SOFTWARE ENGINEERING

Lecturer Dr. Osman AKBULUT

WEEK 10. METRICS

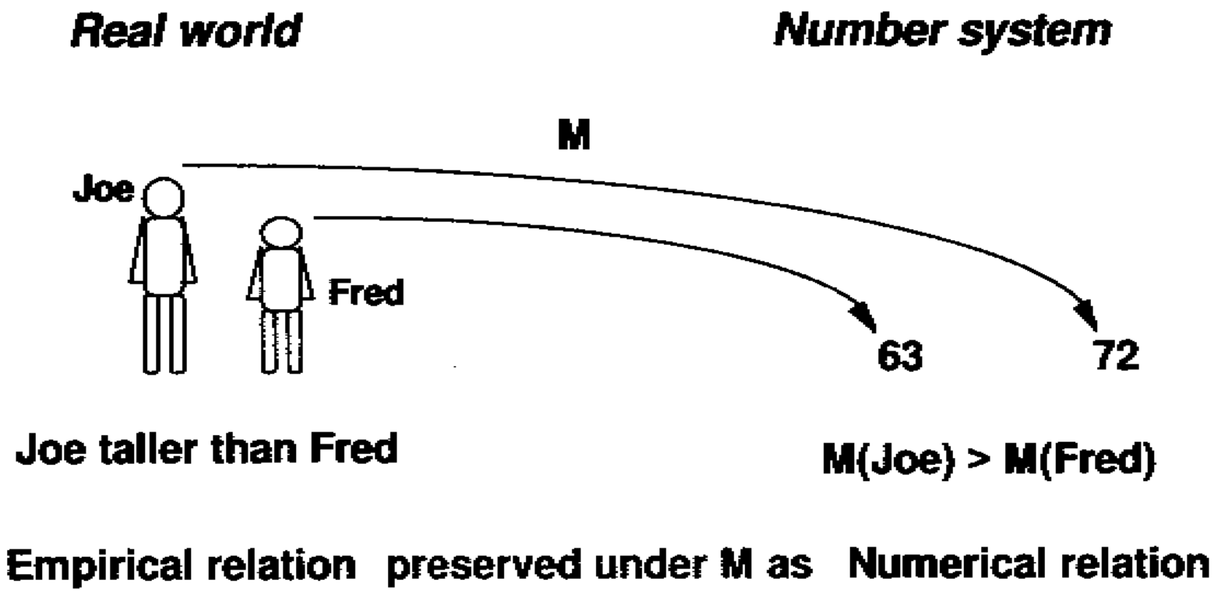
OUTLINE

- Theory of Measurement
- Software Metrics
  - Size oriented Metrics
  - Function oriented Metrics
  - Quality Metrics

THEORY OF MEASUREMENT

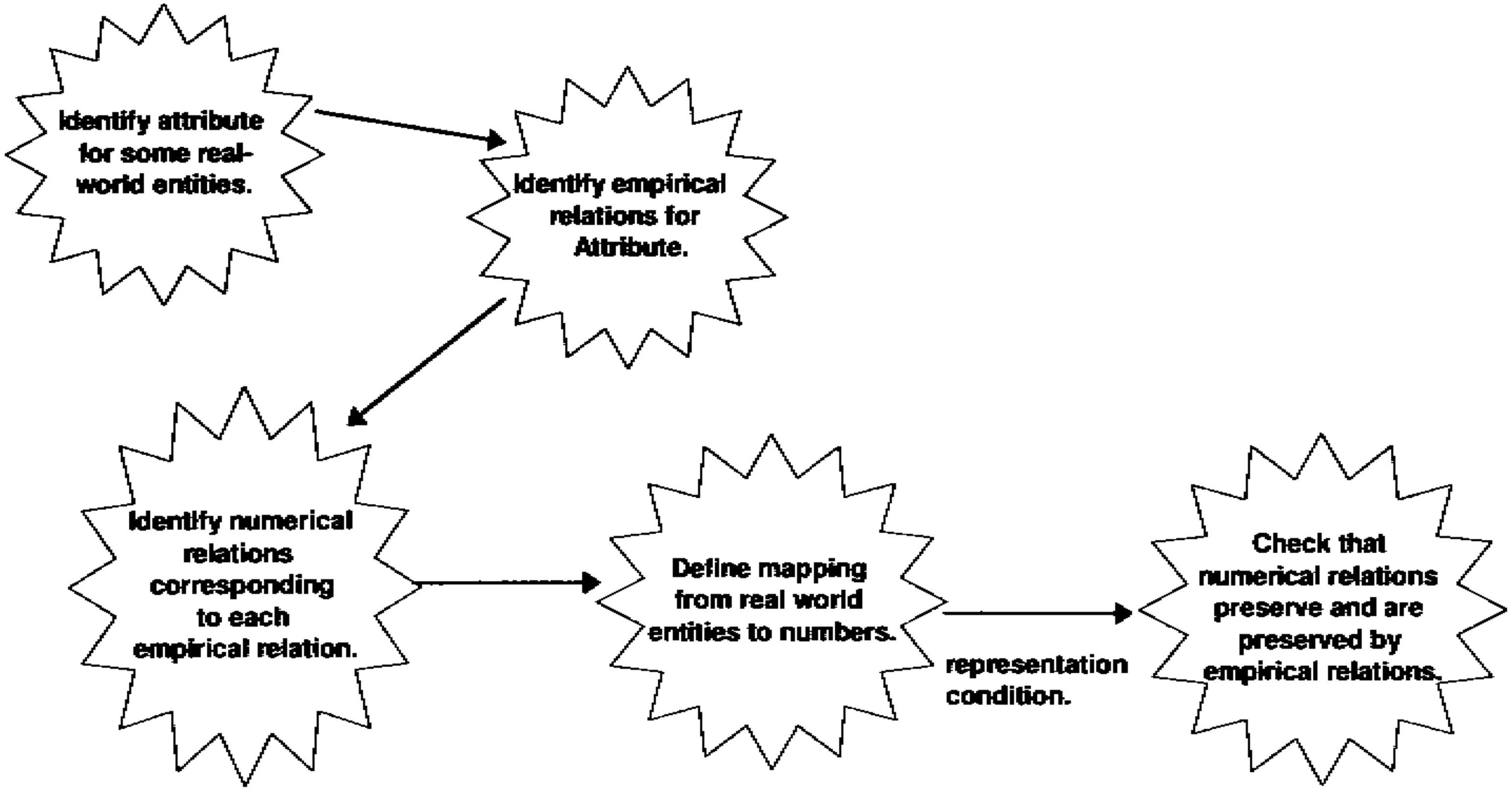
- Measurement process can be characterized by 5 activities;
  - Formulation
  - Collection
  - Analysis
  - Interpretation
  - Feedback

MEASUREMENT



*Empirical: Deneysel

FOR MEASURING



MEASUREMENT SCALES

- Nominal (gender)
- Ordinal (arrival order)
- Interval (temperatures in F)
- Ratio (height)
- Absolute (the actual count)

*Scale: Ölçek

I will not ask...

NOMINAL MEASURES

- Language(Program) = 1, if Program is written in Pascal
- Language(Program) = 2, if Program is written in C
- Language(Program) = 3, if Program is written in Fortran

Few mathematical operations are applicable (mode, histograms, …)

*Nominal: İtibari

I will not ask..

ORDINAL MEASURES

- Difficult(Program) = 1, if Program is easy to read
- Difficult(Program) = 2, if Program is not hard to read
- Difficult(Program) = 3, if Program is hard to read

We can have the median here…

*Ordinal: Sıralı

I will not ask..

INTERVAL, RATIO, AND ABSOLUTE

- Interval measures preserve differences but not ratios. Ex., The time interval when an event occurred. (start and end)
- Ratio measures preserve also the ratio between entities. Ex., LOC in a program. All math operations are applicable.
- Absolute measures are counts. Ex., the number of if statements in a program.

*Interval: Aralık
Ratio: Oran
Absolute: Mutlak

I will not ask...

SOFTWARE METRICS

- Software Metrics are used to measure the software with some measurement scales to help engineers by building higher-quality softwares.
- The metrics should be,
  - Simple and computable
  - Empirically and intuitively persuasive
  - Consistent and objective
  - Consistent in its use of units and dimensions
  - Programming language independent
  - An effective mechanism for high-quality feedback

*Intuitive: Sezgisel
Persuasive: İkna edici

KIND OF METRICS

- A metric is objective if it can be taken by an automated device; it is subjective otherwise
  - *LOC are objective metrics, Function Points are subjective*
- A metric is direct if it can be directly detected, indirect if it is the result of mathematical elaboration on other metrics
  - *LOC, number of errors, and FP are direct*
  - *Number of errors per LOC (Error density) is indirect*

*Elaboration: Detaylandırma

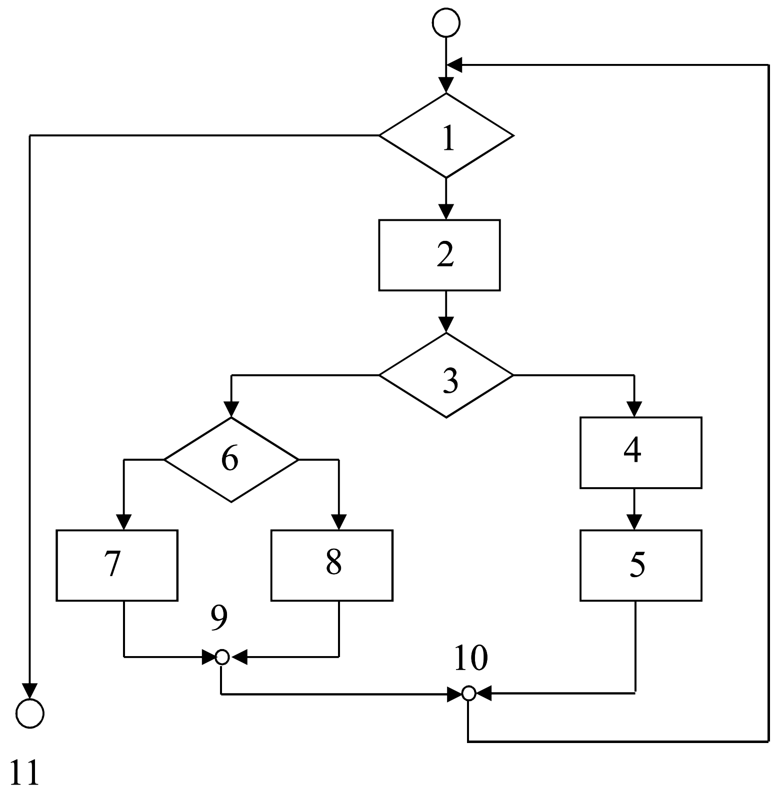
SIZE-ORIENTED METRICS

- errors per KLOC (thousand lines of code)
- defects per KLOC
- $ per LOC
- page of documentation per KLOC
- errors / person-month
- LOC per person-month
- $ / page of documentation

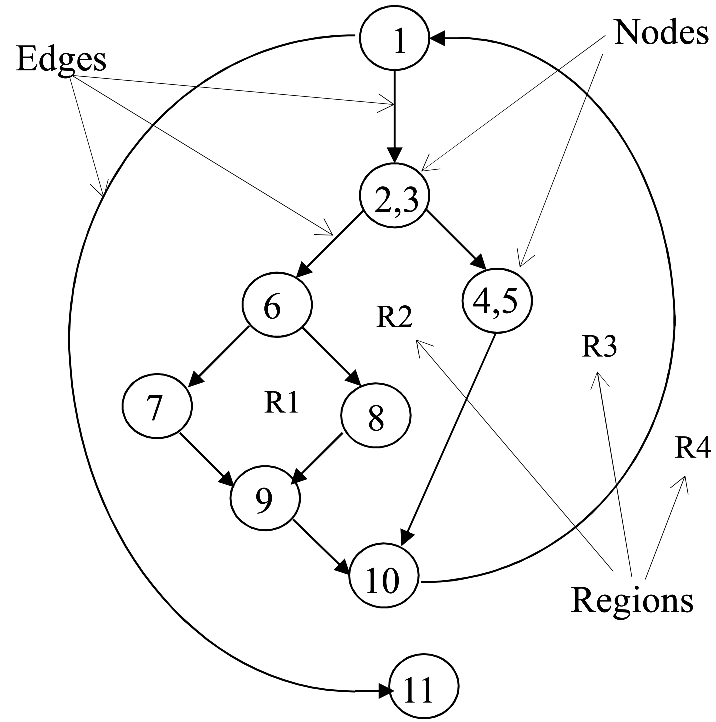
LINES OF CODE

- Need for a standard
  - For instance we use the count of the ";"
- Once a standard is set they can be computed automatically (Objective metrics)
- **They MUST not be used to evaluate people productivity (easy to alter!!!)**
- When used "properly" they work!!

CYCLOMATIC COMPLEXITY



Flow Chart

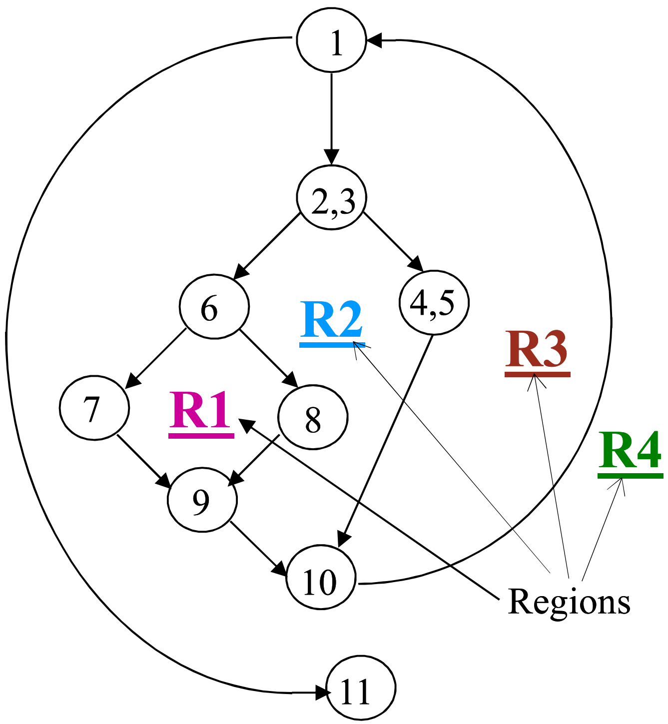


Flow Graph

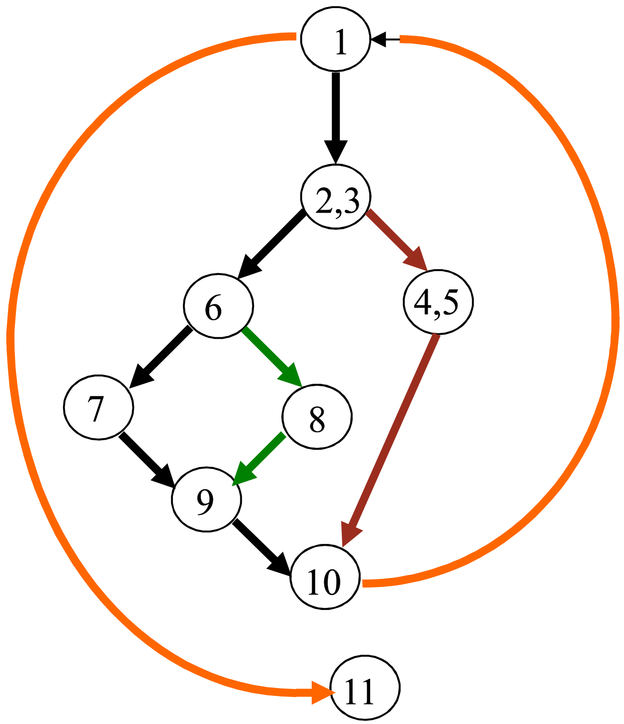
CYCLOMATIC COMPLEXITY (DEF)

- **V(G) = #Regions in the Graph**
- **V(G) = #Independent Paths in the Graph**
- **V(G) = E - N + 2** where E = number of edges and N = number of nodes
- **V(G) = P + 1** P = number of predicated nodes (i.e., if, case, while, for, do)

CYCLOMATIC COMPLEXITY (DEF)

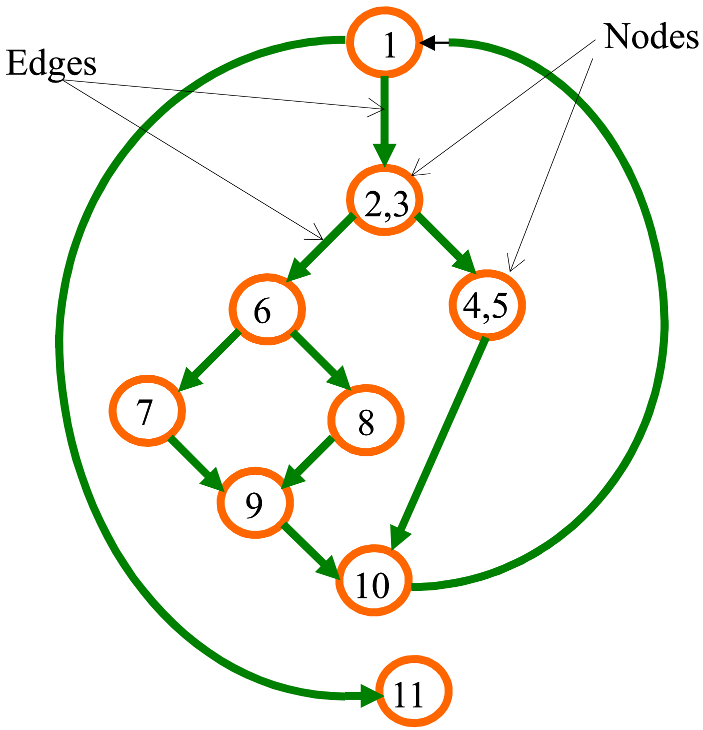


4 regions!!!

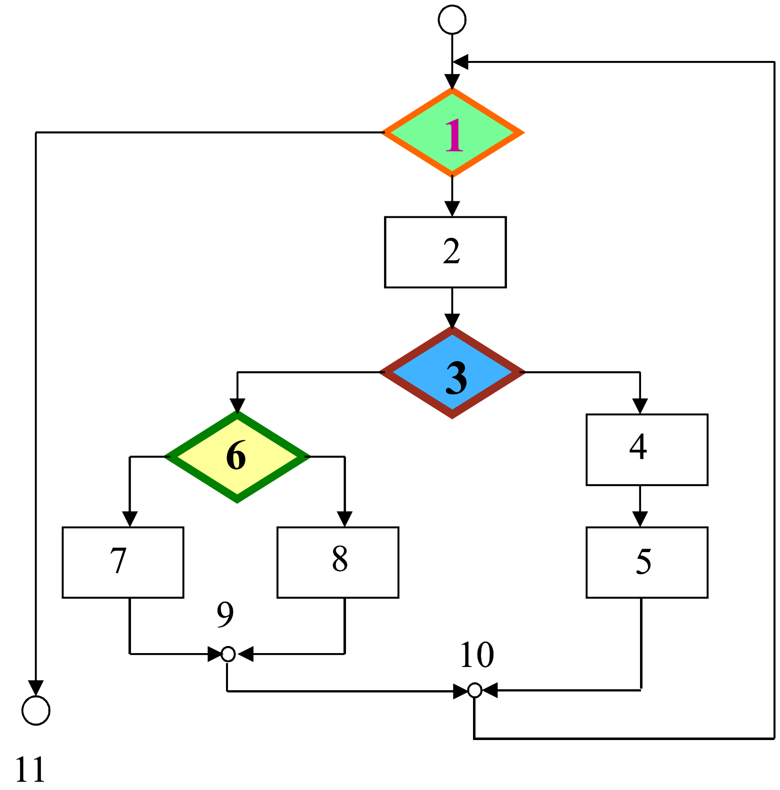


4 independent paths!!!

CYCLOMATIC COMPLEXITY (FORMULA)



11 Edges, 9 Nodes …
11-9+2 = 4 !!!



3+1 = 4 !!

FAN IN AND FAN OUT

- The Fan In of a module is the amount of information that "enters" the module
- The Fan Out of a module is the amount of information that "exits" a module
- We assume all the pieces of information with the same size
- Fan In and Fan Out can be computed for functions, modules, objects, and also non-code components

COMPUTING FAN IN AND FAN OUT

- Usually:
  - **Parameters passed by values count toward Fan In**
  - **External variables used before being modifies count toward Fan In**
  - **External variables modified in the block count toward Fan Out**
  - **Return values count toward Fan Out**
  - **Parameters passed by reference … depend on their use…**

SIMPLE EXAMPLE OF FAN IN / FAN OUT

```
#define<stdio.h>
#define<math.h>                       fan-in    fan-out

int globalInVar = 9;
int globalOutVar;

float Simple(float x, float y){         2
      int a;
      float z;
      z = sqrt( x + y +
              globalInVar);             1
      globalOutVar = int(z+2);                     1
      return z;                                    1
}
                                    -------------------
                                        3          2
```

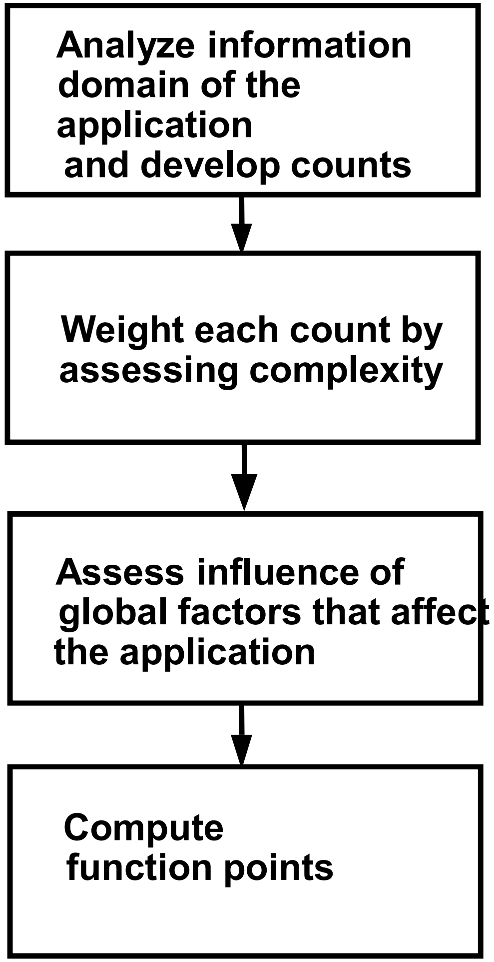
MORE INVOLVED EXAMPLE

	fan-in	**fan-out**
#define<stdio.h>		
#define<math.h>		
int globalVarA = 0;		
int globalVarB = 3;		
float global VarC = 7.0;		
float chechValue(float x, float y){	**2**	
int a;		
float z;		
z = sqrt(x + y + globalVarC);	**1**	
globalVarA ++;		**1**
a = globalVarB;	**1**	
globalVarC = z + (float)globalVarA;	**1**	**1**
return z;		**1**
}	------------	------------
	5	**3**

FUNCTION-ORIENTED METRICS

- Function Points is a measure of "how big" is the program, independently from the actual physical size of it
- It is a weighted count of several features of the program
- Dislikes claim FP make no sense with the representational theory of measurement
- There are firms and institutions taking them very seriously

FUNCTION POINTS



Establish *count* for input domain and system interfaces

Assign level of complexity or *weight* to each count

Grade significance of external factors, F_i such as reuse, concurrency, OS, ...

function points = $\sum$ (count x weight) x C

where:

complexity multiplier: $C = (0.65 + 0.01 \times N)$

degree of influence: $N = \sum F_i$

ANALYZING THE INFORMATION DOMAIN

measurement parameter	count		weighting factor simple	avg.	complex		
number of user inputs	$\boxed{}$	X	3	4	6	=	$\boxed{}$
number of user outputs	$\boxed{}$	X	4	5	7	=	$\boxed{}$
number of user inquiries	$\boxed{}$	X	3	4	6	=	$\boxed{}$
number of files	$\boxed{}$	X	7	10	15	=	$\boxed{}$
number of ext.interfaces	$\boxed{}$	X	5	7	10	=	$\boxed{}$
Unadjusted Function Points:						→	$\boxed{}$
complexity multiplier							$\boxed{}$
function points						→	$\boxed{}$

$$\sum_{Inputs} Wi + \sum_{Output} Wo + \sum_{Inquiry} Win + \sum_{InternalFiles} Wif + \sum_{ExternalInterfaces} Wei$$

TAKING COMPLEXITY INTO ACCOUNT

Factors are rated on a scale of 0 (not important) to 5 (very important):

data communications	on-line update
distributed functions	complex processing
heavily used configuration	installation ease
transaction rate	operational ease
on-line data entry	multiple sites
end user efficiency	facilitate change

Formula:

$$CM = \sum_{ComplexityMultiplier} F_{ComplexityMultiplier}$$

COMPLEXITY

measurement parameter	count		weighting factor simple	avg.	complex		
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 ext.interfaces	☐	X	5	7	10	=	☐

Unadjusted Function Points: → ☐

complexity multiplier ☐

Function Points: → ☐

FP = UFPx(0.65+0.01xCM)

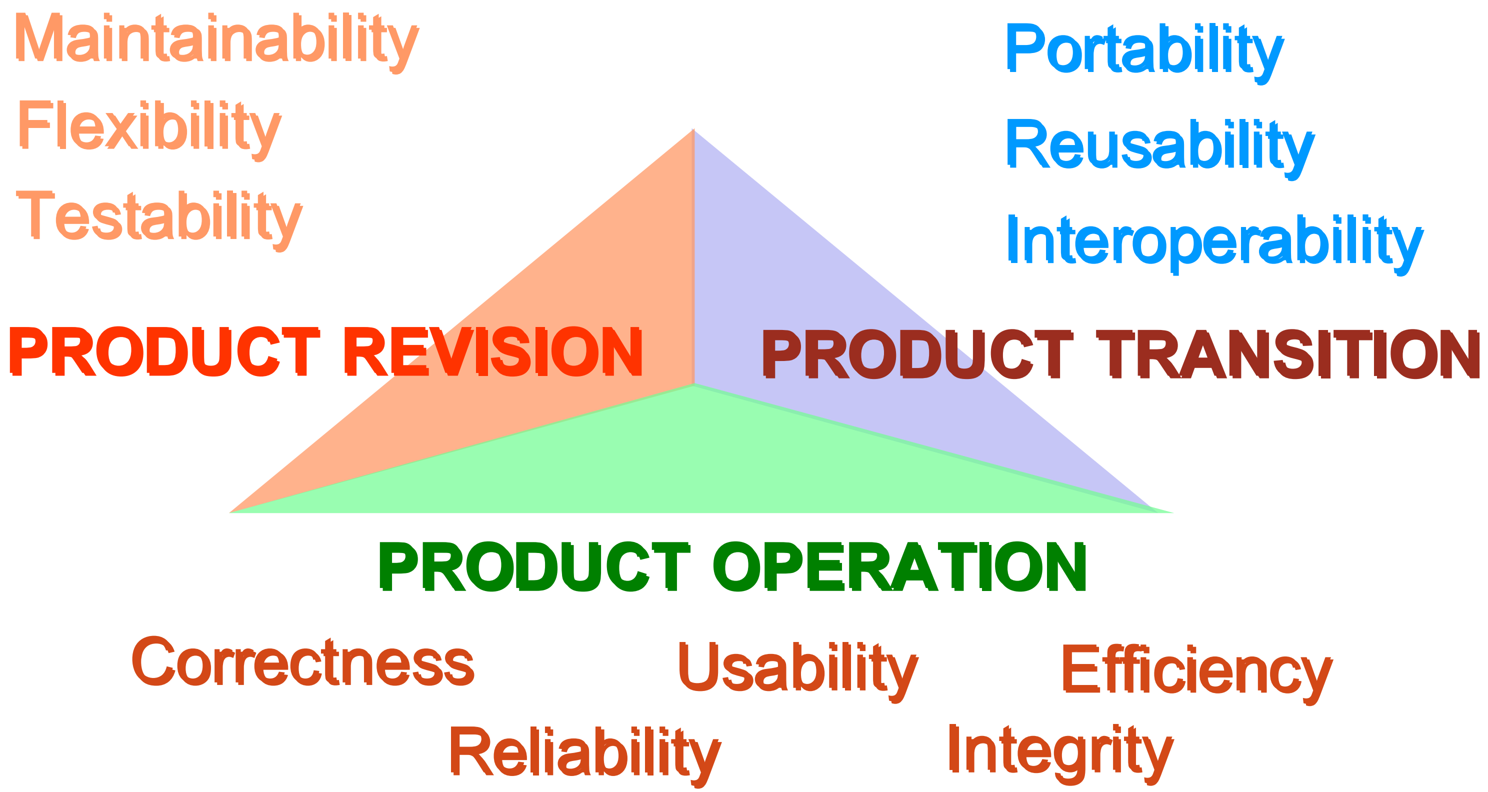
	measurement parameter	count		Weighting Factor simple	average	complex		
number of user inputs		3	x	3	4	6	=	9
number of user outputs		2	x	4	5	7	=	8
number of user inquiries		2	x	3	4	6	=	6
number of files		1	x	7	10	15	=	7
number of external interfaces		4	x	5	7	10	=	20
count-total								**50**

Using FP = count total x $[0.65 + 0.01 \times \sum F_i]$ where $\sum F_i = 46$, we get

$$FP = 50 \times [0.65 + 0.01 \times 46]$$

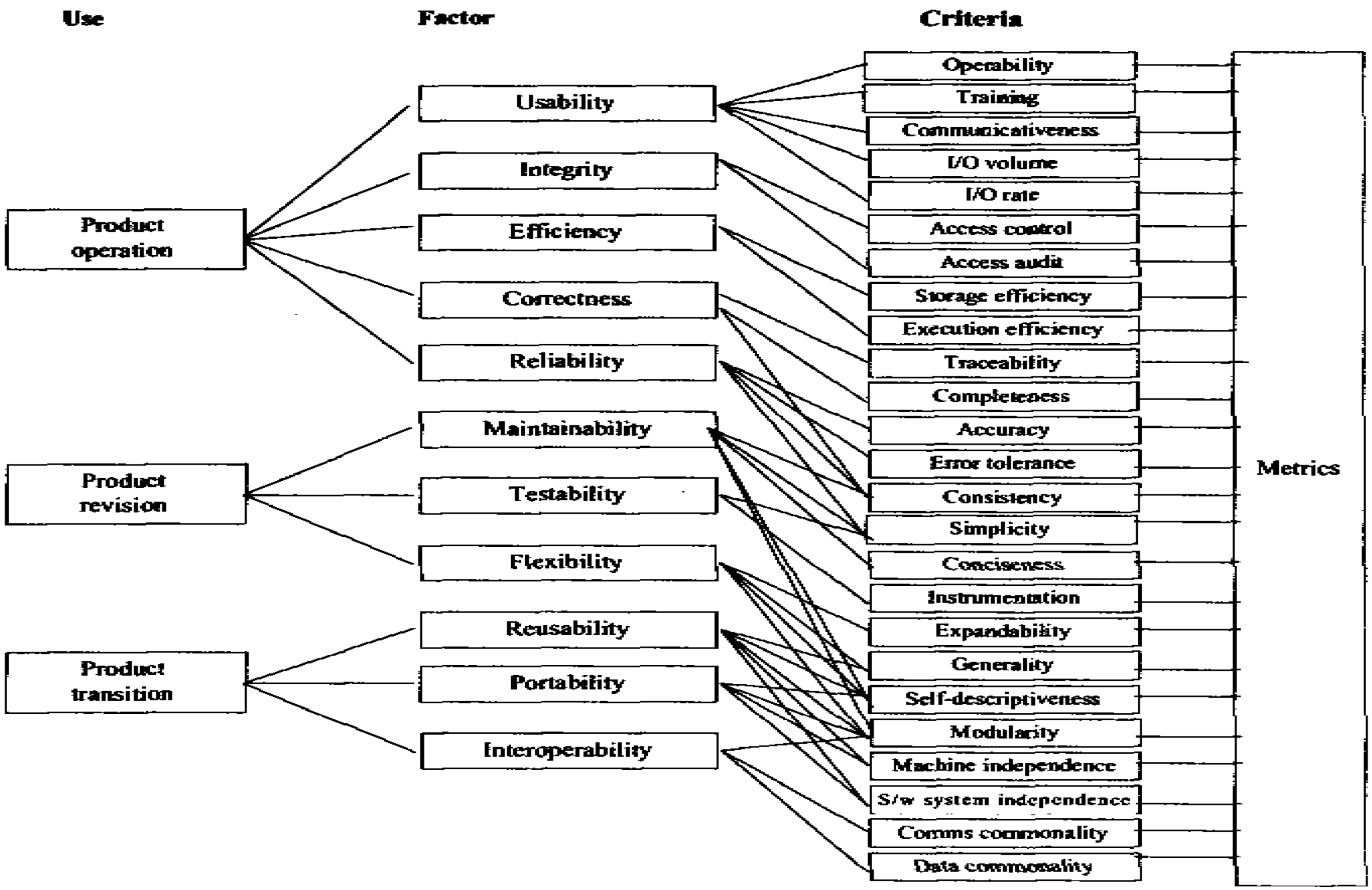
$$FP = 56$$

QUALITY METRICS



McCall's Triangle of Quality

MCCALL SOFTWARE QUALITY MODEL



MCCALL'S TRIANGLE

- Correctness: The extent to which a program satisfies its specification and fulfills the customer's mission objectives
- Reliability: The extent to which a program can be expected to perform its intended function with required precision
- Efficiency: The amount of computing resources and code required by a program to perform its function
- Integrity: Extent to which access to software or data by unauthorized persons can be controlled
- Usability: Effort required to learn, operate, prepare input and interpret output of a program
- Maintainability: Effort required to locate and fix an error in a program

*Precision: Kesinlik
Integrity: Güvenilirlik

MCCALL'S TRIANGLE (CONT.)

- Flexibility: Effort required to modify an operational program
- Testability: Effort required to test a program to ensure that it performs its intended function
- Portability: Effort required to transfer the program from one hardware and/or software system environment to another
- Reusability: Extent to which a program can be reused in other applications
- Interoperability: Effort required to couple one system to another

ISO 9126 QUALITY FACTORS

- The standard identifies 6 key attributes;
  - Functionality: The degree to which the software satisfies stated needs
  - Reliability: The amount of time that the software is available for use
  - Usability: The degree to which the software is easy to use

ISO 9126 QUALITY FACTORS

- (cont.)
  - Efficiency: The degree to which the software makes optimal use of system resources
  - Maintainability: The ease with which repair may be made to the software
  - Portability: The ease with which the software can be transposed from one environment to another

TEAM MANAGEMENT

TEAM?

- The products, when they are too large within the given time constraints must be assigned to a group of professionals organized as team.
- Working as a team may be in two types;
  - Share-able tasks (strawberry picking)
  - Nonshare-able tasks (baby production)

DIFFICULTIES OF TEAM PROGRAMMING

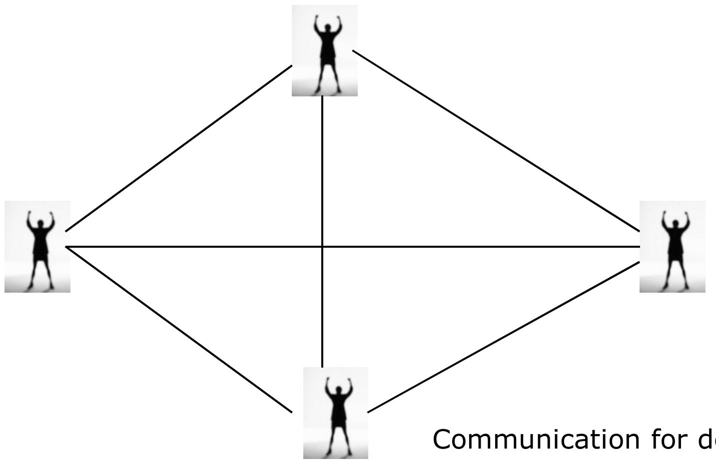
- Problems in communication
- Geometrically increasing communication need (Communication need for 3 members = 3, Communication need for 4 members = 6)

2 EXTREME APPROACHES

- There are 2 extreme approaches to programming-team organizations (Some more approaches were also developed to strength the weak points of these approaches)
  - Democratic Team Approach
  - Classical Chief Programmer Team Approach

DEMOCRATIC TEAM APPROACH (1/2)

- The basic concept underlying the democratic team is egoless programming.
- The difficulty for the programmer is seeing a module as an extension of his or her ego and not to try find them.
- So democratic teams see faults as bugs.



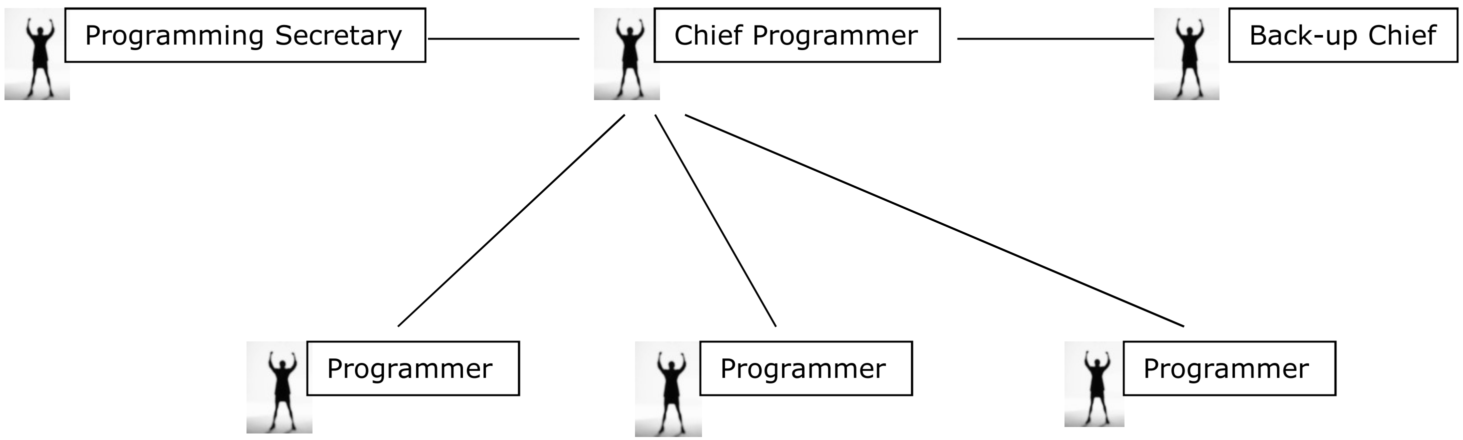
Communication for democratic team approach

DEMOCRATIC TEAM APPROACH (2/2)

- Advantage:
  - Positive attitude toward finding faults
  - So high quality code
- Disadvantage:
  - Hard to manage for managers

CLASSICAL CHIEF PROGRAMMER TEAM APPROACH (1/2)

- To decrease the communication need between the members, it can be designed a hierarchy between them
- The members will have some roles:
  - Chief Programmer: has technical and managerial background
  - Back-up Programmer: is as good as Chief, but waits until the Chief leaves
  - Programming Secretary: documents the project and controls the communication
  - Programmer: only writes codes and don't care about the rest of the project



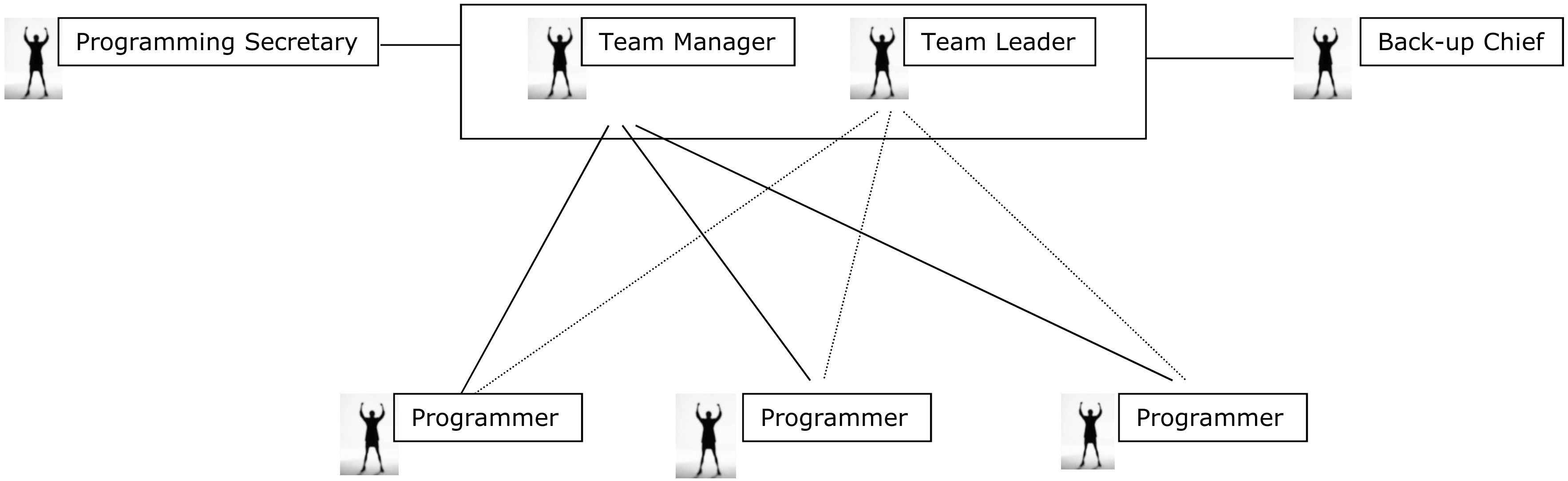
Communication for classical chief programmer team approach

CLASSICAL CHIEF PROGRAMMER TEAM APPROACH (2/2)

- Impracticality?
  - Chief: Too hard to find such a complete chief, with managerial and technical skills!
  - Back-up Chief: It is harder to find to find one more chief to take back seat
  - Programming Secretary: Software Engineers are usually notorius for paperwork

MODIFIED CHIEF PROGRAMMER TEAM

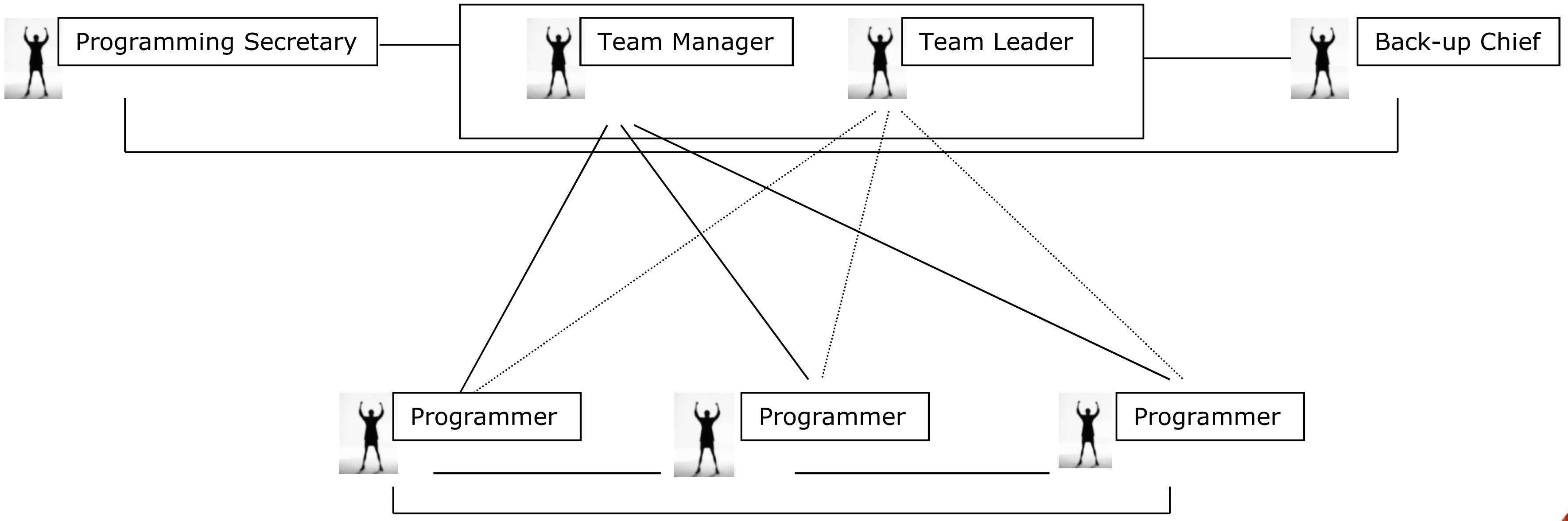
- To solve the Chief problem (hard to find), divide the responsibilities into 2 persons:
  - Team Leader: for Technical decisions
  - Team Manager: for Managerial decisions



Communication for modified chief programmer team approach

MODERN CHIEF PROGRAMMER TEAM

- To solve the communication problem of Chief Programming, the members on the same level and under the same local chief may communicate.



Communication for modern chief programmer team approach

EXTREME PROGRAMMING TEAMS

- The most interesting property: PAIR PROGRAMMING
- Reasons (Advantages)
  - Paralel processing: Writing test cases+ codes
  - When a programmer leaves project, the pair may continue, without interruption
  - A less experienced developer will increase his skills faster
  - Extreme programming teams will work together and this gets group ownership

QUESTIONS?

CE 310–Software Engineering

Lectuer Dr. Osman AKBULUT