PROJECT MANAGEMENT

PROJECT MANAGEMENT

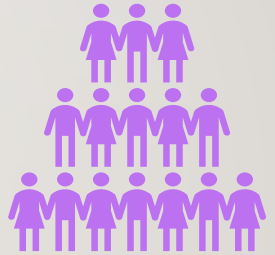
- Project management involves the planning, monitoring, and control of the people, process, and events that occur as software evolves from a preliminary concept to full operational deployment.

THE MANAGEMENT SPECTRUM

- 4 P's
  - People
  - Product
  - Process
  - Project

PEOPLE

People Capability Maturity Model (People-CMM)



The Stakeholders

Senior managers
Project (technical) managers
Practitioners
Customers
End users

- Team Leader
- The Software Team depends on following factors:
  - Difficulty of the problem to be solved
  - Size
  - Team lifetime
  - Degree to which the problem can be modularized
  - Quality and reliability
  - Rigidity of the delivery date
  - Communication required for the project

ORGANIZATIONAL PARADIGMS

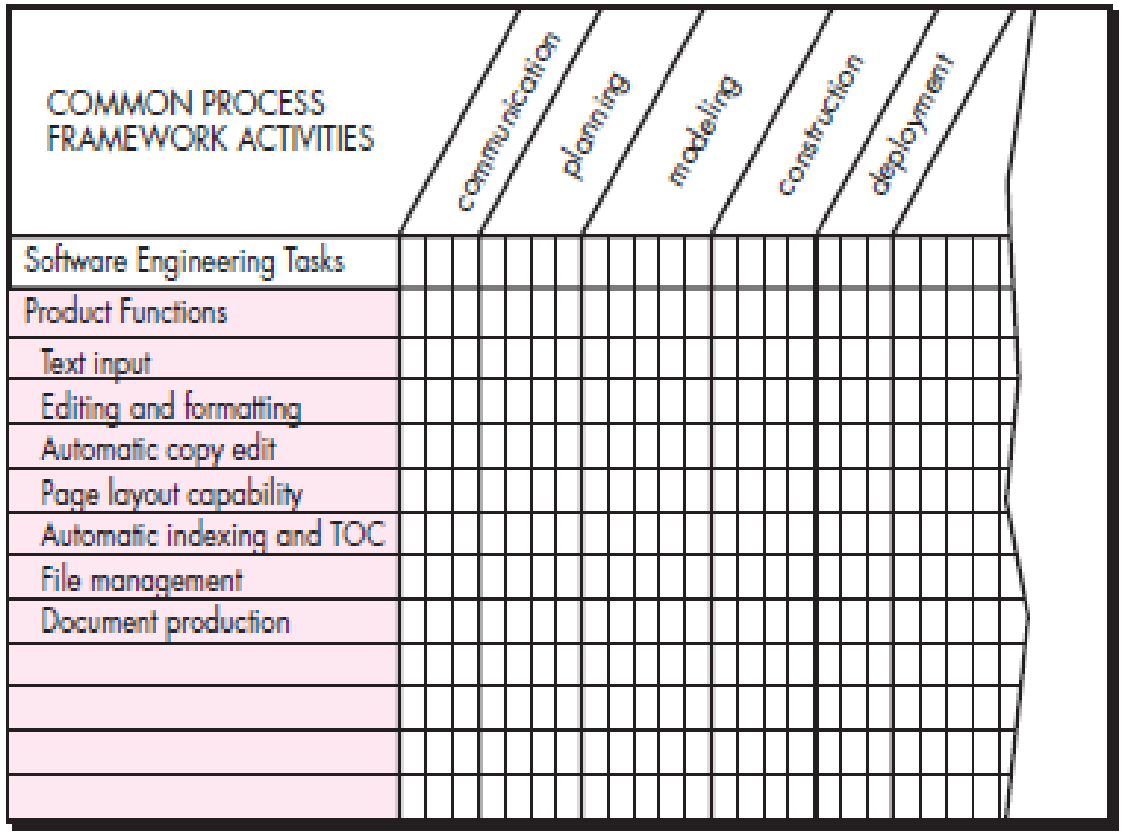
- Closed paradigm
- Random paradigm
- Open paradigm
- Synchronous paradigm

PRODUCT

- Product objectives and scope should be established
- Software Scope:
  - **Context.**
  - **Information objectives**
  - **Function and performance**
- Problem Decomposition

PROCESS

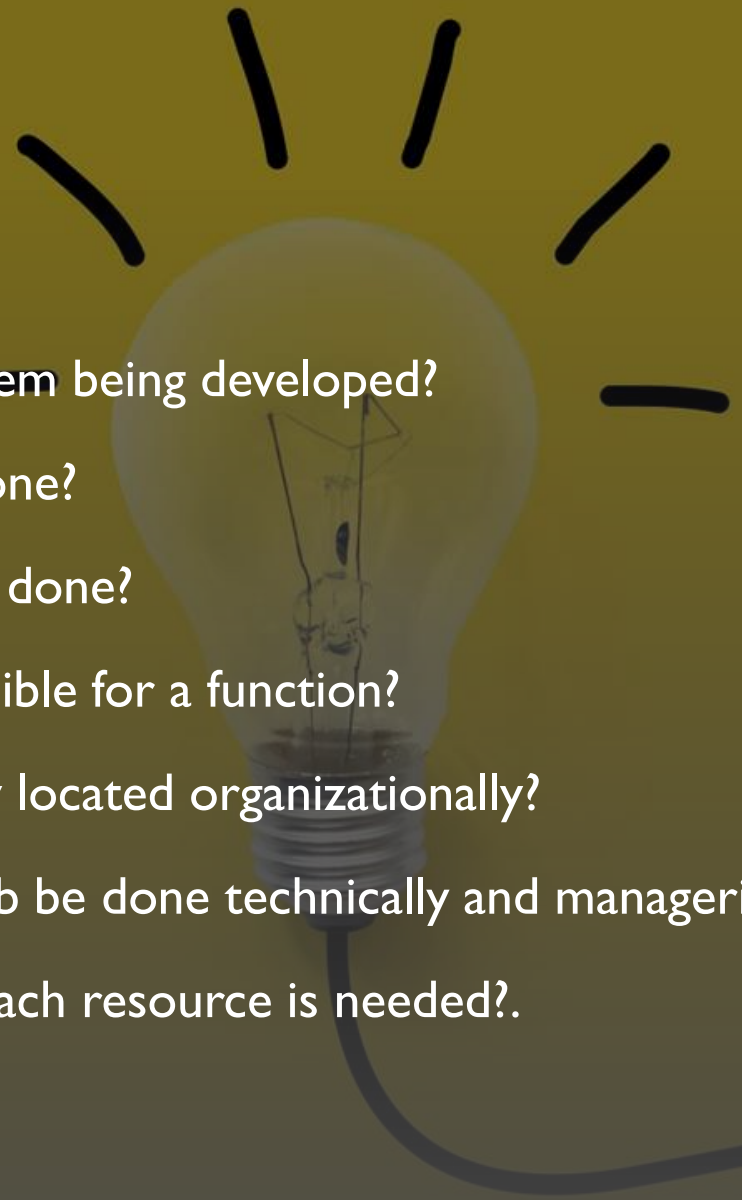
- A number of different task sets—tasks, milestones, work products, and quality assurance points



PROJECT

- Approach for planning, monitoring, and controlling the project.
- Five-part approach to software projects:
  - Start on the right foot
  - Maintain momentum.
  - Track progress
  - Make smart decisions
  - Conduct a postmortem analysis

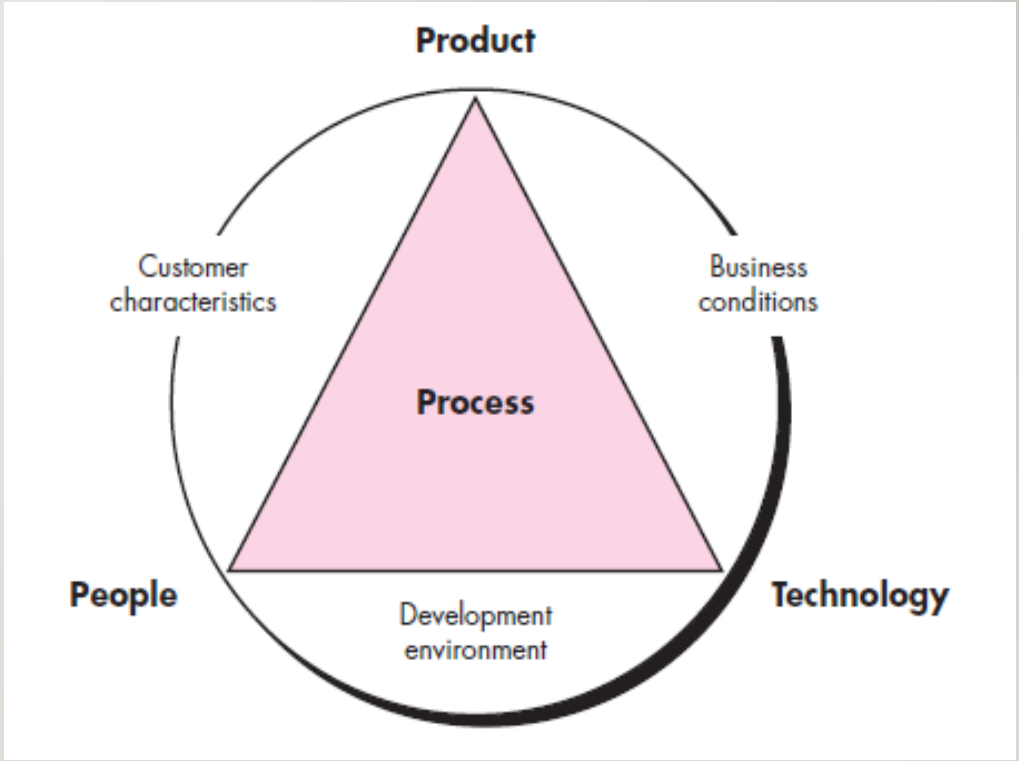
THE W5HH PRINCIPLE



- Why is the system being developed?
- What will be done?
- When will it be done?
- Who is responsible for a function?
- Where are they located organizationally?
- How will the job be done technically and managerially?
- How much of each resource is needed?.

PROCESS METRICS

- *Process metrics* are collected across all projects and over long periods of time.
- lead to long-term software process improvement.
- **Private Metrics**
- **Public metrics**



PROJECT METRICS

- Metrics collected from past projects are used as a basis from which effort and time estimates are made for current software work
- To minimize the development schedule by making the adjustments necessary to avoid delays and, mitigate potential problems and risks.
- project metrics are used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.

MEASUREMENT

- Direct Measure
  - Cost
  - Effort
- Indirect Measure
  - Functionality,
  - Quality
  - Complexity,
  - Efficiency, Reliability, Maintainability etc.

SOFTWARE MEASUREMENT

- process, project, and product metrics
- Product metrics that are private to an individual are often combined to develop project metrics that are public to a software team.
- Project metrics are then consolidated to create process metrics that are public to the software organization as a whole.

PRODUCT METRICS

- METRICS FOR THE REQUIREMENTS MODEL
  - Function based metrics
  - Metrics for Specification Quality
- METRICS FOR THE DESIGN MODEL
  - Architectural Design Metrics
- METRICS FOR SOURCE CODE-Halstead's Theory
- METRICS FOR TESTING
- METRICS FOR MAINTENANCE

METRICS FOR THE REQUIREMENTS MODEL

FUNCTION BASED METRICS

- Predicting the “size” of the resultant system
- A means for measuring the functionality delivered by a system.
- Information domain values are defined as:
  - **Number of external inputs (Eis)-** originates from a user or is transmitted from another application and provides distinct application-oriented data or control information.
  - **Number of external outputs (EOs).** Each *external output* is derived data within the application that provides information to the user. In this context external output refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.
  - **Number of external inquiries (EQs).** An *external inquiry* is defined as an online input that results in the generation of some immediate software response in the form of an online output (often retrieved from an ILF)
  - **Number of internal logical files (ILFs).** Each *internal logical file* is a logical grouping of data that resides within the application’s boundary and is maintained via external inputs.
  - **Number of external interface files (EIFs).** Each *external interface file* is a logical grouping of data that resides external to the application but provides information that may be of use to the application.

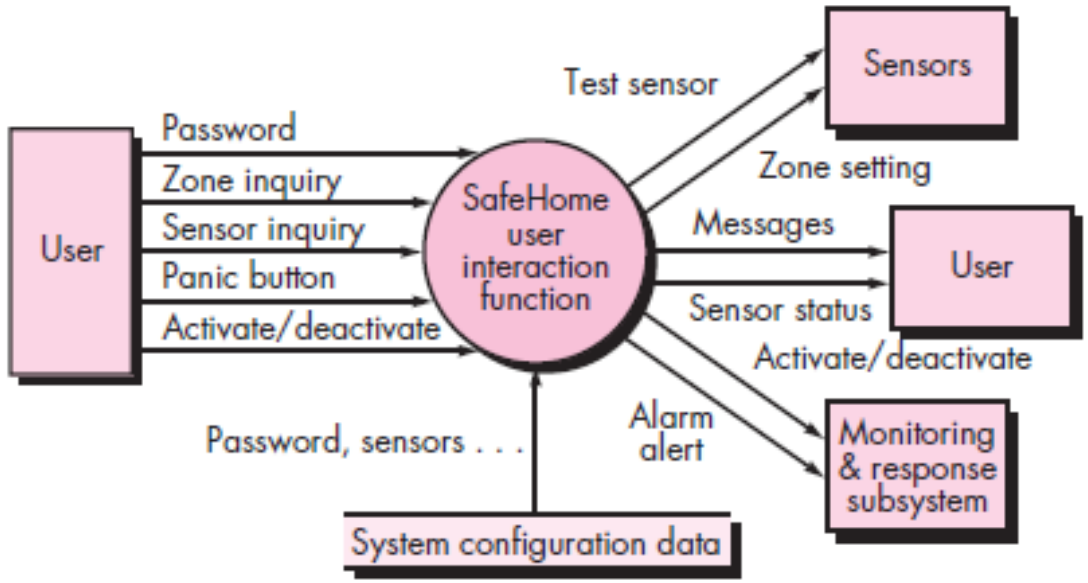
- To compute function points (FP), the following relationship is used:

FP = count total * [0.65 + 0.01∗ $\sum F_i$]

- The *Fi* (*i* 1 to 14) are *value adjustment factors* (VAF) based on responses to the following such questions:
  - Are the ILFs updated online?
  - Are the inputs, outputs, files, or inquiries complex?
  - Is the internal processing complex?
  - Is the code designed to be reusable?
- On a scale of ranges from 0 (not important or applicable) to 5 (absolutely essential

Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total							

Information Domain Value	Count		Weighting factor: Simple	Average	Complex		
External Inputs (EIs)	3	×	(3)	4	6	=	9
External Outputs (EOs)	2	×	(4)	5	7	=	8
External Inquiries (EQs)	2	×	(3)	4	6	=	6
Internal Logical Files (ILFs)	1	×	(7)	10	15	=	7
External Interface Files (EIFs)	4	×	(5)	7	10	=	20
Count total	→						50



FP= 50 * [0.65 + (0.01 * 46)]= 56

	average
User Input = 50	4
User Output = 40	5
User Inquiries = 35	4
User Files = 6	10
External Interface = 4	7

F=42

- Function point=????

	average
User Input = 50	4
User Output = 40	5
User Inquiries = 35	4
User Files = 6	10
External Interface = 4	7

F=42

- Function point= 671.96

METRICS FOR SPECIFICATION QUALITY

$$n_r = n_f + n_{nf}$$

- *Specificity* $Q_1 = n_{UI} / n_n$
- *Completeness* $Q_2 = n_c / (n_c + n_{nv})$

METRICS FOR THE DESIGN MODEL

- ***Structural complexity*** of a module *i* is defined in the following manner:

$$S_i = f_{out}^2(\text{i})$$

- ***Data complexity*** provides an indication of the complexity in the internal interface for a module *i* and is defined as:

$$D_i = v(i)/(f_{out}(\text{i})+1)$$

- ***System complexity*** is defined as the sum of structural and data complexity, specified as

$$C(i) = S(i) + D(i)$$

METRICS FOR SOURCE CODE- HALSTEAD'S THEORY

*n*1 = number of distinct operators that appear in a program
*n*2 = number of distinct operands that appear in a program
*N*1 =total number of operator occurrences
*N*2 = total number of operand occurrences

- **Length**

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

- **Volume**

$$V = N \log_2 (n_1 + n_2)$$

- **Level**

$$L = \frac{2}{n_1} * \frac{n_2}{N_2}$$

- **Difficulty**

$$D = I/L$$

- **Effort**

$$E = V*D$$

COUNTING RULES FOR C LANGUAGE

- Comments are not considered.
- The identifier and function declarations are not considered
- All the variables and constants are considered operands.
- Global variables used in different modules of the same program are counted as multiple occurrences of the same variable.
- Local variables with the same name in different functions are counted as unique operands.

CONTINUE.....

- Functions calls are considered operators.
- All looping statements e.g., do {...} while ( ), while ( ) {...}, for ( ) {...}, all control statements e.g., if ( ) {...}, if ( ) {...} else {...}, etc. are considered as operators.
- In control construct switch ( ) {case:...}, switch as well as all the case statements are considered as operators.
- The reserve words like return, default, continue, break, size, etc., are considered operators.
- All the brackets, commas, and terminators are considered operators.

CONTINUE.....

- GOTO is counted as an operator and the label is counted as an operand.
- The unary and binary occurrences of “+” and “-” are dealt with separately. Similarly “*” (multiplication operator) is dealt with separately.
- In the array variables such as “array-name [index]” “array-name” and “index” are considered as operands and [ ] is considered as operator.
- In the structure variables such as “struct-name, member-name” or “struct-name -> member-name”, struct-name, and member-name are taken as operands, and ‘.’, ‘->’ are taken as operators. Some names of member elements in different structure variables are counted as unique operands.
- All the hash directives are ignored.

```
int sort (int x[ ], int n)

{
int i, j, save, im1;
/*This function sorts array x in ascending order */
If (n< 2) return 1;
for (i=2; i< =n; i++)
{
im1=i-1;
for (j=1; j< =im1; j++)
if (x[i] < x[j])
{
Save = x[i];
x[i] = x[j];
x[j] = save;
}
}
return 0;
}
```

V = 417.23 bits

Operators	Occurrences	Operands	Occurrences
int	4	sort	1
()	5	x	7
,	4	n	3
[]	7	i	8
if	2	j	7
<	2	save	3
;	11	im1	3
for	2	2	2
=	6	1	3
–	1	0	1
<=	2	–	–
++	2	–	–
return	2	–	–
{}	3	–	–
n1=14	N1=53	n2=10	N2=38

METRICS FOR TESTING

- Halstead Metrics Applied to Testing

$$PL = \frac{1}{\left(n_1/2\right) * \left(N_2/n_2\right)}$$

$$e = \frac{V}{PL}$$

METRICS FOR MAINTENANCE

MT = number of modules in the current release

Fc = number of modules in the current release that have been changed

Fa = number of modules in the current release that have been added

Fd = number of modules from the preceding release that were deleted in the current release

- *software maturity index* (SMI) that provides an indication of the stability of a software product

$$\text{SMI} = \frac{M_T - (F_a + F_a + F_d)}{M_r}$$