SOFTWARE ENGINEERING

Week 4
Software Project Management – 1

Prof. Dr. Muhittin GÖKMEN Yard. Doç. Dr. A. Cüneyd TANTUĞ Araş. Gör. Dr. Tolga OVATMAN
Istanbul Technical University
Computer Engineering Department

Agenda

1. Project Management Concepts
2. Estimation
3. Process Metrics
4. Planning and Scheduling

1. Project Management Concepts
2. Estimation
3. Process Metrics
4. Planning and Scheduling

Project Management Concepts

4.1

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 developing 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.

Success Criteria

- Deliver the software to the customer at the agreed time.
- Keep overall costs within budget.
- Deliver software that meets the customer's expectations.
- Maintain a happy and well-functioning development team.

4P's

1. People
   the most important element of a successful project

2. Product
   the software to be built

3. Process
   the set of framework activities and software engineering tasks to get the job done

4. Project
   all work required to make the product a reality

Stakeholders

- *Senior managers* who define the business issues that often have significant influence on the project.
- *Project (technical) managers* who must plan, motivate, organize, and control the practitioners who do software work.
- *Practitioners* who deliver the technical skills that are necessary to engineer a product or application.
- *Customers* who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
- *End-users* who interact with the software once it is released for production use.

Software Teams

- *The following factors must be considered when selecting a software project team structure*
  - the difficulty of the problem to be solved
  - the size of the resultant program(s) in lines of code or function points
  - the time that the team will stay together (team lifetime)
  - the degree to which the problem can be modularized
  - the required quality and reliability of the system to be built
  - the rigidity of the delivery date
  - the degree of sociability (communication) required for the project

Avoid Team "Toxicity"

- A frenzied work atmosphere in which team members waste energy and lose focus on the objectives of the work to be performed.
- High frustration caused by personal, business, or technological factors that cause friction among team members.
- "Fragmented or poorly coordinated procedures" or a poorly defined or improperly chosen process model that becomes a roadblock to accomplishment.
- Unclear definition of roles resulting in a lack of accountability and resultant finger-pointing.
- "Continuous and repeated exposure to failure" that leads to a loss of confidence and a lowering of morale.

Team Organizations

- Small software engineering groups are usually organised informally without a rigid structure.
- For large projects, there may be a hierarchical structure where different groups are responsible for different sub-projects.
  - Decentralized-control team
  - Centralized-control team
  - Mixed-control team

Decentralized-Control Team

- Decisions are made thru consensus
- Review each other's work
- Suitable for long-term projects
- Not suitable for large teams
- Decentralized control is best when communication among engineers is necessary

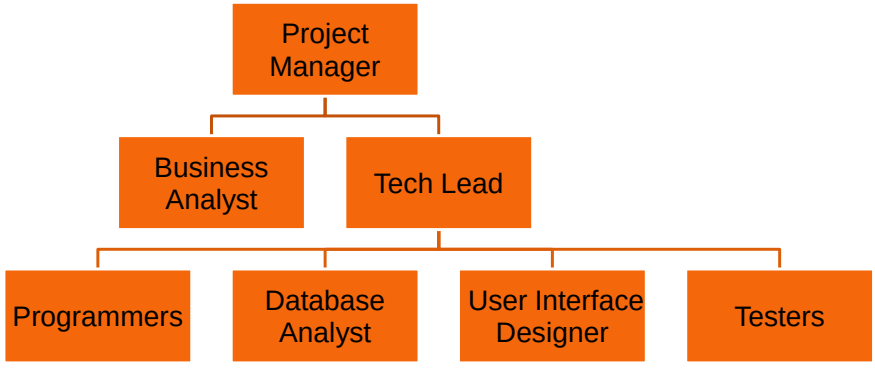
Centralized-Control Team

- Chief programmer is responsible for the design and all the technical details of the project
- The chief programmer may be overloaded
- Centralized control is best when the speed of development is the most important goal and the problem is well understood

Mixed-Control Team

- Hierarchical team
- Combination of centralized and decentralized teams
- Attempts to have advantages of both

Example Team Organization



Agile Teams

- Team members must have trust in one another.
- The distribution of skills must be appropriate to the problem.
- Mavericks may have to be excluded from the team, if team cohesiveness is to be maintained.
- Team is “self-organizing”
  - An adaptive team structure
  - Uses elements of Constantine’s random, open, and synchronous paradigms
  - Significant autonomy

The Project

- Projects get into trouble when …
  - Software people don't understand their customer's needs.
  - The product scope is poorly defined.
  - Changes are managed poorly.
  - The chosen technology changes.
  - Business needs change [or are ill-defined].
  - Deadlines are unrealistic.
  - Users are resistant.
  - Sponsorship is lost [or was never properly obtained].
  - The project team lacks people with appropriate skills.
  - Managers [and practitioners] avoid best practices and lessons learned.

Common-Sense Approach to Projects

- *Start on the right foot.* This is accomplished by working hard (very hard) to understand the problem that is to be solved and then setting realistic objectives and expectations.
- *Maintain momentum. The* project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of the team's way.
- *Track progress.* For a software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved (using formal technical reviews) as part of a quality assurance activity.
- *Make smart decisions.* In essence, the decisions of the project manager and the software team should be to "keep it simple."
- *Conduct a postmortem analysis.* Establish a consistent mechanism for extracting lessons learned for each project.

Critical Practices

- Formal risk management
- Empirical cost and schedule estimation
- Metrics-based project management
- Defect tracking against quality targets
- People aware project management

1. Project Management Concepts
2. Estimation
3. Process Metrics
4. Planning and Scheduling

Estimation

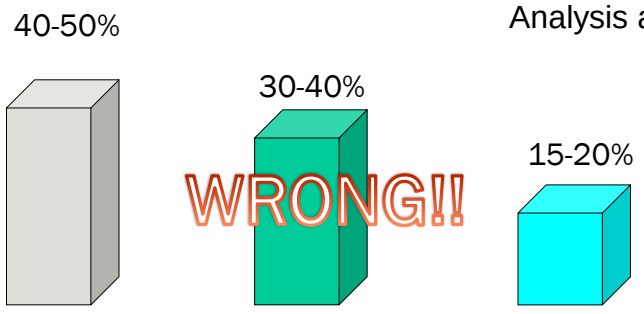
4.2

Estimation

- Organizations need to make software effort and cost estimates.
- Estimation of resources, cost, and schedule for a software engineering effort requires
  - experience
  - access to good historical information (metrics
  - the courage to commit to quantitative predictions when qualitative information is all that exists
- Estimation carries inherent risk and this risk leads to uncertainty.
- There is no simple way to make an accurate estimate of the effort required to develop a software system.
- Initial estimates are usually based on inadequate information about user requirements.

Typical Effort Distribution

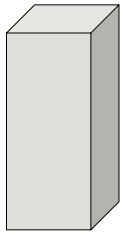
Coding
Testing
Analysis and Design



Typical Effort Distribution

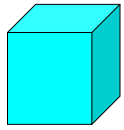
Analysis and Design

40-50%



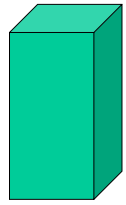
Coding

15-20%

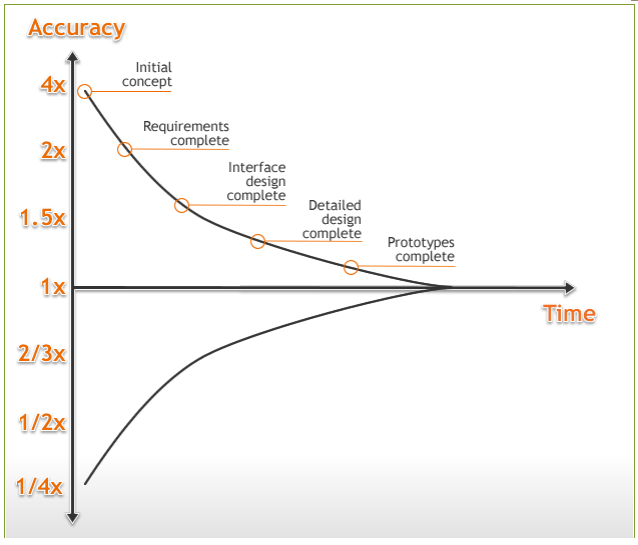


Testing

30-40%



Estimating the uncertainty!



Estimation Techniques

1. Estimation by analogy
   The cost of a new project is estimated by analogy with similar completed projects.
2. Expert judgement
   Several experts on the proposed software application domain are consulted. They each estimate the project cost. The estimation process iterates until an agreed estimate is reached.
3. Algorithmic cost modelling
   A model based on historical cost information that relates some software metric (usually its size) to the project cost is used. COCOMO is an example of algorithmic cost modelling.
4. Parkinson's Law
   Parkinson's Law states that work expands to fill the time available. The cost is determined by available resources rather than by objective assessment.
5. Pricing to win
   The estimated effort depends on the customer's budget and not on the software functionality.

Algorithmic Cost Modelling

- Cost (i.e. Effort) is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:

  $$\text{Effort} = A * (\text{Size})^E * EAF$$

  where
  - A is an organisation-dependent constant,
  - E reflects the disproportionate effort for large projects,
  - EAF (Effort Adjustment Factor) is a multiplier reflecting product, process and people attributes.

- The most commonly used product attribute for cost estimation is **code size**.
- Most models are similar but they use different values for A, E and EAF.

Function Points

- Function Point is a measurement estimation method for a software from the functionality perspective.
  - Functionality as viewed from the user's perspective.
- This estimation method is based on a empirical model which is mainly independent of programming language.
- Function Points can be used for several purposes:
  - Estimating the FP of a new planned software.
  - Assessing the retail value of an existing software.
- Developed by Allan J. Albrecht, and standardized by the "International Function Point User Group". (www.ifpug.org)

FP – Project Types

Project Type	FP Purpose
Development Project	Measures the functions that will be provided to the users in a new application.
Enhancement Project	Measures the modifications to an existing application.
Application Assesment	Measures the functionality provided to users in an existing application.

FP – Counting Steps

1. Count Data Functions and Transactional Functions
2. Calculate Unadjusted Function Point
3. Calculate Value Adjustment Factor (VAF)
4. Calculate Adjusted Function Point

FP - Equations

$$VAF = \left(\sum_{i=1}^{14} GSC_i * 0.01 \right) + 0.65$$

Adjusted FP = (Unadjusted FP) * VAF

VAF: Value Adjustment Factor

GSC: General System Characteristic factors

FP - Value Adjustment Factor (VAF)

- VAF consists of 14 General System Characteristics (GSC) such as data communications, response times, end user efficiency, multiple sites, flexibility,etc.
- Each GSC can be an integer value between 0 and 5.
- Min VAF = (14*0)*0.01 + 0.65 = 0.65
- Max VAF = (14*5)*0.01 + 0.65 = 1.35
- The overall effect of VAF can vary in range from 0.65 (when all GSCs are low) to 1.35 (when all GSCs are high), so its overall adjustment effect could be **± 35 %.**
- We will study the GSCs and VAF later.

FP – Calculating Unadjusted Function Points

Type of Component	Complexity of Components			
	Low	Average	High	Total
EI	☐ x 3	☐ x 4	☐ x 6	= ☐
EO	☐ x 4	☐ x 5	☐ x 7	= ☐
EQ	☐ x 3	☐ x 4	☐ x 6	= ☐
ILF	☐ x 7	☐ x10	☐ x15	= ☐
EIF	☐ x 5	☐ x 7	☐ x10	= ☐
			Unadjusted Function Points	= ☐

FP – Standard Functions

- In counting FPs there are five standard “functions" that you count.

Data Functions:

- Internal Logical Files (ILF)
- External Interface Files (EIF)

Transactional Functions:

- External Inputs (EI)
- External Outputs (EO)
- External Inquiries (EQ)

FP – Data Functions

- **ILF:**
  - Files controlled by the program.
  - Each data file (or database table) is counted.
  - Examples
    - ILF refers to logical group of data files maintained by the application such as **Employee file.**
    - Note the inside application data is updated and not any external data.
- **EIF:**
  - Files controlled by other programs.
  - All machine readable interfaces (import/export data file) that are used to transmit information to another system are counted.
  - Examples
    - EIF refers to logical group of data referenced but not maintained internally such as an **Currency file.**

FP - Transactional Functions

EI:

- Each user input (screens, forms, dialog boxes, controls etc.) that provides distinct data to the software is counted.
- Individual data items within a data-entry screen are not counted separately.
- Inputs should be distinguished from inquiries, which are counted separately.

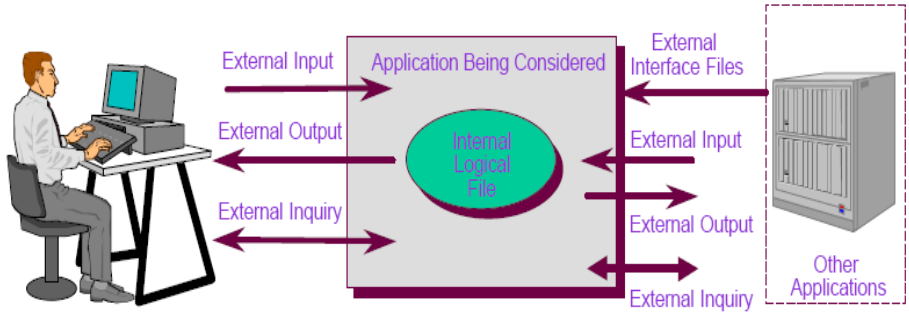
EO:

- Each user output that provides information to the user is counted.
- In this context, output refers to reports, screens, graphs, error messages, etc.
- Individual data items within a report are not counted separately.

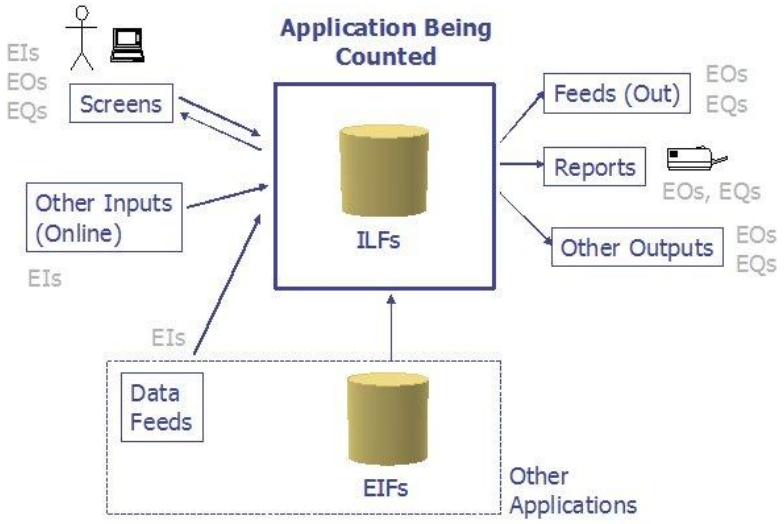
EQ:

- An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output.
- Each distinct inquiry is counted.

FP – System Example 1



FP – System Example 2



FP – Logical File (ILF) example

- ILF refers to logical group of data files maintained by the application such as **Employee file**.
- Note the inside application data is updated and not any external data.

FP – External Interface File (EIF) example

- EIF refers to logical group of data referenced but not maintained internally such as an **Currency file.**

37

FP – External Input (EI) examples

- The basic EI is from **user screens (for data entry / editing).**

- Users should have interface through which they can maintain the data files in ILF through **Add, Delete, Update** menu selections (transactional functions).

  – Passing control data such as **menu selections** into the application is considered as EI.

38

FP – External Output (EO) examples

• EO usually refers to **reports** which contain **derived data** from the internal files (ILF) such as calculated totals**.**
  – Formatted data sent out of application with added value.
  – For example, list of students and the calculated grade average in a class.

• EO can also refer to **screens** which contain the followings:
  – Displaying derived data from the internal files (ILF).
  – Prefilling a listbox with **hardcoded data** in the program.

• Outputs sent to external systems are EO.
  – For example, your application generates CSV (comma separated values) files.
  – These files are then used by some external application to update the external application tables (EIF).

FP – External Query (EQ) examples

• EQ functions will be mainly **reports** which **do not contain derived data**.

• Formatted data sent out of application without added value. (For example, only the list of students in a class)

• Reports may have input criteria, so that can be another EQ.

• Also **search screens** are EQ.

• Prefilling a listbox from ILF is considered as EQ, because this is an **implied inquiry**.

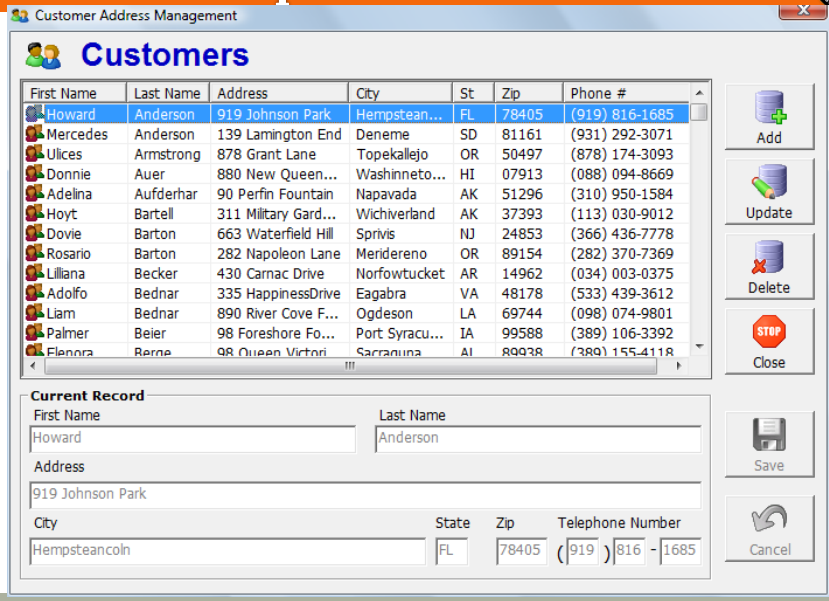
• Note EQ functions don't update any ILF or EIF. They only fetch data for display.

FP Example : "Customer" Application

- In this simple example, we will evaluate a Customer GUI (Graphical User Interface) application.
- The database has only one table, which contains customer information.
- The GUI program will allow the user add, delete, and update the records.

41

FP Example :"Customer" Screen



42

FP Example : Counting the Functions (1)

- There is 1 Internal Logical File
  1) Customer table

Customer : Table

	Field Name	Data Type
🔑	CustID	Number
	LastName	Text
	FirstName	Text
	Address	Text
	City	Text
	State	Text
	Zip	Text
	PhoneNumber	Text

FP Example : Counting the Functions (2)

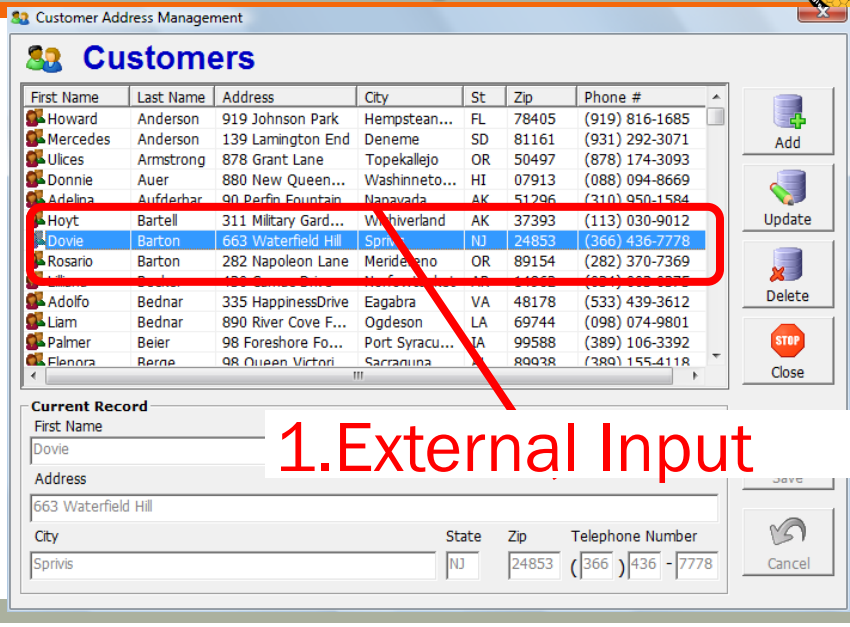
- There are 3 External Inputs
  1) "Customer selection" area in the screen for selecting a customer
  2) "Current record" area in the screen for entering / editing customer data
  3) All transaction buttons (ADD, UPDATE, DELETE, etc.)

FP Example : Counting the Functions (3)

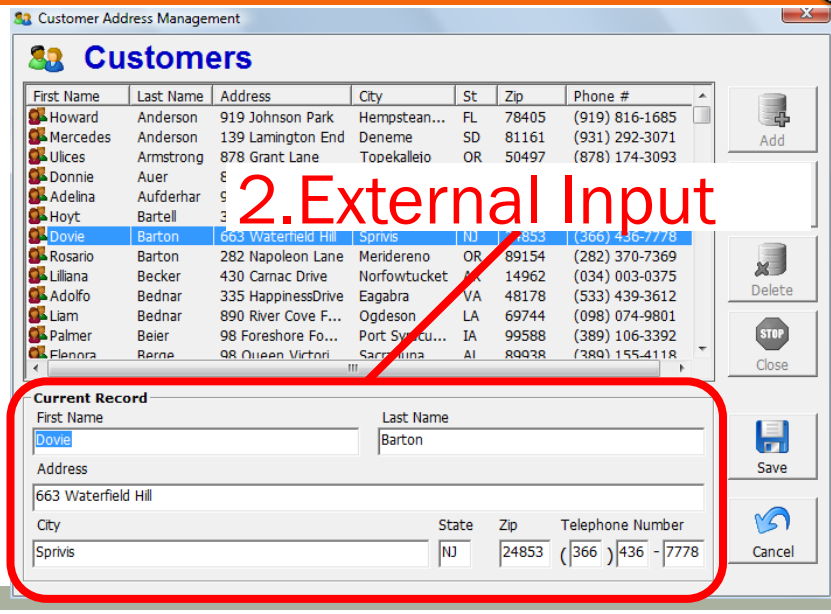
- There are 2 External Outputs
  1) The output listbox of all customer names and addresses (always read only)
  2) "Current record" area in the screen for displaying currently selected customer data (read only in display mode)

- There is 1 External Query
  1) The confirmation dialog boxes for "Delete" and "Save" buttons

FP Example : EI

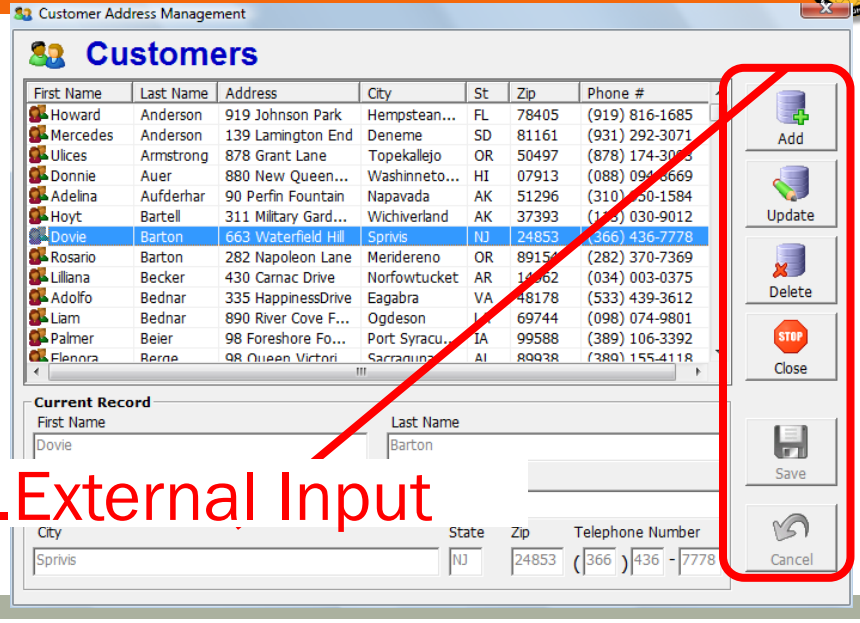


FP Example : EI



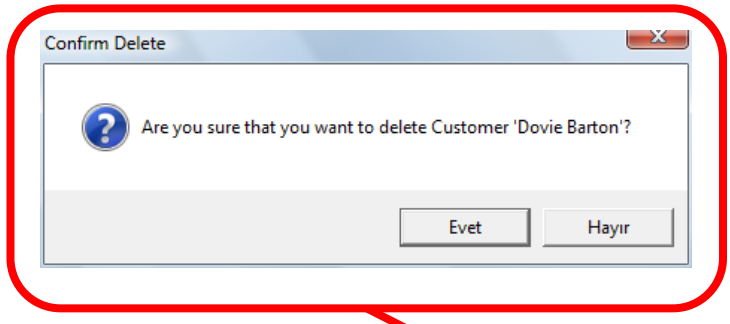
2.External Input

FP Example : EI



3.External Input

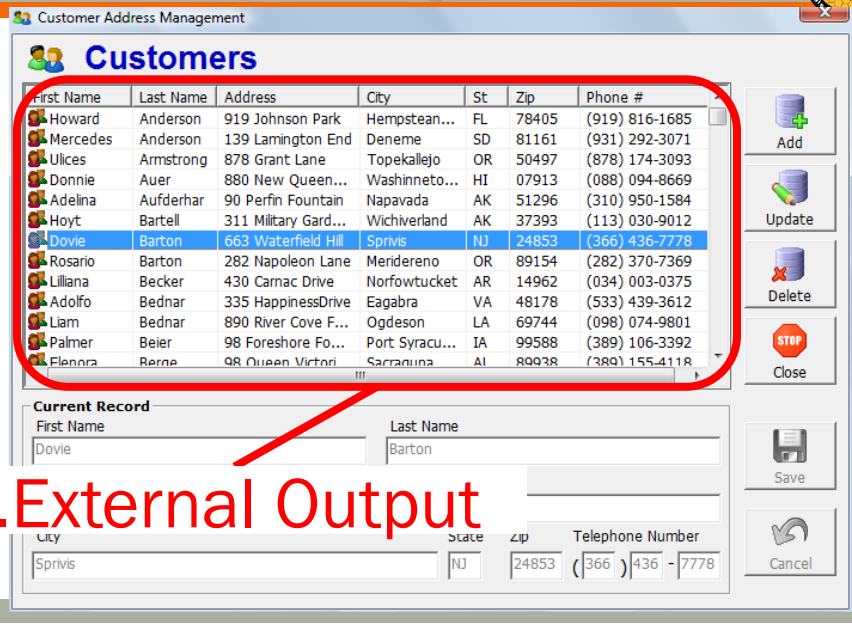
FP Example : EQ



External Query

49

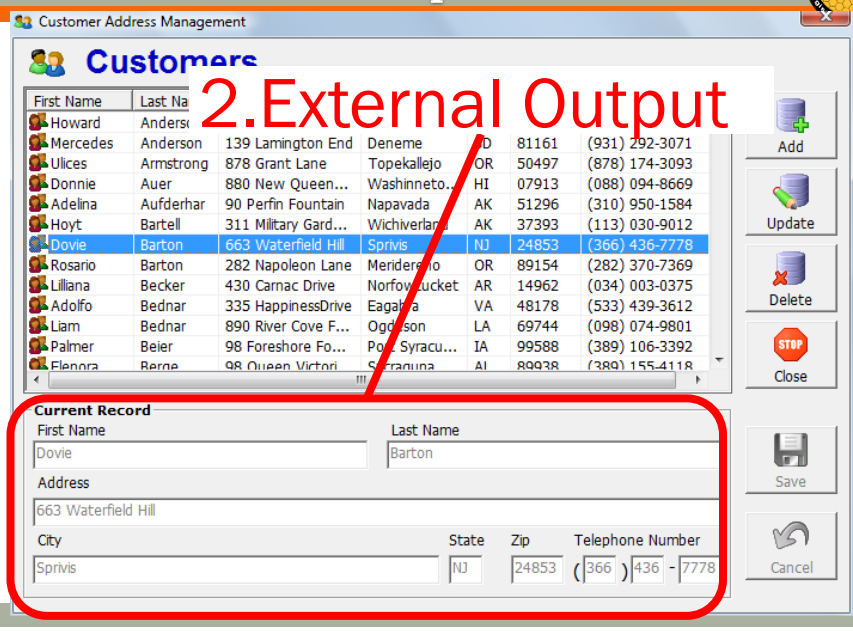
FP Example : EO



1.External Output

50

FP Example : EO



Unadjusted Function Points

Type of Component	Complexity of Components			
	Low	Average	High	Total
EI		3 x4		= 12
EO	2 x4			= 8
EQ	1 x3			= 3
ILF	1 x7			= 7

Unadjusted Function Points = 30

FP - General System Characteristics (GSC)

- This is a very important part in Function Points.
- GSC factors can affect the software a lot and also the cost of it.
- GSC gives us something called as VAF (Value Added Factor).
- There are 14 GSC points considered to come out with VAF.
- Each GSC can have a rating between 0 and 5.

FP - General System Characteristics

1. Data Communication
2. Distributed data processing
3. Performance
4. Heavily used configuration
5. Transaction rate
6. Online data entry
7. End user efficiency
8. Online update
9. Complex processing
10. Reusability
11. Installation ease
12. Operational ease
13. Multiple sites
14. Facilitate change

Description	Rating
No influence	0
Incidental	1
Moderate	2
Average	3
Significant	4
Essential	5

FP - GSC Definitions (1)

1. **Data communications:** How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
2. **Distributed data processing:** How are distributed data and processing functions handled?
3. **Performance:** Did the user require response time or throughput?
4. **Heavily used configuration:** How heavily used is the current hardware platform where the application will be executed?
5. **Transaction rate:** How frequently are transactions executed; daily, weekly, monthly, etc.?
6. **On-Line data entry:** What percentage of the information is entered On-Line?
7. **End-user efficiency:** Was the application designed for end-user efficiency?

FP - GSC Definitions (2)

8. **On-Line update:** How many ILFs are updated by On-Line transaction?
9. **Complex processing:** Does the application have extensive logical or mathematical processing?
10. **Reusability:** Was the application developed to meet one or many user's needs?
11. **Installation ease:** How difficult is conversion and installation?
12. **Operational ease:** How effective and/or automated are start-up, back up, and recovery procedures?
13. **Multiple sites:** Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
14. **Facilitate change:** Was the application specifically designed, developed, and supported to facilitate change?

FP - GSCs for "Customer" Example

GSC	Value (0-5)
1. Data communications	0
2. Distributed data processing	0
3. Performance	3
4. Heavily used configuration	3
5. Transaction rate	4
6. On-Line data entry	5
7. End-user efficiency	4
8. On-Line update	1
9. Complex processing	0
10. Reusability	0
11. Installation ease	0
12. Operational ease	2
13. Multiple sites	0
14. Facilitate change	0
Total =	22

FP - Adjusted FP for "Customer" Example

VAF = 0.65 + (Sum of all GSC factors) * 0.01)
= 0.65 + (22 * 0.01)
= 0.87

Adjusted FP = VAF * (Total Unadjusted FP)
= 0.87 * 30
≈ 26

FP - Approximate Estimation of Size and Effort

- Assume the program will be implemented in Visual Basic, which has a standard rate of 50 LOC/FP.
  - The project will be approximately 50 * 26 ≈ 1300 lines of code.

- Assume a programmer works for 5 FP per day.
  - The project will take approximately 26 / 5 ≈ 5 days.

59

FP - Actual Lines of Code in Visual Basic

Module Name	Total Lines	Effective Coded Lines
frmCustMaint.frm	989	400
modGeneral.bas	65	60
modValidate.bas	246	240
TOTALS =	1300	700

60

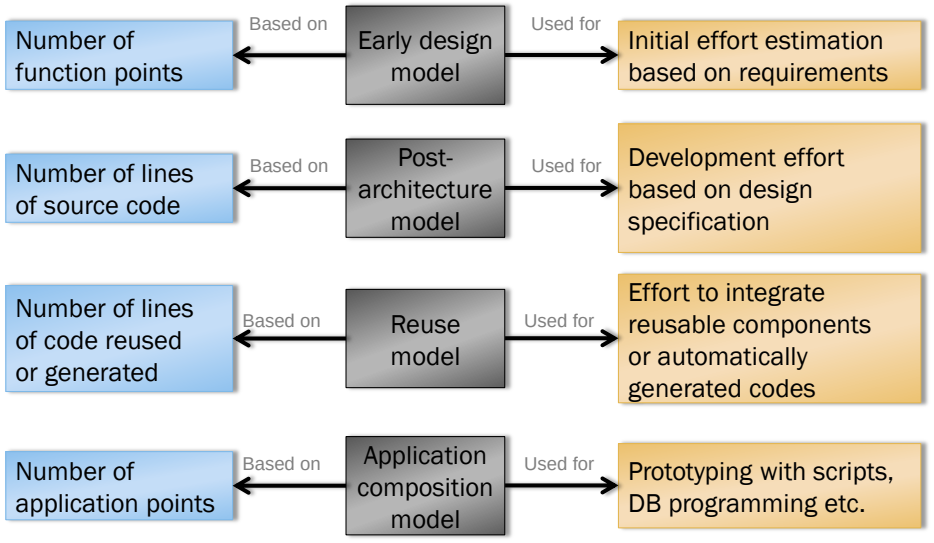
COCOMO 2 Model

- 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 (2000).
- COCOMO 2 takes into account different approaches to software development, reuse, etc.

COCOMO 2 Submodels

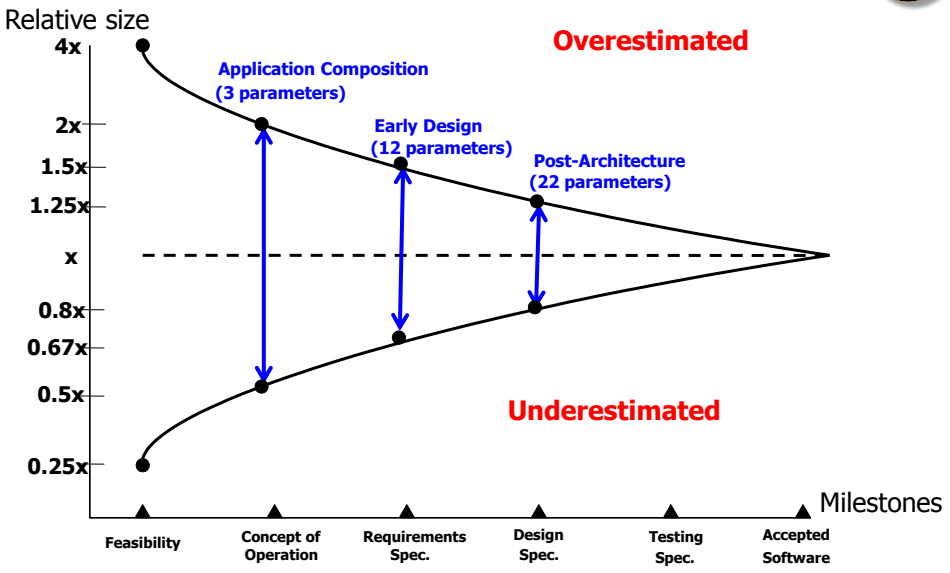
- COCOMO 2 incorporates a range of sub-models that produce increasingly detailed software estimates.
- The sub-models in COCOMO 2 are:
  - Application composition model. Used when software is composed from existing parts.
  - Early design model. Used when requirements are available but design has not 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.

Cocomo 2 Submodels



1.63

COCOMO 2 – Submodel Stages



Estimation for OO Projects - I

- Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.
- Using object-oriented analysis modeling , develop use-cases and determine a count.
- From the analysis model, determine the number of key classes
- Categorize the type of interface for the application and develop a multiplier for support classes:

Interface type	Multiplier
No GUI	2.0
Text-based user interface	2.25
GUI	2.5
Complex GUI	3.0

Estimation for OO Projects - II

- Multiply the number of key classes (step 3) by the multiplier to obtain an estimate for the number of support classes.
- Multiply the total number of classes (key + support) by the average number of work-units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
- Cross check the class-based estimate by multiplying the average number of work-units per use-case

Estimation for Agile Projects

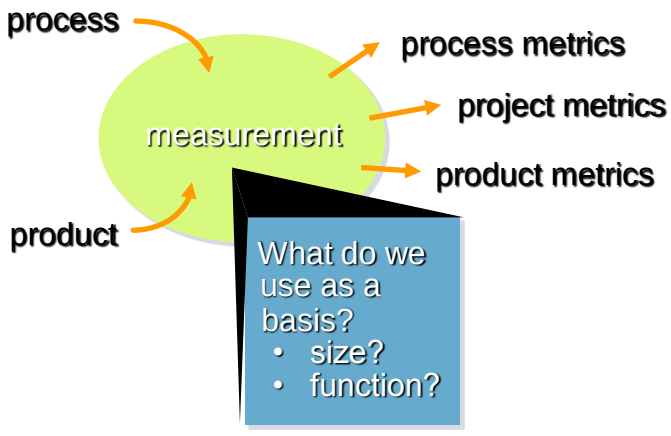
- Each user scenario (a mini-use-case) is considered separately for estimation purposes.
- The scenario is decomposed into the set of software engineering tasks that will be required to develop it.
- Each task is estimated separately. Note: estimation can be based on historical data, an empirical model, or "experience."
  - Alternatively, the 'volume' of the scenario can be estimated in LOC, FP or some other volume-oriented measure (e.g., use-case count).
- Estimates for each task are summed to create an estimate for the scenario.
  - Alternatively, the volume estimate for the scenario is translated into effort using historical data.
- The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

1. Project Management Concepts
2. Estimation
3. Process Metrics
4. Planning and Scheduling

Process and Project Metrics

4.3

A Good Manager Measures



Why Do We Measure?

- assess the status of an ongoing project
- track potential risks
- uncover problem areas before they go “critical,”
- adjust work flow or tasks,
- evaluate the project team’s ability to control quality of software work products.

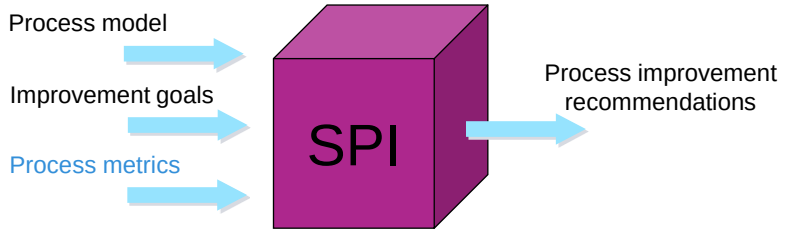
Process Measurement

- We measure the efficiancy of a software process indirectly.
  - That is, we derive a set of metrics based on the outcomes that can be derived from the process.
  - Outcomes include
    - measures of errors uncovered before release of the software
    - defects delivered to and reported by end-users
    - work products delivered (productivity)
    - human effort expended
    - calendar time expended
    - schedule conformance
    - other measures.
- We also derive process metrics by measuring the characteristics of specific software engineering tasks.

Process Metrics Guidelines

- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who collect measures and metrics.
- Don't use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them.
- Never use metrics to threaten individuals or teams.
- Metrics data that indicate a problem area should not be considered "negative." These data are merely an indicator for process improvement.
- Don't obsess on a single metric to the exclusion of other important metrics.

Software Process Improvement



Metrics

Process Metrics

- Quality-related
  - focus on quality of work products and deliverables
- Productivity-related
  - Production of work-products related to effort expended
- Statistical SQA data
  - error categorization & analysis
- Defect removal efficiency
  - propagation of errors from process activity to activity
- Reuse data
  - The number of components produced and their degree of reusability

Project Metrics

- used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.
- every project should measure:
  - *inputs*—measures of the resources (e.g., people, tools) required to do the work.
  - *outputs*—measures of the deliverables or work products created during the software engineering process.
  - *results*—measures that indicate the effectiveness of the deliverables.

Typical Project Metrics

- Effort/time per software engineering task
- Errors uncovered per review hour
- Scheduled vs. actual milestone dates
- Changes (number) and their characteristics
- Distribution of effort on software engineering tasks

Examples:

- total FP estimation (Function Points)
- total LOC estimation (Lines of Code)
- FP per person-month
- LOC per person-month
- errors or defects per KLOC (thousand LOC)
- pages of documentation per KLOC

Object-Oriented Metrics

- Number of scenario scripts (use-cases)
- Number of support classes (required to implement the system but are not immediately related to the problem domain)
- Average number of support classes per key class (analysis class)
- Number of subsystems (an aggregation of classes that support a function that is visible to the end-user of a system)

Web Project Metrics

- Number of static Web pages (the end-user has no control over the content displayed on the page)
- Number of dynamic Web pages (end-user actions result in customized content displayed on the page)
- Number of internal page links (internal page links are pointers that provide a hyperlink to some other Web page within the WebApp)
- Number of persistent data objects
- Number of external systems interfaced
- Number of static content objects
- Number of dynamic content objects
- Number of executable functions

Planning and Scheduling

4.4

Why Are Projects Late?

- an unrealistic deadline established by someone outside the software development group
- changing customer requirements that are not reflected in schedule changes;
- an honest underestimate of the amount of effort and/or the number of resources that will be required to do the job;
- predictable and/or unpredictable risks that were not considered when the project commenced;
- technical difficulties that could not have been foreseen in advance;
- human difficulties that could not have been foreseen in advance;
- miscommunication among project staff that results in delays;
- a failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem

Plan-Driven Development (PDD)

- Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail.
  - Plan-driven development is based on engineering project management techniques and is the 'traditional' way of managing large software development projects.
- A project plan is created that records the work to be done, who will do it, the development schedule and the work products.
- Managers use the plan to support project decision making and as a way of measuring progress.

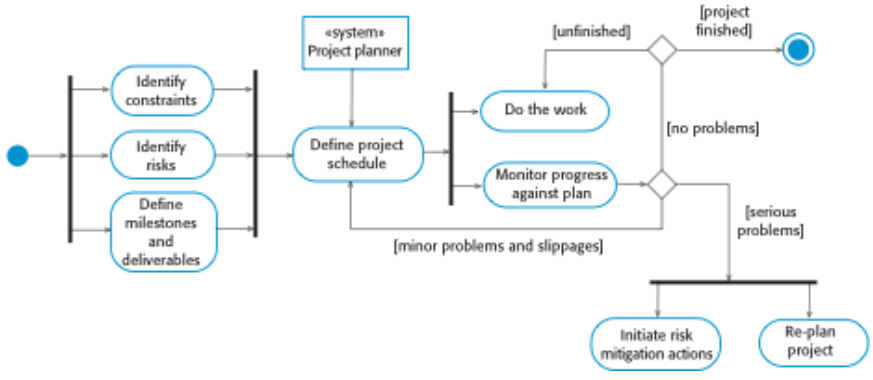
Plan-Driven Development Pros and Cons

- The arguments in favor of a plan-driven approach are that early planning allows organizational issues (availability of staff, other projects, etc.) to be closely taken into account, and that potential problems and dependencies are discovered before the project starts, rather than once the project is underway.
- The principal argument against plan-driven development is that many early decisions have to be revised because of changes to the environment in which the software is to be developed and used.

Project Plans

- In a plan-driven development project, a project plan sets out the resources available to the project, the work breakdown and a schedule for carrying out the work.
- Plan sections
  - Introduction
  - Project organization
  - Risk analysis
  - Hardware and software resource requirements
  - Work breakdown
  - Project schedule
  - Monitoring and reporting mechanisms

The project planning process



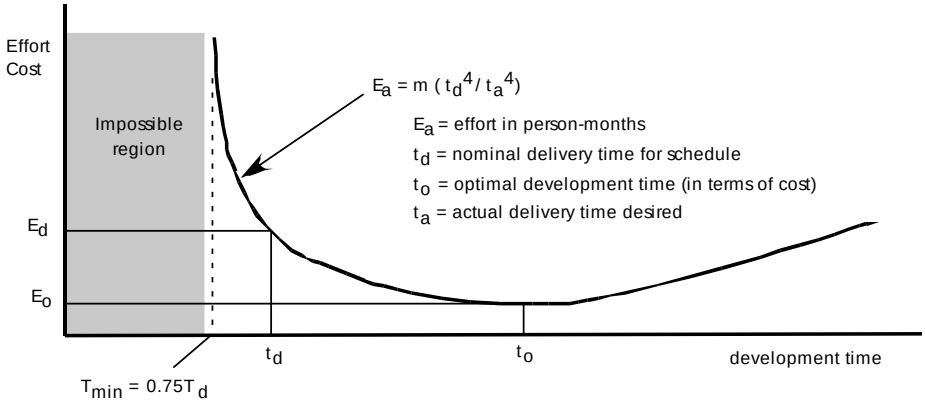
A Typical Software Project Plan Document

1. Introduction
   - A. Purpose of Plan
   - B. Project Scope and Objectives
2. Project Estimates
   - A. Historical Data Used for Estimates
   - B. Estimation Techniques
   - C. Estimates of Effort, Cost, Duration
3. Risks Management Strategy
   - A. Risk Table
   - B. Discussion of Risks to be Managed
   - C. RMMM Plan
4. Schedule
   - A. Project Work Breakdown Structure
   - B. Task Network
   - C. Timeline Chart
   - D. Resource Table
5. Project Resources
   - A. People
   - B. Hardware and Software
   - C. Special Resources
6. Staff Organization
   - A. Team Structure
   - B. Management Reporting
7. Tracking and Control Mechanisms
   - A. Quality Assurance and Control
   - B. Change Management and Control
8. Appendices

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



For Scheduling

- **PERT Chart:**
  - Shows the relationships based on tasks or activities
  - Defines tasks that can be done concurrently or not and critical path
- **Gantt Chart:**
  - Shows schedule (timeline) information for each task as a bar chart

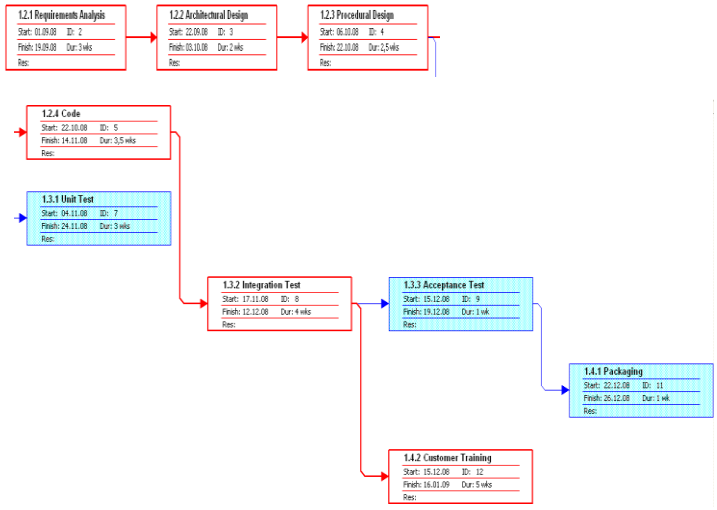
PERT Chart

- Project Evaluation and Review Technique
- Also known as Task Network Diagram
  - **Nodes:** activities / tasks and estimated duration
  - **Edges:** dependencies
- **Critical path:** Longest path from start to finish.
  Any slippage on the critical path will cause project delay.

PERT Example : Tasks and Durations

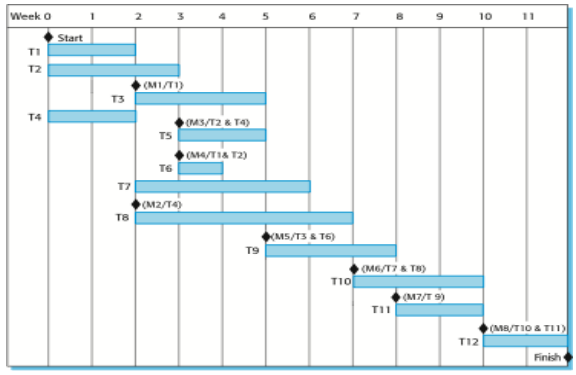
TASKS	WEEKS
1.2 Software Development	**11**
1.2.1 Requirements Analysis	3
1.2.2 Architectural Design	2
1.2.3 Procedural Design	2.5
1.2.4 Code	3.5
1.3 Testing	**7**
1.3.1 Unit Test	3
1.3.2 Integration Test	4
1.3.3 Acceptance Test	1
1.4 Operations	**5**
1.4.1 Packaging	1
1.4.2 Customer Training	5

PERT Chart

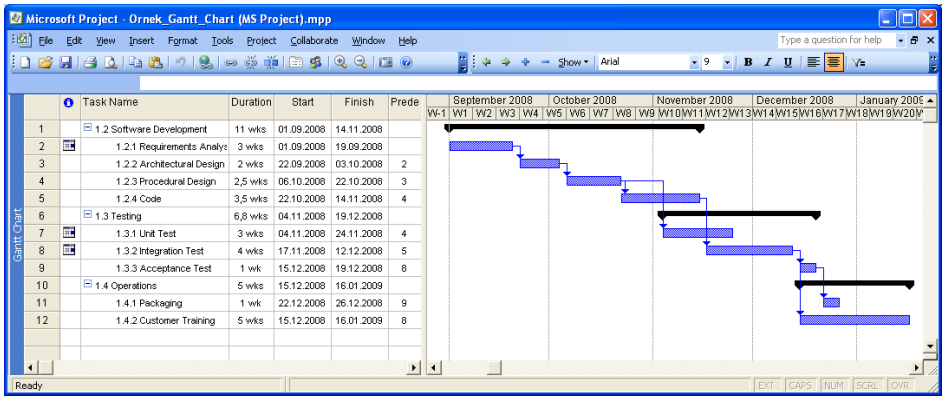


GANTT Chart

- Also known as timeline chart
- Shows high level view of whole project
  - Horizontal bars show duration of tasks
  - Triangles show milestones
  - Vertical lines show dependencies



GANTT Chart Example



Agile Project Scheduling

- Agile methods of software development are iterative approaches where the software is developed and delivered to customers in increments.
- Unlike plan-driven approaches, the functionality of these increments is not planned in advance but is decided during the development.
  - The decision on what to include in an increment depends on progress and on the customer's priorities.
- The customer's priorities and requirements change so it makes sense to have a flexible plan that can accommodate these changes.
- Release planning, which looks ahead for several months and decides on the features that should be included in a release of a system.
- Iteration planning, which has a shorter term outlook, and focuses on planning the next increment of a system. This is typically 2-4 weeks of work for the team.

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 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.

OOP Project Progress

- *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.
- *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 completed*
  - 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.