SOFTWARE ENGINEERING

LECTURE:
SOFTWARE METRICS-
SIZE ORIENTED & FUNCTION ORIENTED

Lecture Objectives

- To understand the importance of measurement in software engineering
- To describe and compare the different metrics that can be used for measuring software
- To understand the important factors that affect the measurement of software

Software Metrics

Software Metrics: What and Why ?

- How to measure the size of a software?
- How much will it cost to develop a software?
- How many bugs can we expect?
- When can we stop testing?
- When can we release the software?

Software Metrics

- What is the complexity of a module?
- What is the module strength and coupling?
- What is the reliability at the time of release?
- Which test technique is more effective?
- Are we testing hard or are we testing smart?
- Do we have a strong program or a week test suite?

Why Do We Measure?

- To indicate the quality of the product
- To assess the productivity of the people who produce the product
- To assess the benefits derived from new software engineering methods and tools
- To form a baseline for estimation
- To help justify requests for new tools or additional training

Definitions

- *Measure* - quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process.
  - E.g., Number of errors
- *Measurement* - the act of determining a measure
- *Metric* - quantitative measure of degree to which a system, component or process possesses a given attribute. "A handle or guess about a given attribute."
  - E.g., Number of errors found per person hours expended

Motivation for Metrics

- Estimate the cost & schedule of future projects
- Evaluate the productivity impacts of new tools and techniques
- Establish productivity trends over time
- Improve software quality
- Forecast future staffing needs
- Anticipate and reduce future maintenance needs

Software Metrics

Areas of Applications

- The most established area of software metrics is cost and size estimation techniques.
- The prediction of quality levels for software, often in terms of reliability, is another area where software metrics have an important role to play.
- The use of software metrics to provide quantitative checks on software design is also a well established area.

Software Metrics

Categories of Metrics

- **Product metrics:** describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability, etc.
- **Process metrics:** describe the effectiveness and quality of the processes that produce the software product.

  Examples are:
  - effort required in the process
  - effectiveness of defect removal during development
  - number of defects found during testing
  - maturity of the process

Software Metrics

Categories of Metrics

- **Project metrics:** describe the project characteristics and execution. Examples are :
  - number of software developers
  - staffing pattern over the life cycle of the software
  - cost and schedule
  - productivity

Types of Measures

- **Direct Measures (internal attributes)**
  - Cost, effort, LOC, speed, memory
- **Indirect Measures (external attributes)**
  - Functionality, quality, complexity, efficiency, reliability, maintainability

Example: length of a pipe is a direct measure.

the quality of the pipes can only be measured indirectly by finding the ratio of the accepted pipes to the rejected.

Size Oriented Metrics

Size-Oriented Metrics

- Based on the “size” of the software produced.
- Project data measured, including cost and effort, pages, defects etc.
- Mainly uses the LOC as the normalization value.
- Advantages: easily counted, large body of literature and data based on LOC
- Disadvantages: language dependent, programmer dependent.
- Useful for projects with similar environment.
- Therefore, size-oriented metrics are not universally accepted as the best way to measure the software process.

Size-Oriented Metrics

- Size of the software produced
- *LOC* - Lines Of Code
- *KLOC* - 1000 Lines Of Code
- *SLOC* – Statement Lines of Code (ignore whitespace)
- Typical Measures:
  - Errors/KLOC
  - Defects/KLOC
  - Cost/LOC
  - Documentation Pages/KLOC

Example of Project Measurements

project	effort (person-month)	cost ($)	kLOC	Doc. (pgs)	errors	people
Proj_1	24	168,000	12.1	365	29	3
Proj_2	62	440,000	27.2	1224	86	5

Example of Size-Oriented Metrics

- Productivity = Size / Effort
  = kLOC / person-month
- Quality = Errors / Size
  = Errors / kLOC
- Cost = $ / kLOC
- Documentation = pages / kLOC
- Other metrics can also be developed like: errors/KLOC, page/KLOC…etc.
- Or errors/person-month, LOC/person-month, cost/page.

Function Oriented Metrics

Function-Oriented Metrics

- Based on "functionality" delivered by the software as the normalization value.
- Functionality is measured indirectly
- **Function points (FP) measure-** derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity

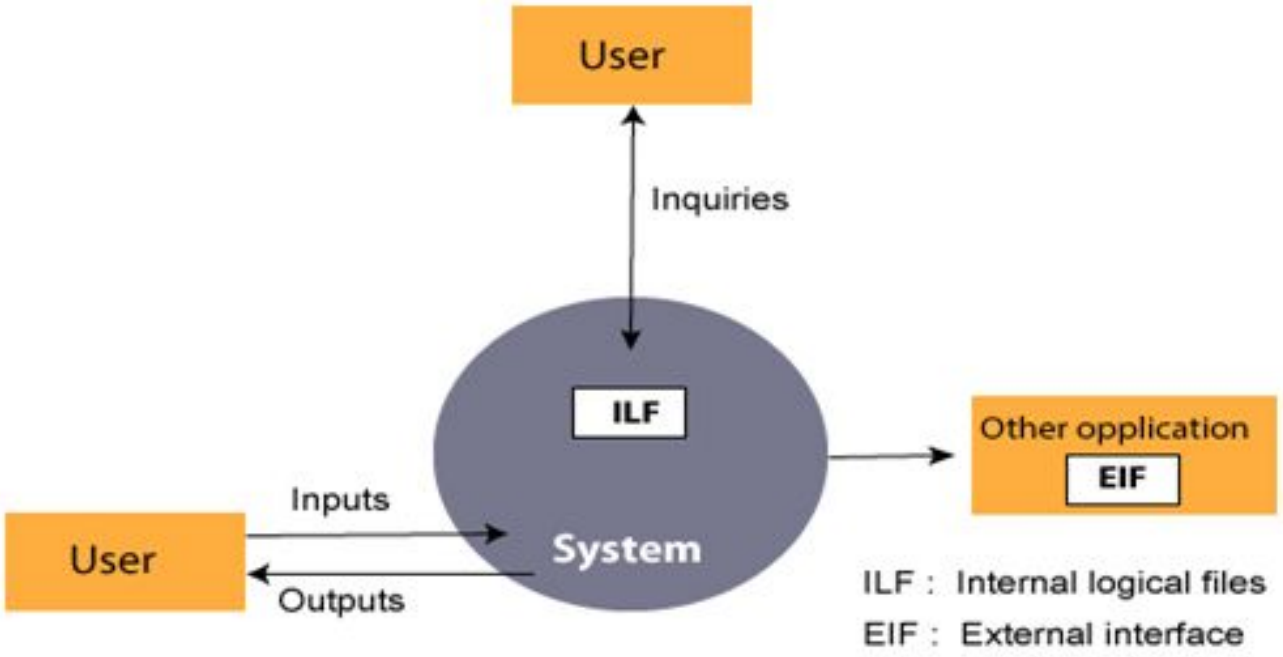
Alan Albrecht while working for IBM, recognized the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem.

The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

- Inputs : information entering the system
- Outputs : information leaving the system
  (reports, screens, error messages)
- Enquiries : requests for instant access to information

  e.g. an on-line input that generates an immediate on-line output response
- Internal logical files : information held within the system
- External interface files : information held by other system that is used by the system being analyzed.

Measurements Parameters	Examples
1. Number of External Inputs(EI)	Input screen and tables
2. Number of External Output (EO)	Output screens and reports
3. Number of external inquiries (EQ)	Prompts and interrupts.
4. Number of internal files (ILF)	Databases and directories
5. Number of external interfaces (EIF)	Shared databases and shared routines.



FPAs Functional Units System

Steps In Calculating FP

1. Count the information domain values.
2. Assess the complexity of the values.
3. Calculate the raw FP (see next table).
4. Rate the complexity factors to produce the complexity adjustment value (CAV)

   CAV = $\sum F_i$ i = 1 to 14

5. Calculate the adjusted FP as follows:

   FP = raw FP x [0.65 + 0.01 x CAV]

Counting function points

Functional Units	Weighting factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
External logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

Table 1 : Functional units with weighting factors

Table 2: UFP calculation table

Functional Units	Count Complexity		Complexity Totals	Functional Unit Totals
External Inputs (EIs)		Low x 3	=	
		Average x 4	=	
		High x 6	=	
External Outputs (EOs)		Low x 4	=	
		Average x 5	=	
		High x 7	=	
External Inquiries (EQs)		Low x 3	=	
		Average x 4	=	
		High x 6	=	
External logical Files (ILFs)		Low x 7	=	
		Average x 10	=	
		High x 15	=	
External Interface Files (EIFs)		Low x 5	=	
		Average x 7	=	
		High x 10	=	
Total Unadjusted Function Point Count				

The weighting factors are identified for all functional units and multiplied with the functional units accordingly. The procedure for the calculation of Unadjusted Function Point (UFP) is given in table shown above.

The procedure for the calculation of UFP in mathematical form is given below:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

Where i indicate the row and j indicates the column of Table 1

W_{ij} : It is the entry of the i^{th} row and j^{th} column of the table 1

Zij : It is the count of the number of functional units of Type *i* that have been classified as having the complexity corresponding to column *j*.

Organizations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to [0.65 + 0.01 x ΣF_i]. The F_i (i=1 to 14) are the degree of influence and are based on responses to questions given in next slide

Table 3 : Computing function points.

Rate each factor on a scale of 0 to 5.

Number of factors considered (F_i)

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

Complexity Adjustment Value

- The rating for all the factors, F_1 to F_{14}, are summed to produce the complexity adjustment value (CAV)
- CAV is then used in the calculation of the function point (FP) of the software

Example of Function-Oriented Metrics

- Productivity = Functionality / Effort
  = FP / person-month
- Quality = Errors / Functionality
  = Errors / FP
- Cost = $ / FP
- Documentation = pages / FP

Example 1

Consider a project with the following functional units:

Number of user inputs = 50

Number of user outputs = 40

Number of user enquiries = 35

Number of user files = 06

Number of external interfaces = 04

Assume all complexity adjustment factors and weighting factors are average. Compute the function points for the project.

Solution

We know

$$UFP = \sum_{i=1}^{5}\sum_{J=1}^{3} Z_{ij} w_{ij}$$

UFP = 50 x 4 + 40 x 5 + 35 x 4 + 6 x 10 + 4 x 7

= 200 + 200 + 140 + 60 + 28 = 628

CAF = $(0.65 + 0.01\ \Sigma F_i)$

= (0.65 + 0.01 (14 x 3)) = 0.65 + 0.42 = 1.07

FP = UFP x CAF

= 628 x 1.07 = 672

Example 2

An application has the following:

10 low external inputs, 12 high external outputs, 20 low internal logical files, 15 high external interface files, 12 average external inquiries, and a value of complexity adjustment factor of 1.10.

What are the unadjusted and adjusted function point counts ?

Solution

Unadjusted function point counts may be calculated using as:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

= 10 x 3 + 12 x 7 + 20 x 7 + 15 + 10 + 12 x 4

= 30 + 84 +140 + 150 + 48

= 452

FP = UFP x CAF

= 452 x 1.10 = 497.2.

Example 3

Consider a project with the following parameters.

(*i*) External Inputs:

(a) 10 with low complexity

(b) 15 with average complexity

(c) 17 with high complexity

(*ii*) External Outputs:

(d) 6 with low complexity

(e) 13 with high complexity

(*iii*) External Inquiries:

(a) 3 with low complexity

(b) 4 with average complexity

(c) 2 high complexity

(*iv*) Internal logical files:

(a) 2 with average complexity

(b) 1 with high complexity

(v) External Interface files:

(c) 9 with low complexity

In addition to above, system requires

i. Significant data communication

ii. Performance is very critical

iii. Designed code may be moderately reusable

iv. System is not designed for multiple installation in different organizations.

Other complexity adjustment factors are treated as average. Compute the function points for the project.

Software Project Planning

Table 2: UFP calculation table

Functional Units	Count Complexity		Complexity Totals	Functional Unit Totals
External Inputs (EIs)		Low x 3	=	
		Average x 4	=	
		High x 6	=	
External Outputs (EOs)		Low x 4	=	
		Average x 5	=	
		High x 7	=	
External Inquiries (EQs)		Low x 3	=	
		Average x 4	=	
		High x 6	=	
External logical Files (ILFs)		Low x 7	=	
		Average x 10	=	
		High x 15	=	
External Interface Files (EIFs)		Low x 5	=	
		Average x 7	=	
		High x 10	=	
Total Unadjusted Function Point Count				

Software Project Planning

Solution: Unadjusted function points may be counted using table 2

Functional Units	Count	Complexity		Complexity Totals	Functional Unit Totals
External Inputs (EIs)	10	Low x 3	=	30	192
	15	Average x 4	=	60	
	17	High x 6	=	102	
External Outputs (EOs)	6	Low x 4	=	24	115
	0	Average x 5	=	0	
	13	High x 7	=	91	
External Inquiries (EQs)	3	Low x 3	=	9	37
	4	Average x 4	=	16	
	2	High x 6	=	12	
External logical Files (ILFs)	0	Low x 7	=	0	35
	2	Average x 10	=	20	
	1	High x 15	=	15	
External Interface Files (EIFs)	9	Low x 5	=	45	45
	0	Average x 7	=	0	
	0	High x 10	=	0	
Total Unadjusted Function Point Count					424

Software Project Planning

$$\sum_{i=1}^{14} F_i = 3+4+3+5+3+3+3+3+3+3+2+3+0+3=41$$

CAF = $(0.65 + 0.01 \times \Sigma F_i)$

= (0.65 + 0.01 x 41)

= 1.06

FP = UFP x CAF

= 424 x 1.06

= 449.44

Hence | FP = 449 |

Example 4

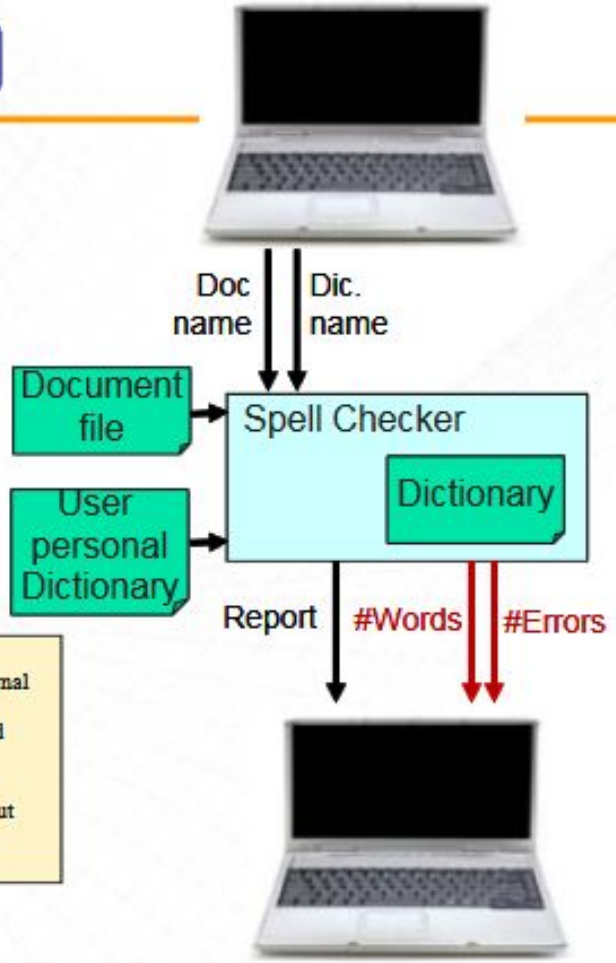
Specification: Spell Checker

- Checks all words in a document by comparing them to a list of words in the internal dictionary and an optional user-defined dictionary
- After processing the document sends a report on all misspelled words to standard output
- On request from user shows number of words processed on standard output
- On request from user shows number of spelling errors detected on standard output
- Requests can be issued at any point in time while processing the document file

FP Example /1d

Solution A:

- EI: Doc. name + User Dic. name → 2
- EO: Report + #Words + #Errors → 3
- EQ: --
- EIF: Document + User Dictionary → 2
- ILF: Dictionary → 1



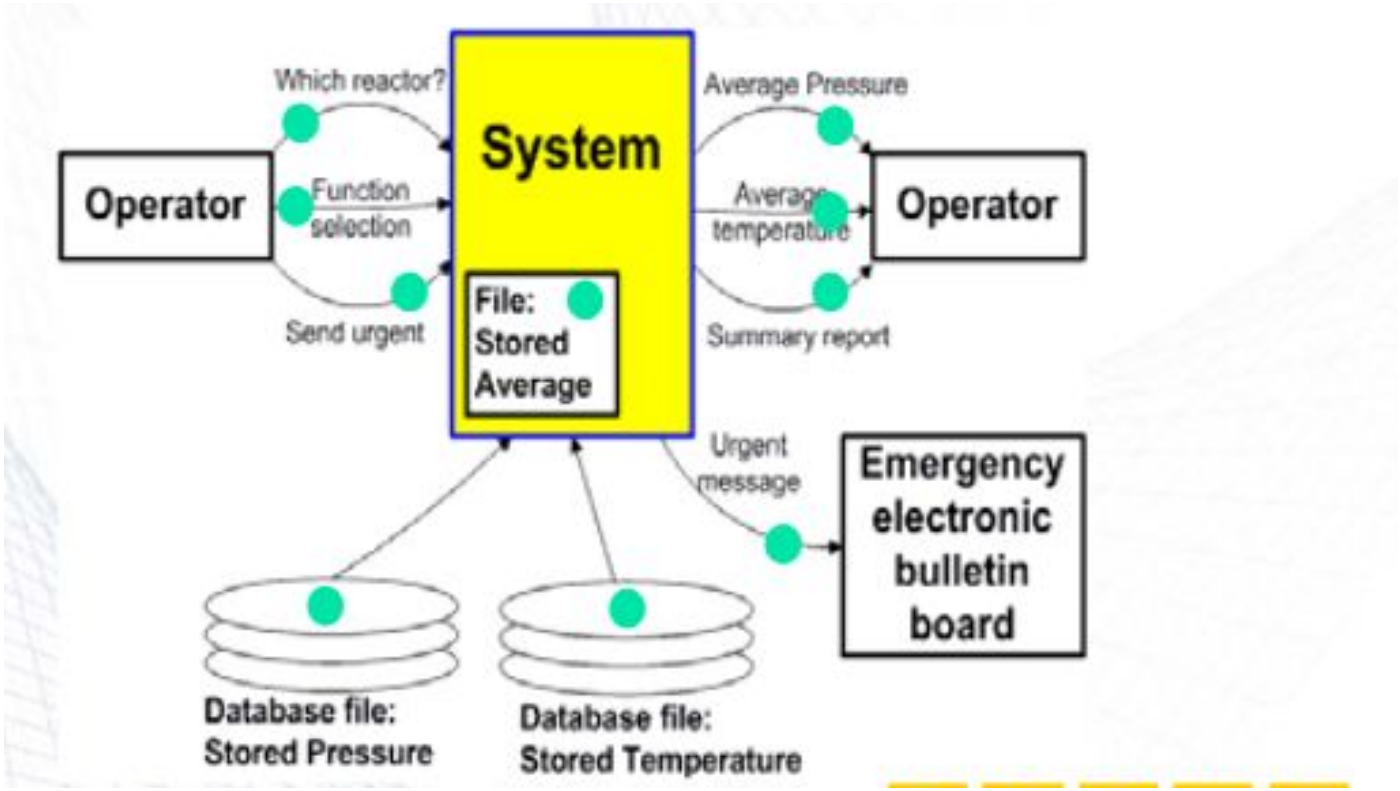
Specification:

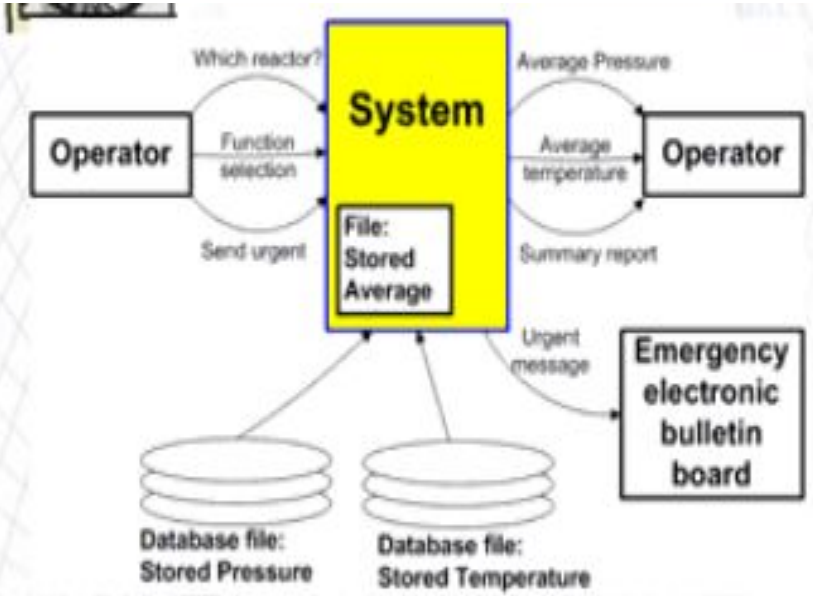
1. Checks all words in a document by comparing them to a list of words in the internal dictionary and an optional user-defined dictionary
2. After processing the document sends a report on all misspelled words to standard output
3. On request from user shows number of words processed on standard output
4. On request from user shows number of spelling errors detected on standard output
5. Requests can be issued at any point in time while processing the document file

EI EO EQ EIF ILF

Example 5

- Consider the following system that processes various commands from the operator of a chemical plant. Various interactions of the system with the operator and internal and external files are shown.
- The system consults two databases for stored pressure and temperature readings. In the case of emergency, the system asks the user whether to send the results to an electronic emergency bulletin board or not.
- Calculate the unadjusted function point count (UFC) for this system. Consider average weighting factors.


Which reactor?
Operator
Function selection
Send urgent
System
File:
Stored
Average
Average Pressure
Average temperature
Summary report
Operator
Urgent message
Emergency electronic bulletin board
Database file:
Stored Pressure
Database file:
Stored Temperature



- N_{EI} number of external inputs **2**
- N_{EO} number of external outputs **3**
- N_{EQ} number of external inquiries **1**
- N_{EIF} number of external interface files **2**
- N_{ILF} number of internal logical files **1**

$$UFC = 4N_{EI} + 5N_{EO} + 4N_{EQ} + 7N_{EIF} + 10N_{ILF}$$

UFC = 4x2 + 5x3 + 4x1 + 7x2 + 10x1 = 51

FP Characteristics

- Advantages: language independent, based on data known early in project, good for estimation
- Disadvantages: calculation complexity, subjective assessments, FP has no physical meaning (just a number)

Extended Function Point Metrics

- **Feature points**
  - A major shortcoming of the function point measure is that it does not take into account the algorithmic complexity of a function.
  - FP only considers the number of functions that the system supports, without distinguishing the difficulty levels of developing the various functionalities. To overcome this problem, an extension to the function point metric called **feature point metric** has been proposed.
  - Feature point metric incorporates **algorithm complexity** as an extra parameter. This parameter ensures that the computed size using the feature point metric reflects the fact that higher the complexity of a function, the greater the effort required to develop it- therefore, it should have larger size compared to a simpler function.
  - Function points were originally designed to be applied to **business information systems.** Extensions have been suggested called feature points which may enable this measure to be applied to **other software engineering applications.**