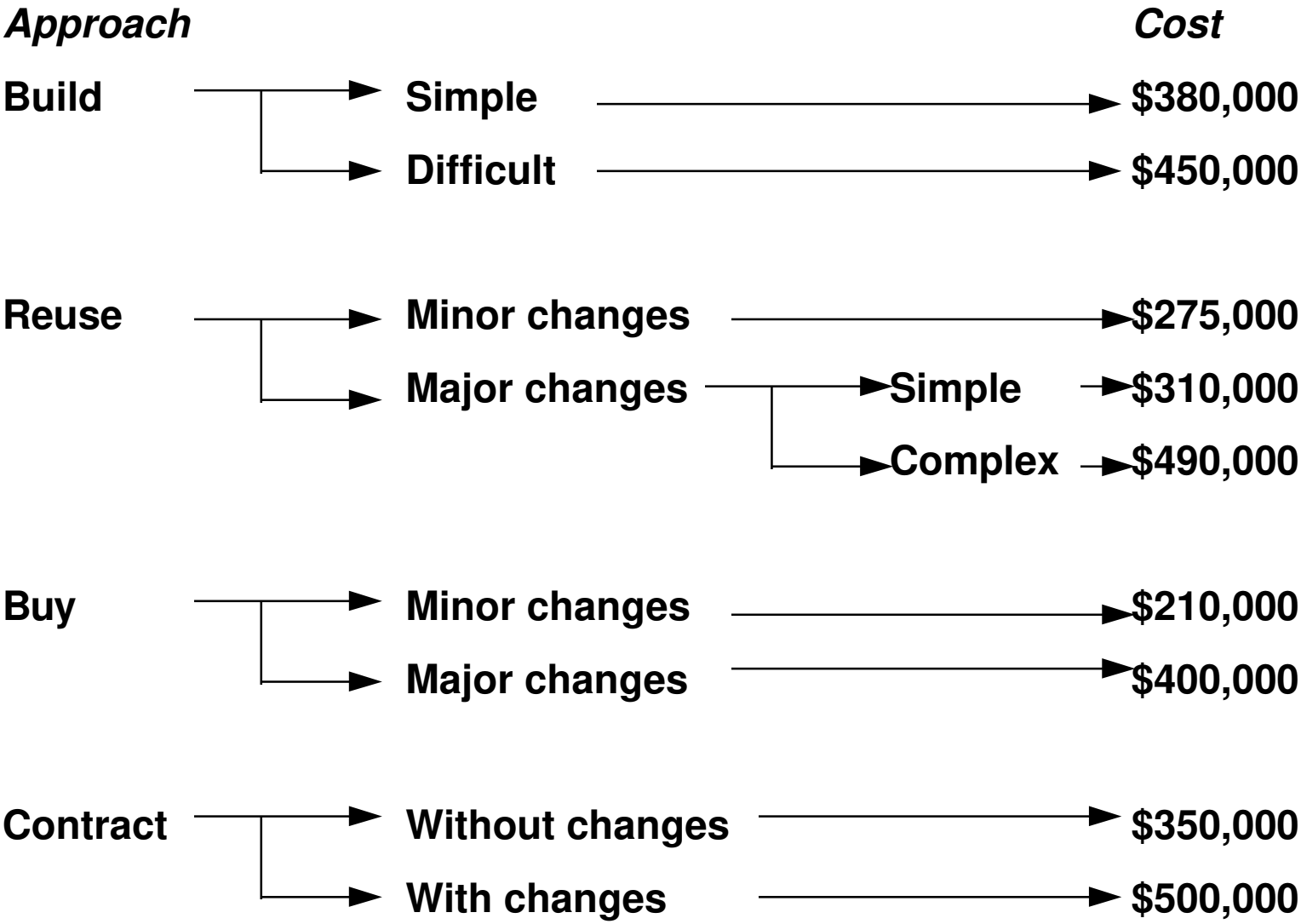
# Approaches to Project Tracking

- Conducting periodic project status meetings in which each team member reports progress and problems
- Evaluating the results of all reviews conducted throughout the engineering process
- Determining whether formal project milestones have been accomplished by the scheduled date
- Comparing the actual start date to the planned start date for each task
- Meeting informally with software engineers to obtain their subjective assessments of the progress to date and problems on the horizon

# Software Acquisition

- Make or buy?
  - Who will use?
  - Buy and modify?
  - Contact outside contractor to build?
- Decision based on:
  - Reduced cost
  - Earlier delivery date
  - Not enough or properly skilled people to develop
  - Better support outside

# Software Acqusition Decision Tree



# Software Re-Engineering

- For often-used programs, build a controlled database of components for all to use.
- Include documents, source code, user's guide, maintenance guide, test procedures and data, and a history of use with the components.
- Software re-engineering may be enhanced by object-oriented design and implementation.

# Organizational Planning

- There are lots of human organizational structures for software development
- Possibilities - consider N people working for K years on M different functional tasks

| *Approach* | *Level of Interaction* | *Coordination* |
|---|---|---|
| 1 Assign N people to M tasks ( M > N ) | Individual | Project Mgr |
| 2 Assign N people to M tasks ( M < N ) | Teams | Project Mgr, Team Leader |
| 3 Assign N people to T teams, each team resp. for 1 or more tasks | Formal Teams | Project Mgr, Team Leader |

# Enhancements to a Good Organization

- The Chief Programmer Team
- The Software Librarian
- Egoless programming with a team environment

# The Software Project Plan (SPP)

A brief document which describes --

- The scope of the project
- The resources to be used
- Risks and risk avoidance techniques
- Cost and schedule
- Overall approach to software development

Management, technical staff, and customer are the primary reads of the SPP.

The SPP provides a starting point for the rest of the project.