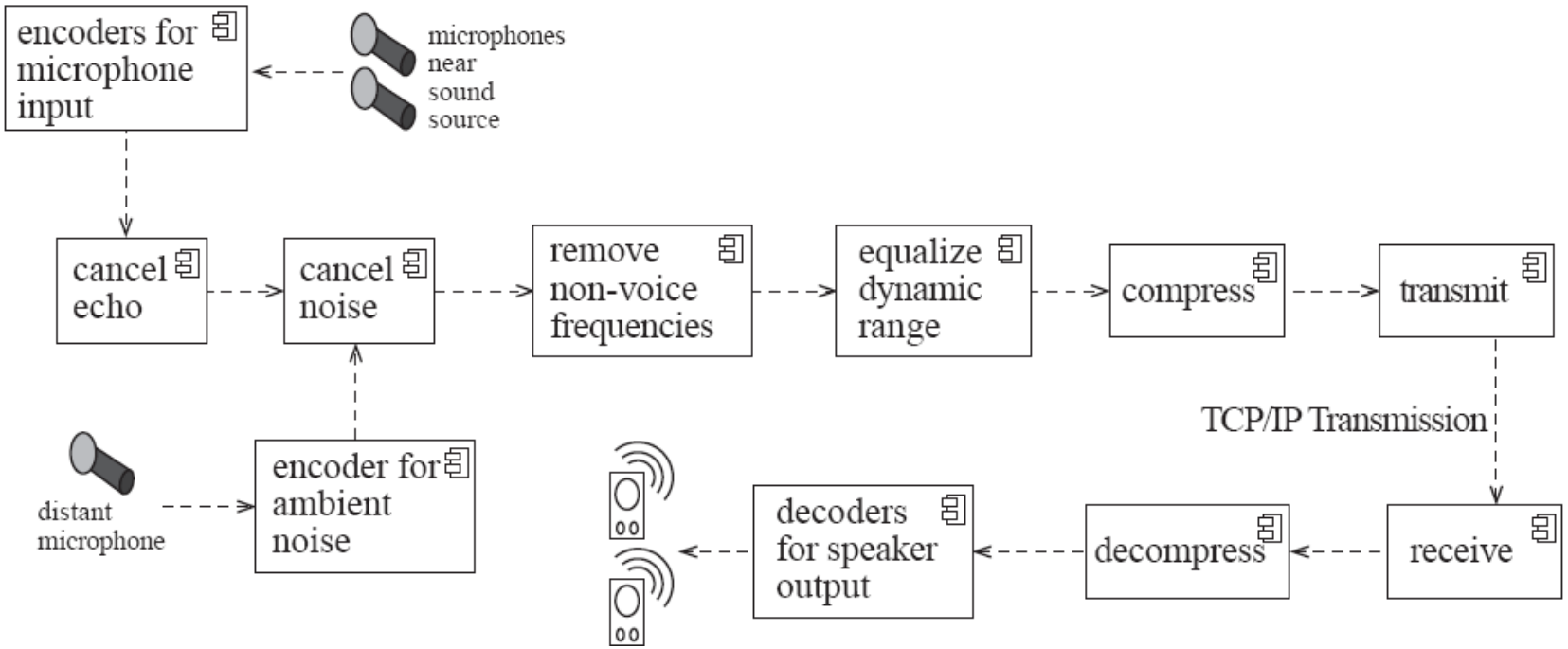
The Pipe-and-Filter architectural pattern

- A stream of data, in a relatively simple format, is passed through a series of processes
  - Each of which transforms it in some way.
  - Data is constantly fed into the pipeline.
    - Almost all the components could be removed.
    - Components could be replaced.
    - New components could be inserted.
    - Certain components could be reordered.

Example of a pipe-and-filter system



The pipe-and-filter architecture and design principles

Divide and conquer: The separate processes can be independently designed.

Increase cohesion: The processes have functional cohesion.

Reduce coupling: The processes have only one input and one output.

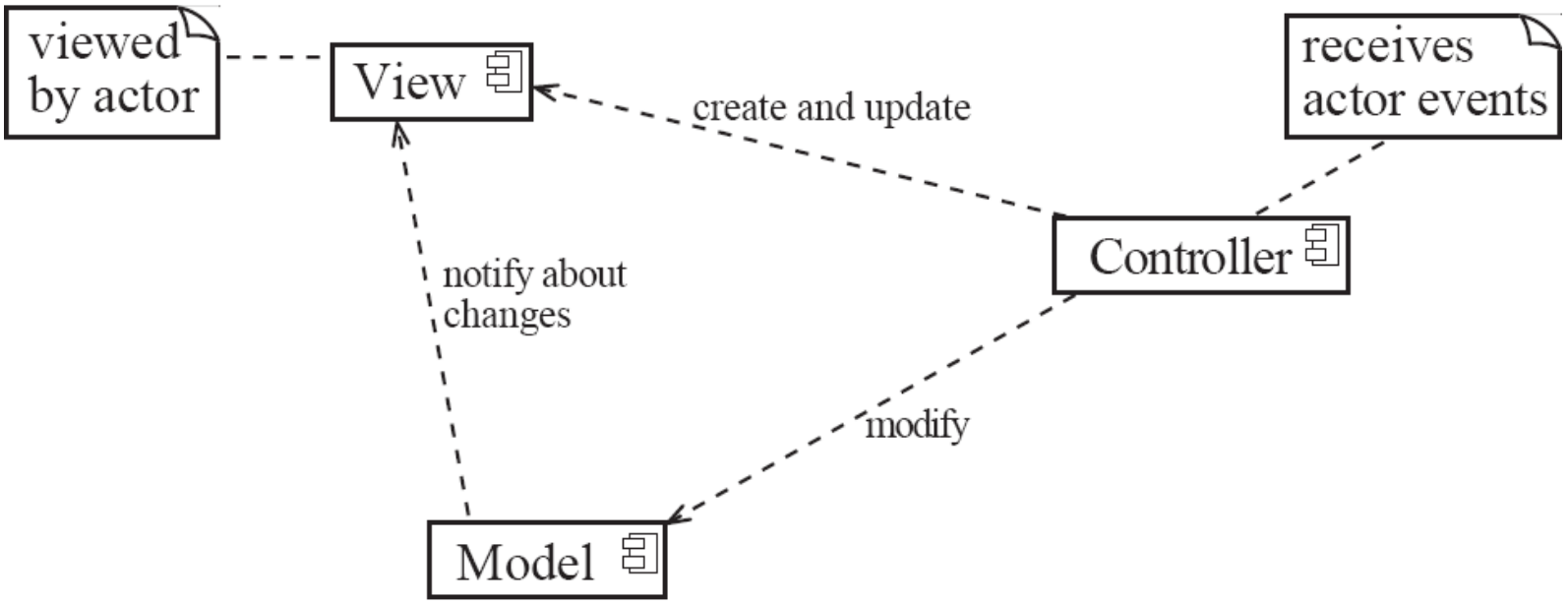
Increase abstraction: The pipeline components are often good abstractions, hiding their internal details.

Increase reuse: It is often possible to find reusable components to insert into a pipeline.

The Model-View-Controller (MVC) architectural pattern

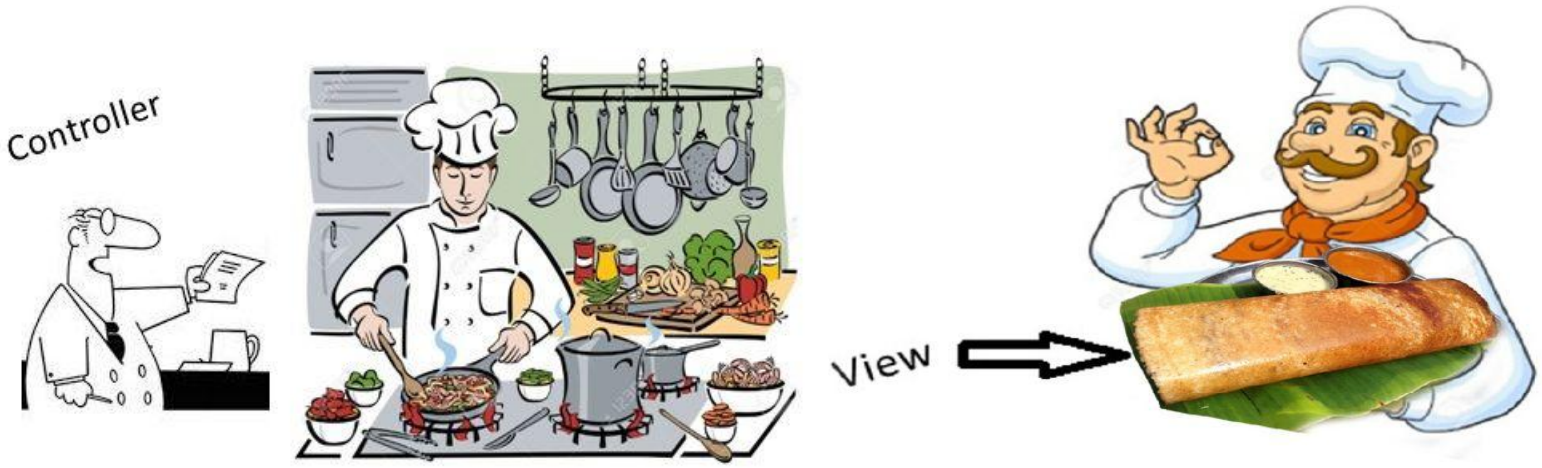
- An architectural pattern used to help separate the user interface layer from other parts of the system
  - The *model* contains the underlying classes whose instances are to be viewed and manipulated
  - The *view* contains objects used to render the appearance of the data from the model in the user interface
  - The *controller* contains the objects that control and handle the user's interaction with the view and the model

Example of the MVC architecture for the UI



Example of MVC in Web architecture

- The *View* component generates the HTML code to be displayed by the browser.
- The *Controller* is the component that interprets 'HTTP post' transmissions coming back from the browser.
- The *Model* is the underlying system that manages the information.



The MVC architecture and design principles

Divide and conquer: The three components can be somewhat independently designed.

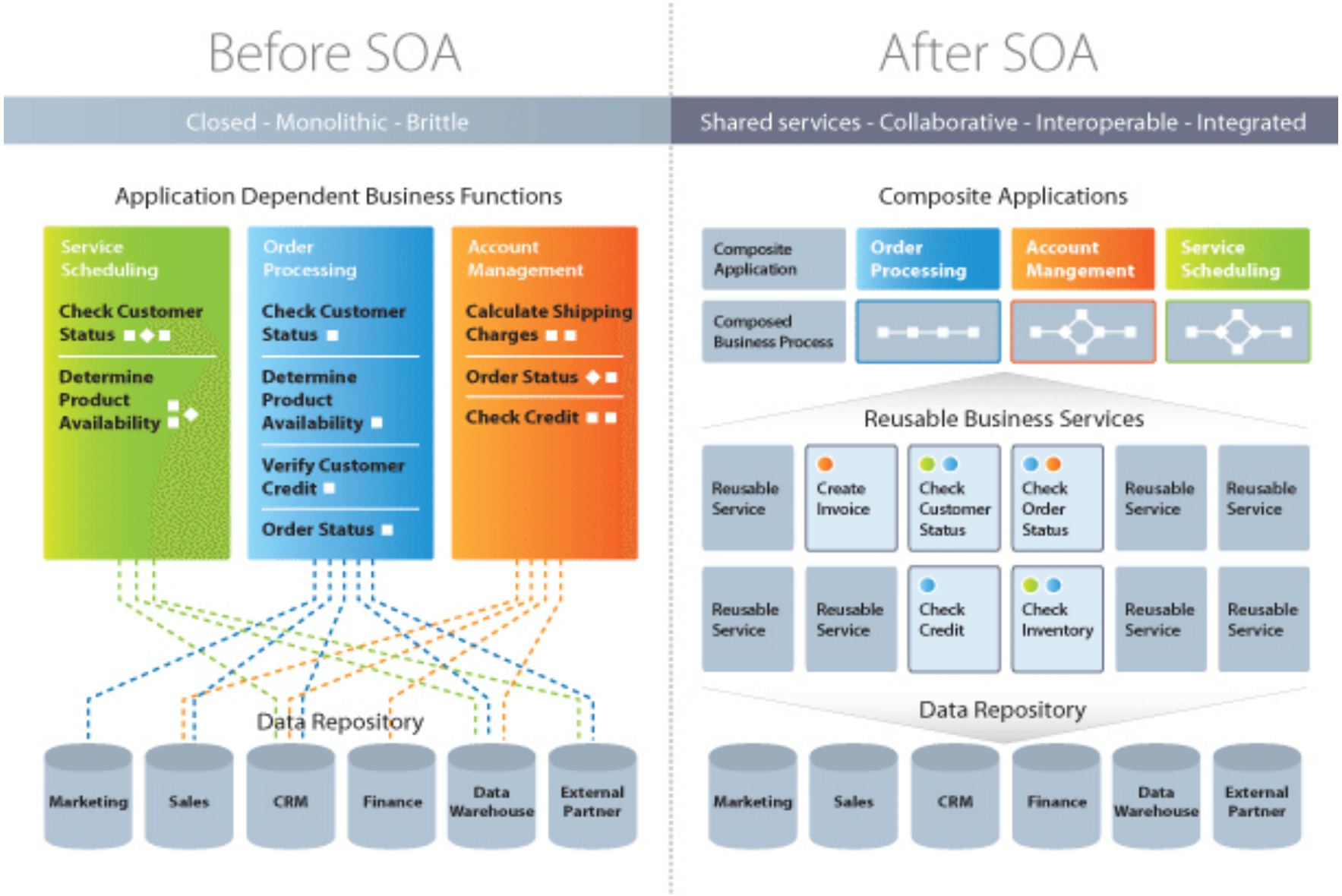
Increase cohesion: The components have stronger layer cohesion than if the view and controller were together in a single UI layer.

Reduce coupling: The communication channels between the three components are minimal.

Design for flexibility: It is usually quite easy to change the UI by changing the view, the controller, or both.

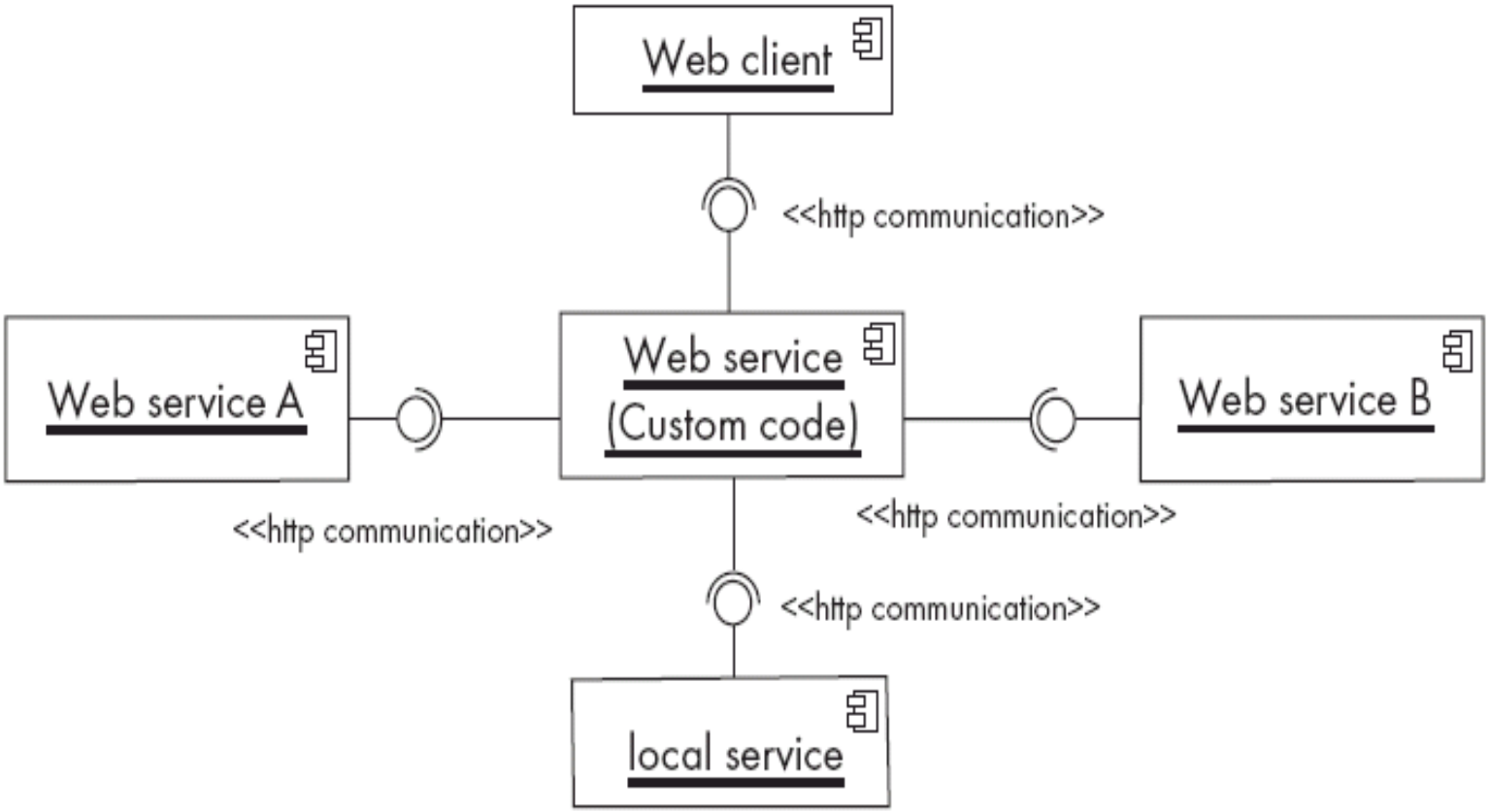
The Service-oriented architectural pattern

- This architecture organizes an application **as a collection of services that communicates using well-defined interfaces**
- Services are provided to the other components by application components, through a communication protocol over a network.
- A service is
  - a repeatable business activity with a specified outcome
  - self-contained
  - a black box for its consumers
  - may be composed of other services


Before SOA
After SOA
Closed - Monolithic - Brittle
Shared services - Collaborative - Interoperable - Integrated
Application Dependent Business Functions
Service Scheduling
Check Customer Status
Determine Product Availability
Order Processing
Check Customer Status
Determine Product Availability
Verify Customer Credit
Order Status
Account Management
Calculate Shipping Charges
Order Status
Check Credit
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner
Composite Applications
Composite Application
Order Processing
Account Mangement
Service Scheduling
Composed Business Process
Reusable Business Services
Reusable Service
Create Invoice
Check Customer Status
Check Order Status
Reusable Service
Reusable Service
Reusable Service
Reusable Service
Check Credit
Check Inventory
Reusable Service
Reusable Service
Data Repository
Marketing
Sales
CRM
Finance
Data Warehouse
External Partner

Web Services

- In the context of the Internet, the services are called *Web services*
- A web service is an application, accessible through the Internet, that can be integrated with other services to form a complete system
- The different components generally communicate with each other using open standards such as XML.


Web client
<<http communication>>
Web service A
Web service
(Custom code)
Web service B
<<http communication>>
<<http communication>>
<<http communication>>
local service

SOA

- The application is made of independently designed services.
- *Increase cohesion*: The Web services are structured as layers and generally have good functional cohesion.
- *Reduce coupling*: Web-based applications are loosely coupled built by binding together distributed components.
- *Increase reusability*: A Web service is a highly reusable component.

Introduction to Metrics

What and Why Metrics?

A **software metric** is a measure of **software** characteristics which are measurable or countable.

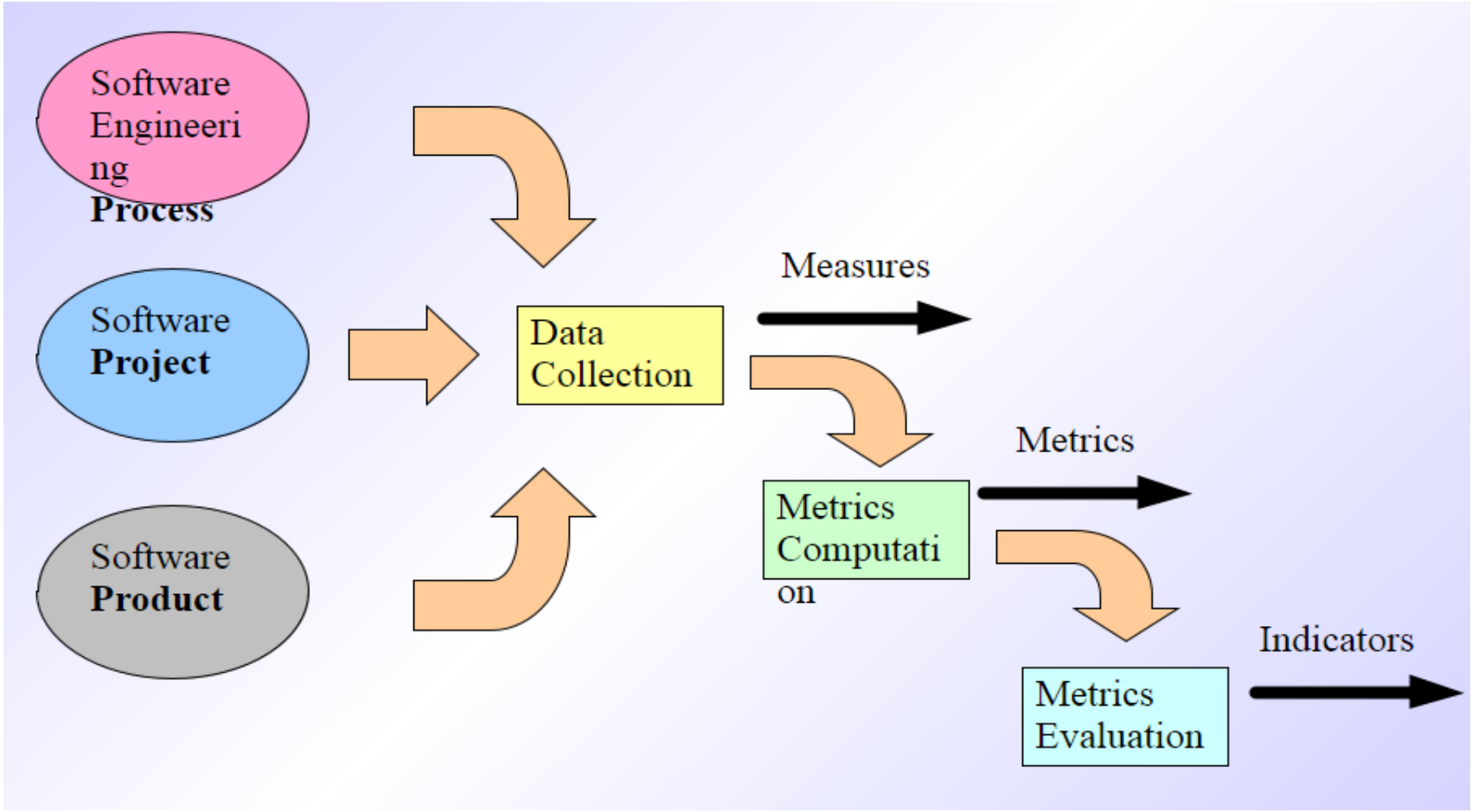
- How good is the design?
- How complex is the code?
- How much efforts will be required?
- **Makes comparisons** possible.

Software metrics are valuable for many reasons, including measuring **software** performance, **planning** work items, measuring productivity, **decision making** and many other uses.

What to be measured?

- **Characteristics of the software product:**
  - Software size and complexity
  - Software reliability and quality
  - Functionalities
  - Performance
- **Characteristics of the software project:**
  - Number of software developer
  - Staffing pattern over the life cycle of software
  - Cost and schedule
  - Productivity
- **Characteristics of software process:**
  - Methods, tools, and techniques used for software development
  - Efficiency of detection of fault

Software Metrics Baseline Process



After data is collected and metrics are computed, the metrics should be evaluated and **applied during estimation**, **technical work, project control, and process improvement**

- Metrics calculations possible over the different project phases
  - Use case oriented metrics
  - Function based metrics (Eg: Function point-FP)
  - Object oriented metrics – Design metric
  - Size oriented metrics (Eg: Lines of code-LOC)

Estimation

Importance of Estimation

- During the planning phase of a project, a first **guess about cost and time** is necessary
- Estimations are often the basis for the decision to start a project
- Estimations are the foundation for project planning and for further actions

Challenges

- Incomplete knowledge about:
  - Project scope and changes
  - Prospective resources and staffing
  - Technical and organizational environment
  - Feasibility of functional requirements
- Comparability of projects **in case of new or changing** technologies, staff, methodologies
- Learning curve problem
- **Different expectations** towards project manager.

Guiding Principles

- **Documentation of assumptions** about
  - Estimation methodology
  - Project scope, staffing, technology
- **Increasing accuracy with project phases** (Refining)
  - Example: Better estimation for implementation phase after object design is finished
- **Reviews by experienced colleagues**

Estimating Development Time

Development time often estimated by formula

Duration = Effort / People; M=PM/P

Person day: Effort of one person per working day

Person month: Effort of one person per month

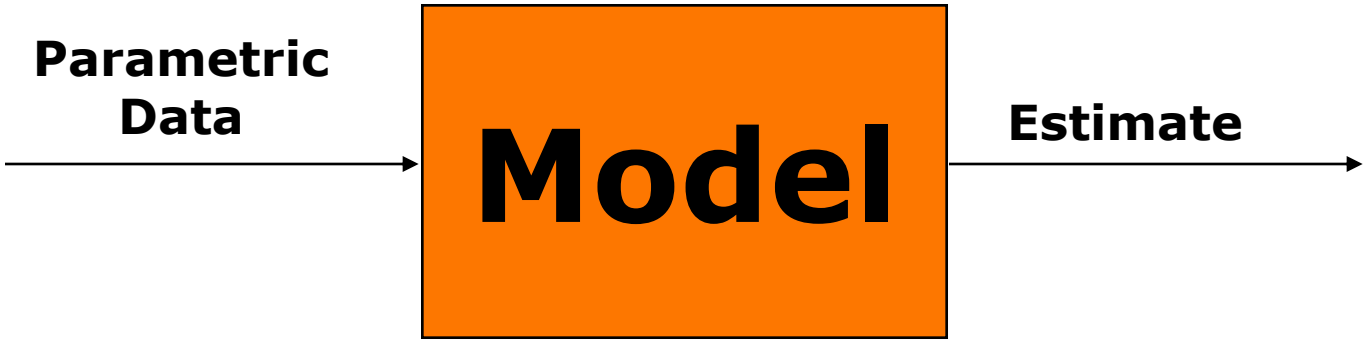
Problem with formula, because:

- A larger project team increases communication complexity which usually reduces productivity
- Therefore it is not possible to reduce duration arbitrarily by adding more people to a project

Estimating Effort

- Most difficult part during project planning
  - Many planning tasks (especially project schedule) depend on determination of effort
- Basic principle:
  - Select an estimation model (or build one first)
  - Evaluate known information: size and project data, resources, software process, system components
  - Feed this information as parametric input data into the model
  - Model converts the input into estimates: effort, schedule, performance, cycle time.

Basic Use of Estimation Models



Examples:

Data Input	Estimate
Size & Project Data	Effort & Schedule
System Model	Performance
Software Process	Cycle Time

Top-Down and Bottom-Up Estimation

- Two common approaches for estimations
  - Top-Down Approach
    - Estimate effort for the whole project
    - Breakdown to different project phases and work products
  - Bottom-Up Approach
    - Start with effort estimates for tasks on the lowest possible level
    - Aggregate the estimates until top activities are reached.

Top-Down versus Bottom-Up (cont'd)

- Top-Down Approach
  - Normally used in the planning phase when little information is available how to solve the problem
  - Based on experiences from similar projects
  - Not appropriate for project controlling (too high-level)
  - Risk add-ons usual
- Bottom-Up Approach
  - Normally used after activities are broken down the task level and estimates for the tasks are available
  - Result can be used for project controlling (detailed level)
  - Smaller risk add-ons
- Often a mixed approach with recurring estimation cycles is used.

Estimation Techniques

- Expert estimates
- Lines of code
- **Function point analysis**
- **COCOMO** I
- COCOMO II

Expert Estimates

= Guess from experienced people

- Suitable for atypical projects
- Result justification difficult
- Important when no detailed estimation can be done (due to lacking information about scope)

Lines of Code

- Traditional way for estimating application size
- **Advantage**: Easy to do
- **Disadvantages**:
  - Focus on developer's point of view
  - *No standard definition* for "Line of Code"
  - "You get what you measure": If the number of lines of code is the primary measure of productivity, programmers ignore opportunities of reuse
  - Hard to compare different language projects
  - It measures project size, not a good technique for telling how good the code is.

Function Point Analysis

- Developed by Allen Albrecht, IBM Research, 1979
- Technique to determine size of software projects
  - **Size is measured from a functional point of view**
  - Estimates are based on functional requirements
- Albrecht originally used the technique to predict effort
  - Size is usually the primary driver of development effort
- Independent of
  - Implementation language and technology
  - Development methodology
  - Capability of the project team
- Three steps: Plan the count, perform the count, estimate the effort.

Steps in Function Point Analysis

- Plan the count
  - Identify the counting boundary
  - Identify sources for counting information: software, documentation and/or expert
- Perform the count
  - Count data access functions
  - Count transaction functions
- Estimate the effort
  - Compute the unadjusted function points (UFP)
  - Compute the Value Added Factor (VAF)
  - Compute the adjusted Function Points (FP)
  - Compute the performance factor
  - Calculate the effort in person days

Function Types

Data function types

- # of internal logical files (ILF)
- # of external interface files (EIF)

Transaction function types

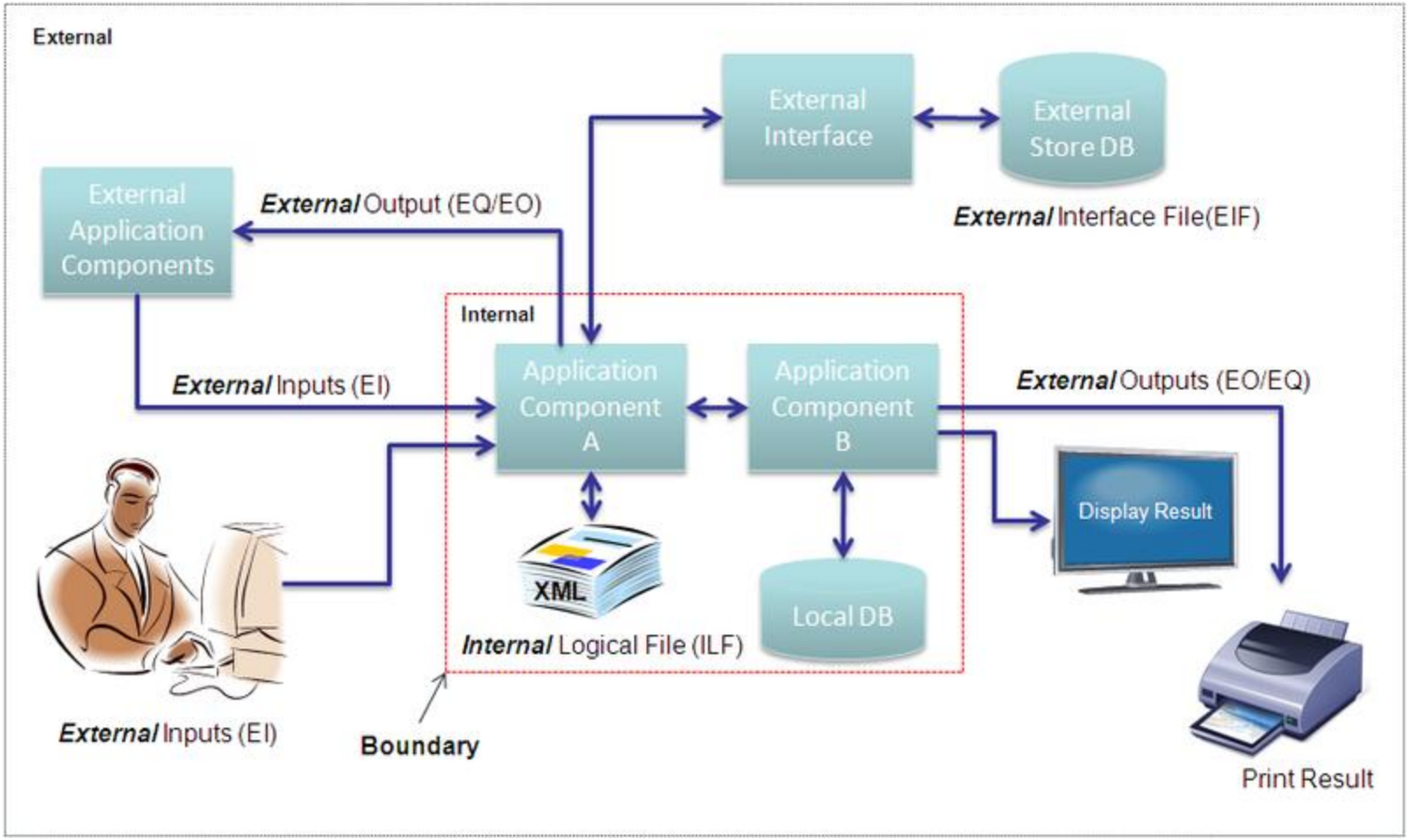
- # of external input (EI)
- # of external output (EO)
- # of external queries (EQ)

Calculate the UFP (unadjusted function points):

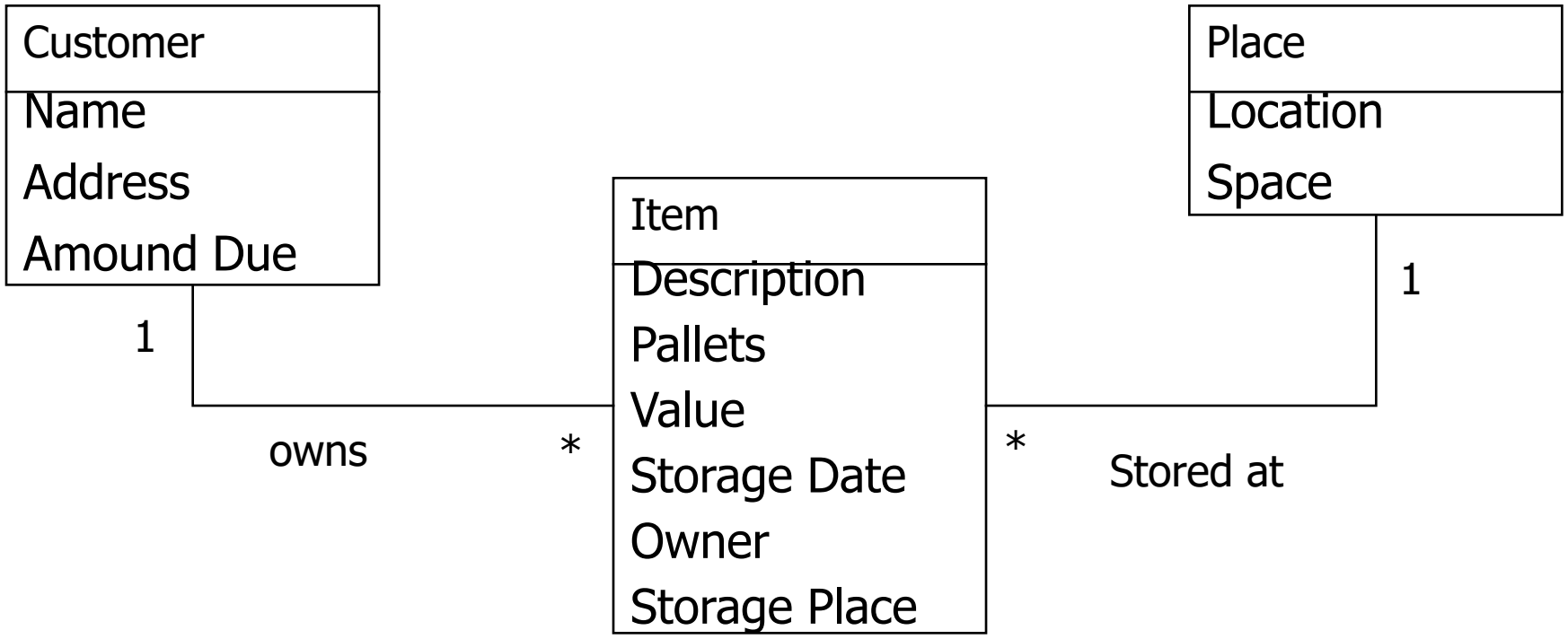
$$UFP = a \cdot EI + b \cdot EO + c \cdot EQ + d \cdot ILF + e \cdot EIF$$

a-e are weight factors (see coming slides)

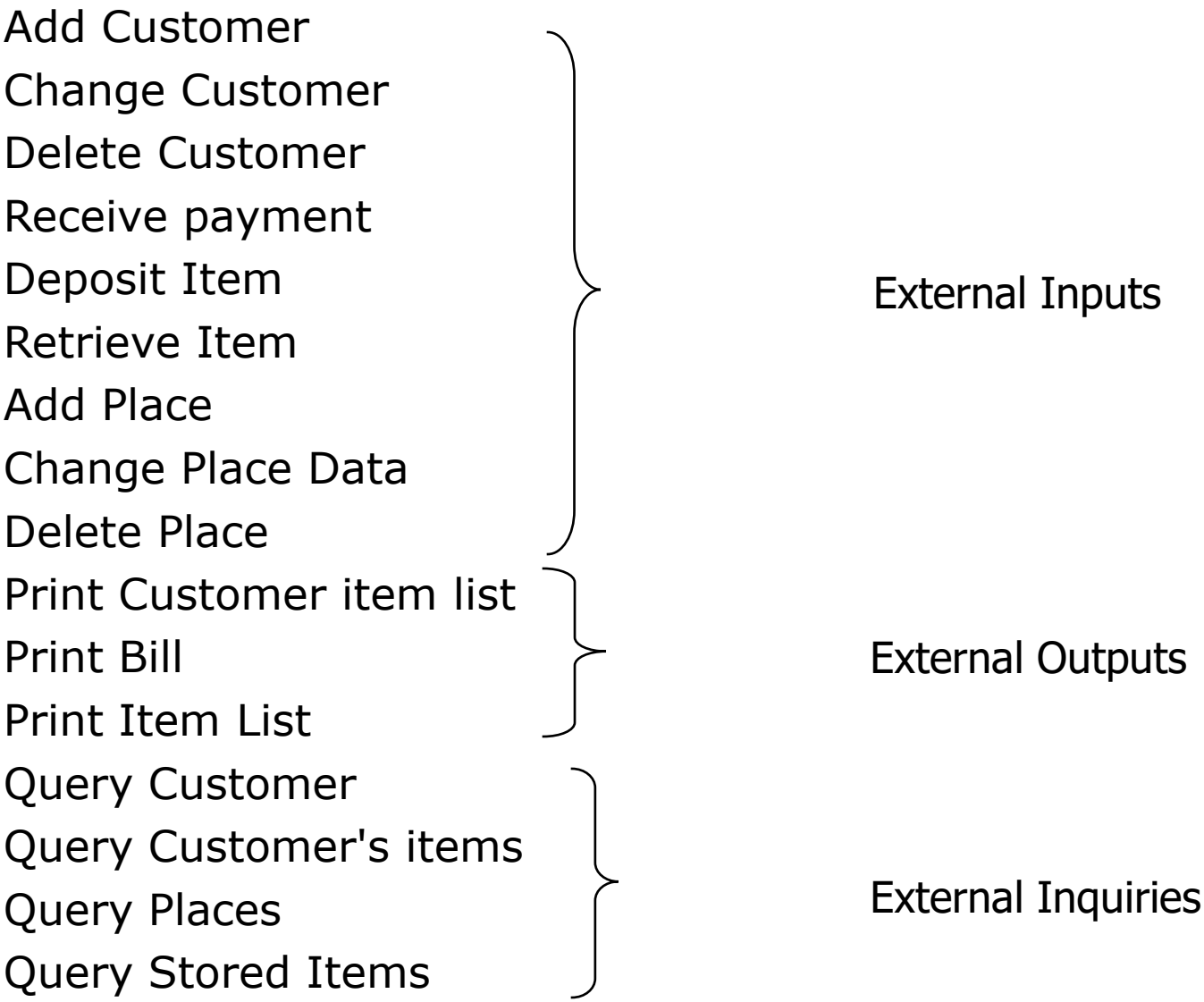
FPA components



Object Model Example



Mapping Functions to Transaction Types



Calculate the Unadjusted Function Points

Function Type	Number		Weight Factors: simple	average	complex		
External Input (EI)		x	3	4	6	=	
External Output (EO)		x	4	5	7	=	
External Queries (EQ)		x	3	4	6	=	
Internal Datasets (ILF)		x	7	10	15	=	
Interfaces (EIF)		x	5	7	10	=	

Unadjusted Function Points (UFP) =

14 General System Complexity Factors

- The unadjusted function points are adjusted with general system complexity (GSC) factors

GSC1: Reliable Backup & Recovery
GSC2: Use of Data Communication
GSC3: Use of Distributed Computing
GSC4: Performance
GSC5: Realization in heavily used configuration
GSC6: On-line data entry
GSC7: User Friendliness
GSC8: On-line data change
GSC9: Complex user interface
GSC10: Complex procedures
GSC11: Reuse
GSC12: Ease of installation
GSC13: Use at multiple sites
GSC14: Adaptability and flexibility

- Each of the GSC factors gets a value from 0 to 5.

  0-No influence, 1-Incidental Influence, 2-Moderate Influence, 3-Average Influence, 4-Significant Influence, 5-Strong Influence

There are 14 GSC factors

1. **Reliable Backup & Recovery or Operational ease**: How effective and/or automated are start-up, back up, and recovery procedures?
2. **Data communications**: How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
3. **Distributed data processing**: How are distributed data and processing functions handled?
4. **Performance:** Did the user require response time or throughput? Transaction rate : How frequently are transactions executed daily, weekly, monthly, etc.?
5. **Heavily used configuration**: - How heavily used is the current hardware platform where the application will be executed?
6. **On-line data entry** :- What percentage of the information is entered On-Line?
7. **User Friendliness**: Was the application designed for end-user efficiency?
8. **On-line data change** : How many ILF's are updated by on-line transactions?
9. **Complex user interface**: Does the user interface require extensive clicks and does it have many data entry fields?
10. **Complex procedures**: Does the application have extensive logical or mathematical processing?
11. **Reusability** : Was the application developed to meet one or many user's needs?
12. **Ease of Installation**: How difficult is conversion and installation?
13. **Use at Multiple sites:** Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
14. **Adaptability and flexibility**: Was the application specifically designed, developed, and supported to facilitate change?

Function Points: Example of a GSC Rating

GSC	Value(0-5)
Data communications	1
Distributed data processing	1
Performance	4
Heavily used configuration	0
Transaction rate	1
On-Line data entry	0
End-user efficiency	4
On-Line update	0
Complex processing	0
Reusability	3
Installation ease	4
Operational ease	4
Multiple sites	0
Adaptability and Flexibility	0
Total	**22**

Calculate the Effort

- After the GSC factors are determined, compute the Value Added Factor (VAF):

$$VAF = 0.65 + 0.01 * \sum_{i=1}^{14} GSC_i \qquad GSC_i = 0,1,...,5$$

- Function Points =
  Unadjusted Function Points * Value Added Factor
  - FP = UFP · VAF

- Performance factor
  - Effort = FP / PF

  PF = Number of function points that can be completed per day

Examples

- UFP = 18
- Sum of GSC factors = 22
- VAF = 0.87
- Adjusted FP = VAF * UFP = 0.87 * 18 ~ 16
- PF =2
- Effort = 16/2 = 8 person days

- UFP = 18
- Sum of GSC factors = 70
- VAF = 1.35
- Adjusted FP = VAF * UFP = 1.35 * 18 ~ 25
- PF = 1
- Effort = 25/1 = 25 person days

Advantages of Function Point Analysis

- **Independent of implementation language** and technology
- Estimates are based on design specification
  - Usually known before implementation tasks are known
- **Users without technical knowledge** can be integrated into the estimation process
  - Incorporation of experiences from different organizations
- **Easy to learn**
  - Limited time effort

Disadvantages of Function Point Analysis

- Complete description of functions necessary
  - Often not the case in early project stages -> especially in iterative software processes
- Only complexity of specification is estimated
  - Implementation is often more relevant for estimation
- **High uncertainty in calculating function points**:
  - Weight factors are usually deducted from past experiences (environment, used technology and tools may be out-of-date in the current project)
- Does not measure the performance of people

COCOMO (COnstructive COst MOdel)

- Developed by Barry Boehm in 1981
- Also called COCOMO I or Basic COCOMO
- There is Basic, Intermediate and detailed (COCOMO II) model
- The more complex models account for more factors, and make more accurate estimates.

- Assumptions:
  - Derivability of effort by comparing finished projects ("COCOMO database")
  - System requirements do not change during development

Calculation of Effort

- Estimate number of instructions
  - KDSI = "Kilo Delivered Source Instructions"
- Determine project complexity parameters: A, B
  - Regression analysis, matching project data to equation
- 3 levels of difficulty that characterize projects
  - Simple project ("organic mode")
  - Semi-complex project ("semidetached mode")
  - Complex project ("embedded mode")
- Calculate effort
  - Effort = $A * KDSI^B$
- Also called Basic COCOMO

Calculation of Effort in Basic COCOMO

Formula: Effort = A * $KDSI^B$

- Effort is counted in person months: 152 productive hours (8 hours per day, 19 days/month, less weekends, holidays, etc.)
- A, B are constants based on the complexity of the project

Project Complexity	A	B
Simple (Organic)	2.4	1.05
Semi-Complex (Semi-detached)	3.0	1.12
Complex (Embedded)	3.6	1.20

Calculation of Development Time

Basic formula: $T = C * Effort^{D}$

- T = Time to develop in months
- C, D = constants based on the complexity of the project
- Effort = Effort in person months (see slide before)

Project Complexity	C	D
Simple (Organic)	2.5	0.38
Semi-Complex (Semi-detached)	2.5	0.35
Complex (Embedded)	2.5	0.32

Basic COCOMO Example

Volume = 30000 LOC = 30KLOC

Project type = Simple

Effort $= 2.4 * (30)^{1.05} = 85$ PM

Development Time $= 2.5 * (85)^{0.38} = 13.5$ months

=> Avg. staffing: 85/13.5 = 6.3 persons

Compare:

Semi-detached:	135 PM	13.9 M	9.7 persons
Embedded:	213 PM	13.9 M	15.3 persons

Other COCOMO Models

- Intermediate COCOMO
  - 15 cost drivers yielding a multiplicative correction factor
  - Basic COCOMO is based on value of 1.00 for each of the cost drivers
- Detailed COCOMO
  - Multipliers depend on phase: Requirements; System Design; Detailed Design; Code and Unit Test; Integrate & Test; Maintenance

Steps in Intermediate COCOMO

- Basic COCOMO steps:
  - Estimate number of instructions
  - Determine project complexity parameters: A, B
  - Determine level of difficulty that characterizes the project
- New step:
  - Determine cost drivers
    - 15 cost drivers c1 , c2 …. c15
- Calculate effort
  - Effort = $A * KDSI^{B} * c1 * c2 \ldots. * c15$

Calculation of Effort in Intermediate COCOMO

Basic formula:

Effort = $A * KDSI^{B} * c_1 * c_1 \ldots * c_{15}$

Effort is measured in PM (person months, 152 productive hours (8 hours per day, 19 days/month, less weekends, holidays, etc.)

A, B are constants based on the complexity of the project

Project Complexity	A	B
Simple(Organic)	3.2	1.05
Semi-Complex(Semi-detached)	3.0	1.12
Complex(Embedded)	2.8	1.20

Intermediate COCOMO: 15 Cost drivers

- Product Attributes
  - Required reliability
  - Database size
  - Product complexity
- Computer Attributes
  - Execution Time constraint
  - Main storage constraint
  - Virtual Storage volatility
  - Turnaround time
- Personnel Attributes
  - Analyst capability
  - Applications experience
  - Programmer capability
  - Virtual machine experience
  - Language experience
- Project Attributes
  - Use of modern programming practices
  - Use of software tools
  - Required development schedule

- Rated on a qualitative scale between “very low” and “extra high”
- Associated values are multiplied with each other.

Cocomo: Example of Cost Driver Rating

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Required software reliability	0.75	0.88	1.00	1.15	1.40	-
Database size	-	0.94	1.00	1.08	1.16	-
Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
Execution Time Constraint	-	-	1.00	1.11	1.30	1.66
Main storage constraint	-	-	1.00	1.06	1.21	1.56
Virtual Storage volatility	-	0.87	1.00	1.15	1.30	-
Computer turn around time	-	0.87	1.00	1.07	1.15	-
Analyst capability	1.46	1.19	1.00	0.86	0.71	-
Applications experience	1.29	1.13	1.00	0.91	0.82	-
Programmer Capability	1.42	1.17	1.00	0.86	0.70	-
Virtual machine experience	1.21	1.10	1.00	0.90	-	-
Prog. language experience	1.14	1.07	1.00	0.95	-	-
Use of modern Practices	1.24	1.10	1.00	0.91	0.82	-
Use of software tools	1.24	1.10	1.00	0.91	0.83	-
Required schedule	1.23	1.08	1.00	1.04	1.10	-

Example

- For a given project (semi-detached) was estimated with a size of 300 KLOC. Calculate the Effort, Scheduled time for development by considering developer having very high application experience and very low experience in programming.

  Ans:

Given the estimated size of the project is: 300 KLOC

Developer having very highly application experience: 0.82 (as per above table)

Developer having very low experience in programming: 1.14(as per above table)

Example

- EAF = 0.82*1.14 = 0.9348
  Effort (E) = $A*(KLOC)^{B}$ *EAF =
  $3.0*(300)^{1.12}$ *0.9348 = 1668.07 PM
  Scheduled Time (D) = $c*(E)^{d}$ =
  $2.5*(1668.07)^{0.35}$ = 33.55 Months(M)

COCOMO II

- Revision of COCOMO I in 1997
- Provides three models of increasing detail
  - **Application Composition Model**
    - Estimates for prototypes based on GUI builder tools and existing components
  - **Early Design Model**
    - Estimates before software architecture is defined
    - For system design phase, closest to original COCOMO, uses function points as size estimation
  - **Post Architecture Model**
    - Estimates once architecture is defined
    - For actual development phase and maintenance; Uses FPs or LOC as size measure
- Estimator selects one of the three models based on current state of the project.

COCOMO II (cont'd)

- Targeted for iterative software lifecycle models
  - Boehm's spiral model
  - COCOMO I assumed a waterfall model
    - 30% design; 30% coding; 40% integration and test
- COCOMO II includes new costs drivers to deal with
  - Team experience
  - Developer skills
  - Distributed development

Advantages of COCOMO

- Appropriate for a quick, high-level estimation of project costs
- **Fair results with smaller projects** in a well known development environment
  - Assumes comparison with past projects is possible
- **Covers all development activities** (from analysis to testing)

Principles of effective cost estimation

1. Divide and conquer
2. Include all activities when making estimates
3. Base your estimates on past experience combined with what you can observe of the current project
4. Be sure to account for differences when extrapolating from other projects
5. Anticipate the worst case and plan for contingencies
6. Combine multiple independent estimates
7. Revise and refine estimates as work progresses

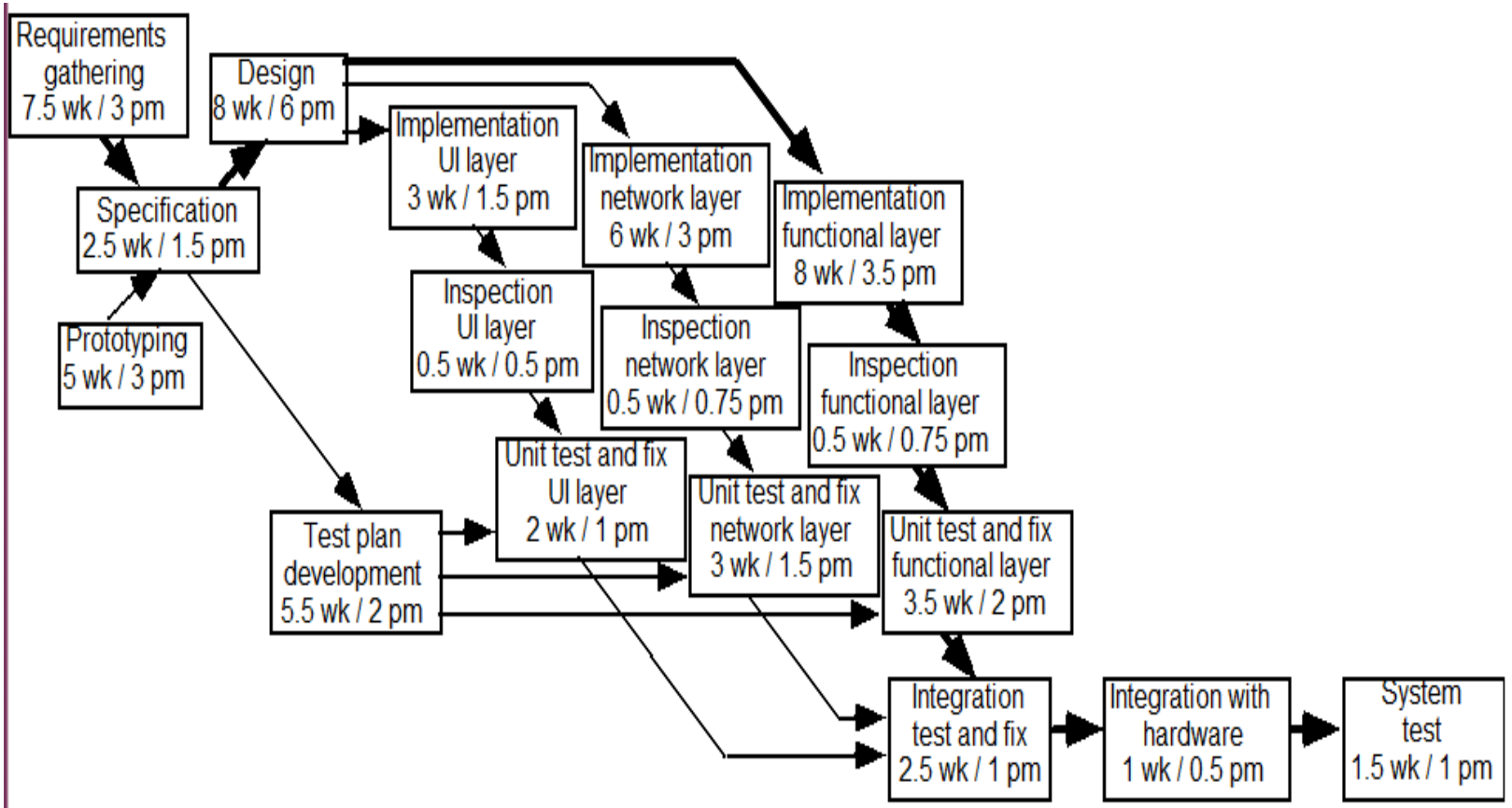
Project Scheduling and Tracking

- *Scheduling* is the process of deciding:
  - In what sequence a set of activities will be performed.
  - When they should start and be completed.
  - Two important scheduling techniques
    - PERT chart
    - Gantt chart
- *Tracking* is the process of determining how well you are sticking to the cost estimate and schedule.
  - Important tracking technique
    - Earned value charts

PERT charts (Program Evaluation Review Technique)

- A PERT chart shows the sequence in which tasks must be completed.
  - In each node of a PERT chart, you typically show the elapsed time and effort estimates.
  - One of the most important uses of a PERT chart is to determine the ***critical path***.
    - The ***critical path*** indicates the minimum time in which it is possible to complete the project.
    - It is computed by searching for the path through the chart that has the greatest cumulative elapsed time and no idle time.

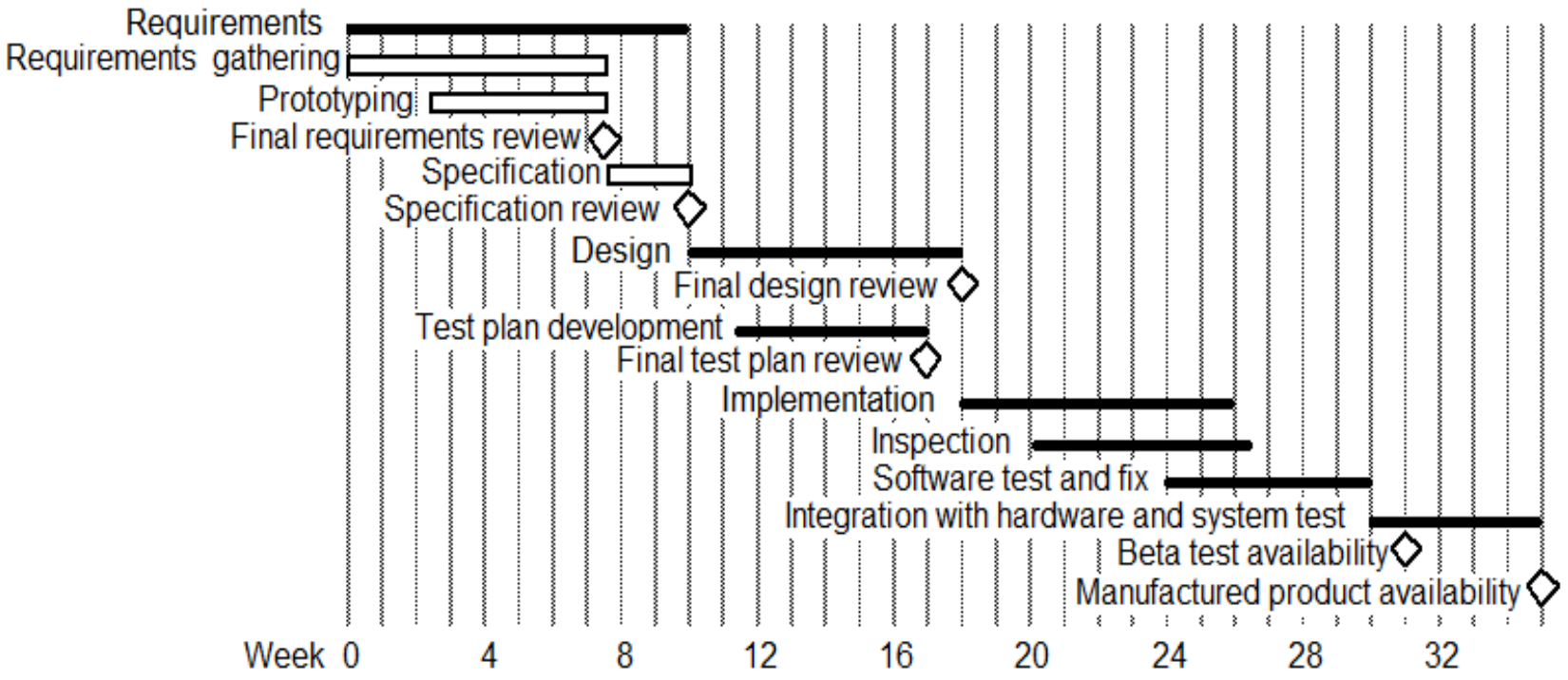
Example of a PERT chart



Gantt charts

- A Gantt chart is used to graphically present the start and end dates of each software engineering task
  - One axis shows time.
  - The other axis shows the activities that will be performed.
  - The black bars are the top-level tasks.
  - The white bars are subtasks
  - The diamonds are *milestones:*
    - Important deadline dates, at which specific events may occur

Example of a Gantt chart



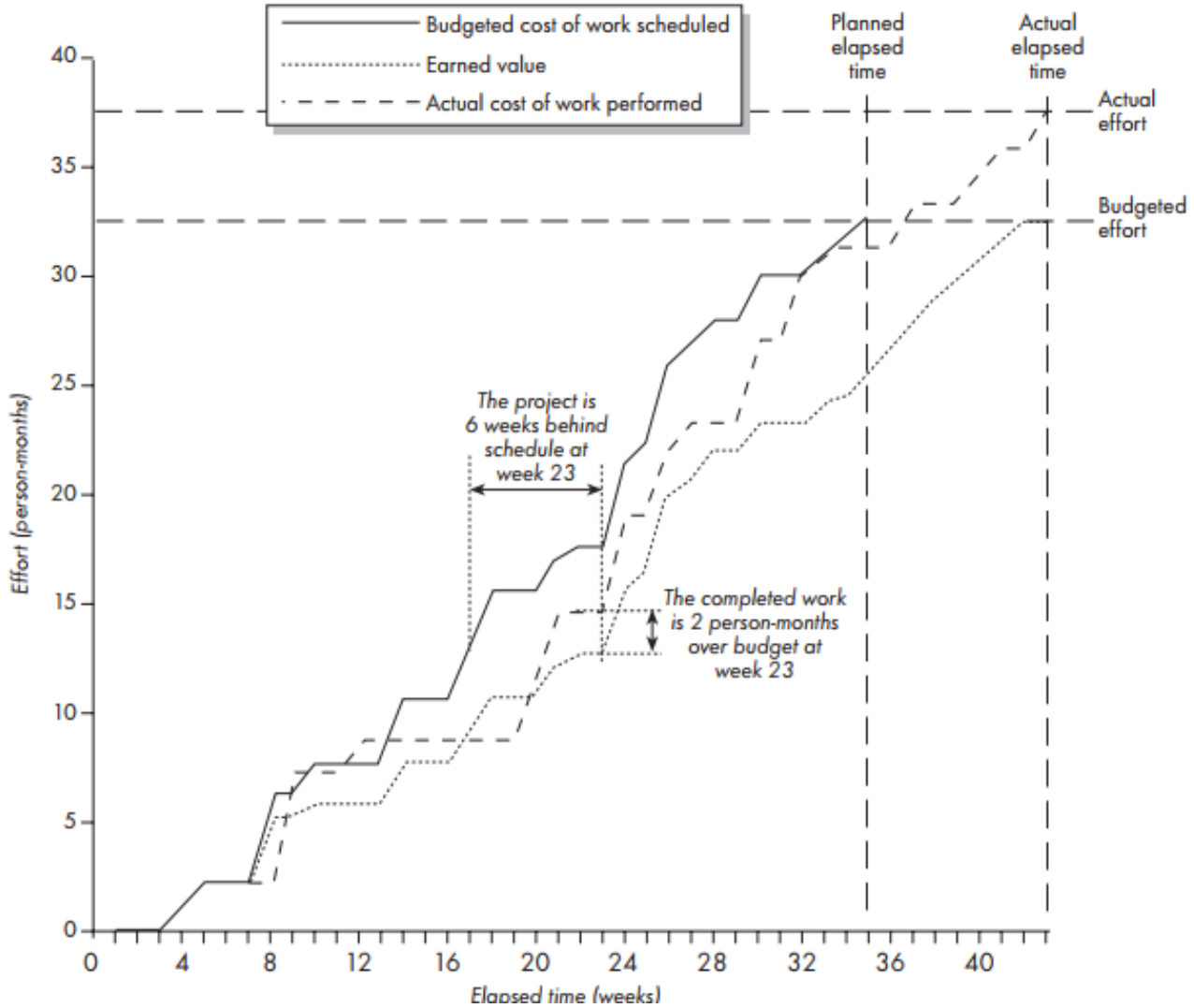
Earned value

- *Earned value* is the amount of work completed, measured according to the *budgeted* effort that the work was supposed to consume.
- It is also called the *budgeted cost of work performed*.

Earned value charts

- An earned value chart has three curves:
  - The budgeted cost of the work scheduled(planned value)
  - The earned value.
  - The actual cost of the work performed so far.

Example of an earned value chart



- Assignment Quiz for 7 Marks
  - NEXT WEEK on course slot
  - Topic - Estimation