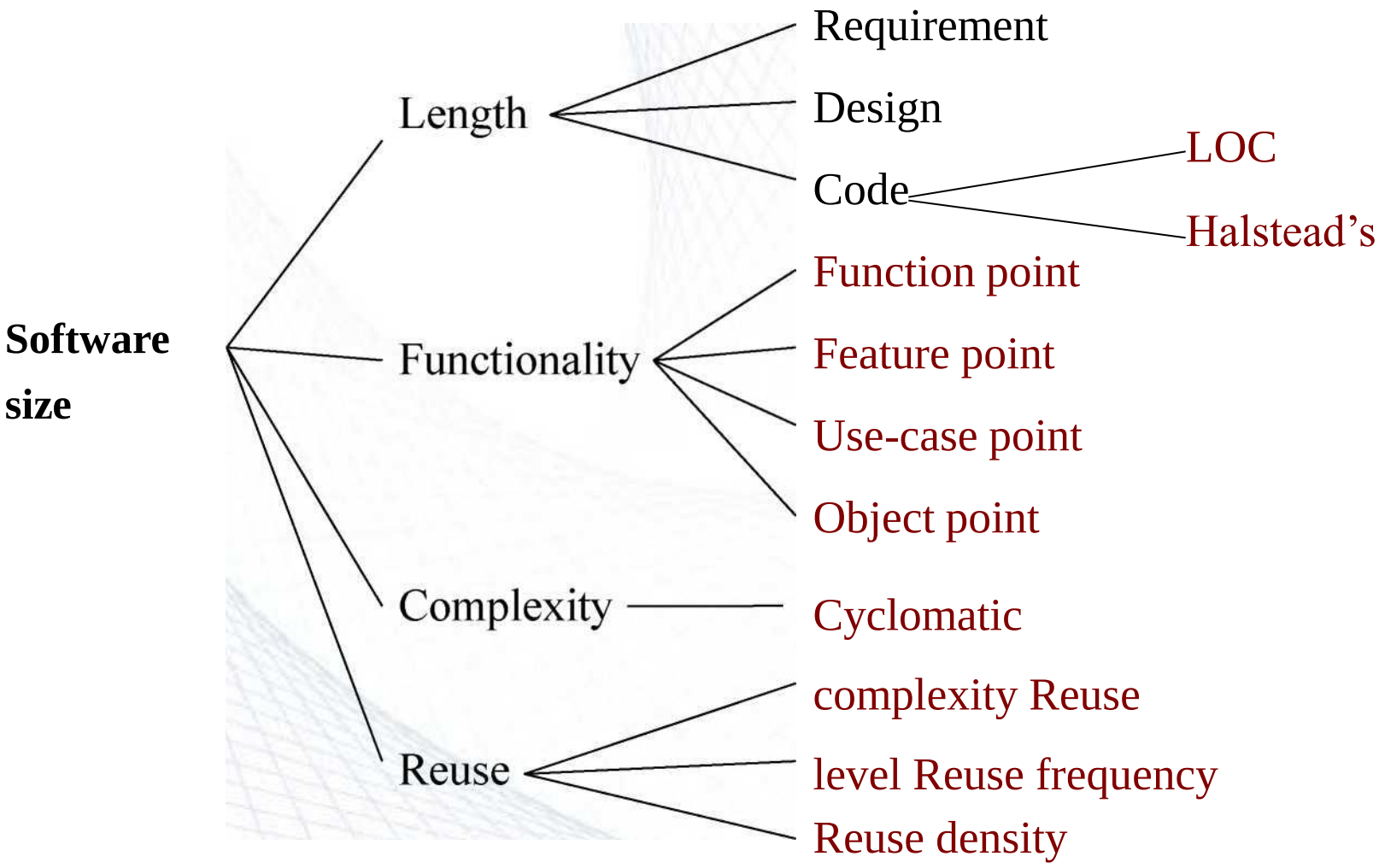
Chapter 5

Measuring Internal Product Attributes: Software Size

Contents

- Software size
- Size: Length (code, specification, design)
- Size: Reuse
- Size: Functionality (function point, feature point, object point, use-case point)
- Size: Complexity

Software Size Metrics: Summary



Software Size

- Internal product attributes describe a software product in a way that is dependent only on the product itself.
- One of the most useful attributes is the **size** of a sofware product, which can be measured statically, i.e., without executing the system.
- It is necessary to define software size in terms of more than one internal attributes, each capturing a key aspect of software size.
- Size measurement must reflect *effort*, *cost* and *productivity*.

Software Size: Length

- Length is the “physical size” of the product.
- In a software development effort, there are three major development products: **specification**, **design**, and **code**.
- The length of the specification can indicate how long the design is likely to be, which in turn is a predictor of code length.
- Traditionally, code length refers to text-based code length.

Length: Code - LOC

- The most commonly used measure of source code program length is the number of lines of code (LOC).
  - ***NCLOC:*** non-commented source line of code or effective lines of code (ELOC).
  - ***CLOC:*** commented source line of code.
- By measuring NCLOC and CLOC separately we can define:

$$total\ length\ (LOC) = NCLOC + CLOC$$

- The ratio: *CLOC/LOC* measures the density of comments in a program.

Length: Code - LOC *(cont.)*

LOC variations:

- Count of physical lines including blank lines.
- Count of all lines except blank lines and comments.
- Count of all statements except comments (statements taking more than one line count as only one line).
- Count of all lines except blank lines, comments, declarations and headings.
- Count of only executable statements, not including exception conditions.

Length: Code - LOC *(cont.)*

Measurement Unit: Lines of Source Code

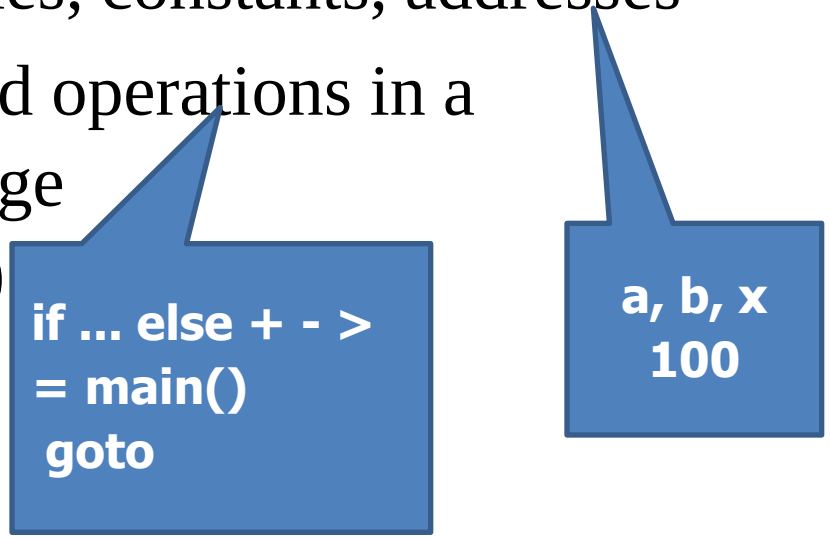
Statement Type		*Includes*	*Excludes*
Executable		X	
Non-executable			
Declarations		X	
Compiler Directives		X	
Comments			X
On their own lines			X
On lines with source code			X
Banners and nonblank spacers			X
Blank (empty) comments			X
Blank Lines			X

Length: Code - LOC *(cont.)*

- Advantages of LOC
  - Simple and automatically measurable
  - Correlates with programming effort (& cost)
- Disadvantage of LOC
  - Vague definition
  - Language dependability
  - Not available for early planning
  - Developers' skill dependability
  - Encouraging "sumo" development!

Length: Halstead's Work

- Maurice Halstead's Theory (1971~1979):
  - A program *P* is a collection of tokens, composed of two basic elements: *operands* and *operators*
    - *Operands* are variables, constants, addresses
    - *Operators* are defined operations in a programming language (language constructs)



Length: Halstead's Work /2

- Number of distinct operators in the program (μ1)
- Number of distinct operands in the program *(*μ2*)*
- Total number of occurrences of operators in the program (N1)
- Total number of occurrences of operands in the program (N2)

Program vocabulary (μ)

$$\mu = \mu 1 + \mu 2$$

Length: Halstead's Work / 3

- Program length is the total number of occurrences of operators and operands:

$$N = N1 + N2$$

- Program volume is the number of mental comparisons needed to write a program of length N

$$\mathrm{V} = \mathrm{N}\log_2 \boldsymbol{\mu} = (\boldsymbol{N1} + \boldsymbol{N2})\log_2(\boldsymbol{\mu 1} + \boldsymbol{\mu 2})$$

- Program level (L): $L = V^*/V$ or $L = \frac{1}{D} = \frac{2}{\mu_1} \times \frac{\mu_2}{N_2}$

- where $\boldsymbol{V^*}$ is the minimum size potential volume (i.e., minimal size of implementation) and $\boldsymbol{D}$ is program difficulty

- Program estimated length $(\hat{N})$

$$\hat{N} = \mu_1 \log_2 \mu_1 + \mu_2 \log_2 \mu_2$$

- Effort required to generate program P: number of elementary discriminations

$$E = V \times D = \frac{V}{L} = \frac{\mu_1 N_2}{2\mu_2} \times N \log_2 \mu$$

- Time required for developing program *P* is the total effort divided by the number of elementary discriminations per second

$$T = E / \beta$$

- In cognitive psychology β is usually a number between 5 and 20
- Halstead claims that $\beta = 18$

- Remaining bugs: the number of bugs left in the software at the delivery time

$$B = E^{2/3} / 3000$$

- Conclusion: the bigger program needs more time to be developed and more bugs remained

Example 1

For the following C program:

```
#include<stdio.h>
main()
{
int a ;
scanf (“%d”, &a);
if ( a >= 10 )
if ( a < 20 )
printf ("10 < a< 20 %d\n" , a);
else
 printf ("a >= 20 %d\n" , a);
else
printf ("a <= 10 %d\n" , a);
}
```

Operands

Example 1 (cont'd)

- Determine number of operators (μ1).
- Determine number of operands (μ2).
- Determine the program length in terms of the total number of occurrences of operators *(N1)* and operands *(N2):*

$$N = N1 + N2$$

- Estimate program length

Example 1 (cont'd)

Operators	Number of occurrences	Operators	Number of occurrences
#	1	<=	1
include	1	\n	3
stdio.h	1	printf	3
< … >	1	<	3
main	1	>=	2
(...)	7	if ... else	2
{ … }	1	&	1
int	1	,	4
;	5	%d	4
scanf	1	"…."	4
$\mu 1 = 20$		$N_1 = 47$	

Example 1 (cont'd)

Operands	Number of occurrences
a	10
10	3
20	3
$\mu 2 = 3$	N2 = 16
$\mu 1 = 20$	$N_1 = 47$
Program length: $N = N_1 + N_2 = 63$	
Program Estimated length:	

$$N = \mu_1 \log_2 \mu_1 + \mu_2 \log_2 \mu_2 = 20 \log_2 20 + 3 \log_2 3 = 91.1934$$

Example 2

- For the following program calculate Halstead's (cl) number of operators; (c2) number of operands; (c3) program vocabulary; (c4) occurrences of operators in the program; (c5) occurrences of operands in the program; (c6) program length; (c7) program volume; (c8) program estimated length.

```
1. read x,y,z;
2. type = "scalene";
3. if (x == y or x == z or y == z) type ="isosceles";
4. if (x == y and x == z) type ="equilateral";
5. if (x >= y+z or y >= x+z or z >= x+y) type ="not a
   triangle";
6. if (x <= 0 or y <= 0 or z <= 0) type ="bad inputs";
7. print type;
```

Example 2 (cont'd)

Operators				Operands	
read	1	==	5	strings	5
,	2	or	6	x	9
;	7	and	1	y	8
" ... "	6	>=	3	z	8
=	5	<=	3	0	3
if	4	+	3	type	6
( )	4	print	1		

$\mu_1 = 14$
$N_1 = 51$

$\mu_2 = 6$
$N_2 = 39$

Example 2 (cont'd)

(cl) Number of distinct operators in the program: $\mu 1 = 14$

(c2) Number of distinct operands in the program: $\mu 2 = 6$

(c3) Program vocabulary: $\mu = \mu 1 + \mu 2 = 20$

(c4) Total number of occurrences of operators in the program:

$N1 = 51$

(c5) Total number of occurrences of operands in the program:

$N2 = 39$

(c6) Program length: $N = Ni + N2 = 90$

(c7) Program volume: $V = N \log_2 \mu = 90 \log_2 (20) = 388.9735$

(c8) Program estimated length:

$$\hat{N} = \mu_1 \log_2 \mu_1 + \mu_2 \log_2 \mu_2 = 14 \log_2 14 + 6 \log_2 6 = 68.81274$$

Critics of Halstead's work

- Developed in the context of assembly languages and too fine grained for modern programming languages.
- The treatment of basic and derived measures is somehow confusing.
- The notions of time to develop and remaining bugs are arguable.
- Unable to be extended to include the size for specification and design.

Length: Alternative Methods

- Alternative methods for text-based measurement of code length:

  1) Source memory size: Measuring length in terms of number of bytes of computer storage required for the program text. Excludes library code.

  2) Char size: Measuring length in terms of number of characters (CHAR) in program text.

  3) Object memory size: Measuring length in terms of an object (executable or binary) file. Includes library code.

- All are relatively easy to measure (or estimate).

Length: Code - Problems /1

- One of the problems with text-based definition of length is that the line of code measurement is increasingly less meaningful as software development turns to more automated tools, such as:
  - Tools that generate code from specifications
  - Visual programming tools
- What can be the length measure for objects that are not textual?
- How can one account for components that are constructed externally? (library, repository, inherited functions, design patterns, etc.)

Length: Code - Problems /2

- The LOC size measurement needs to be replaced by some other size measures, such as:

➢ Object points (as used in COCOMO 2.0), etc.

➢ Measuring length by taking into account the reused portion of code.

Length: Spec. & Design

- Specification and design documents are usually composed of logical text (axioms, constraints, etc.) and diagrams (flow graphs, etc.)
- Define atomic objects to measure textual contents and graphical contents, e.g.,

Diagram	Atomic objects
Data-flow diagram	bubbles
Data dictionary	Data element
Entity-relationship diagram	Objects, relations
State transition diagrams	States, transitions

Length: Prediction

- Theoretically, size of design can be used to predict size of code.
- Each project has a design-to-code expansion ratio. If design is measured by the number of modules of size S_i, then:

$$LOC = \sigma \sum_{i=1}^{m} s_i$$

- σ is the design-to-code expansion ratio recorded for previous similar projects.

Measuring Software Size:

- **Function Point (FP),**
- **Feature Point,**
- **Object Point and**
- **Use-case Point**

Review: Software Size

- Size measurement must reflect ***effort***, ***cost*** and ***productivity***.
- Size-oriented metrics are direct measures of software and the process by which it was developed. These metrics include effort (time), money spent, LOC, pages of documents created, errors, and number of staff.
- Defining software size in terms of length, functionality, and complexity, each capturing a key aspect of software size.
- Basic measure for length is LOC. Simple size-oriented metrics can be generated from LOC, such as:
  - Productivity = KLOC / person-month
  - Quality = defects / KLOC
  - Documentation = pages of documents / KLOC
  - etc.

Length: Problem with LOC

- Depending on the programmer and/or coding standards, the "line of code" could be, and usually is, written on many separate lines:

```
for (i=0; i<100; ++i)
{
        printf("hello");
}   /* Now how many lines of code is this? */
```

- In this example we have:
  - 4 Physical Lines of Code (is placing braces worth to be estimated?)
  - 2 Logical Line of Code (What about all the work writing non-statement lines?)
  - 1 Comment Line (?)

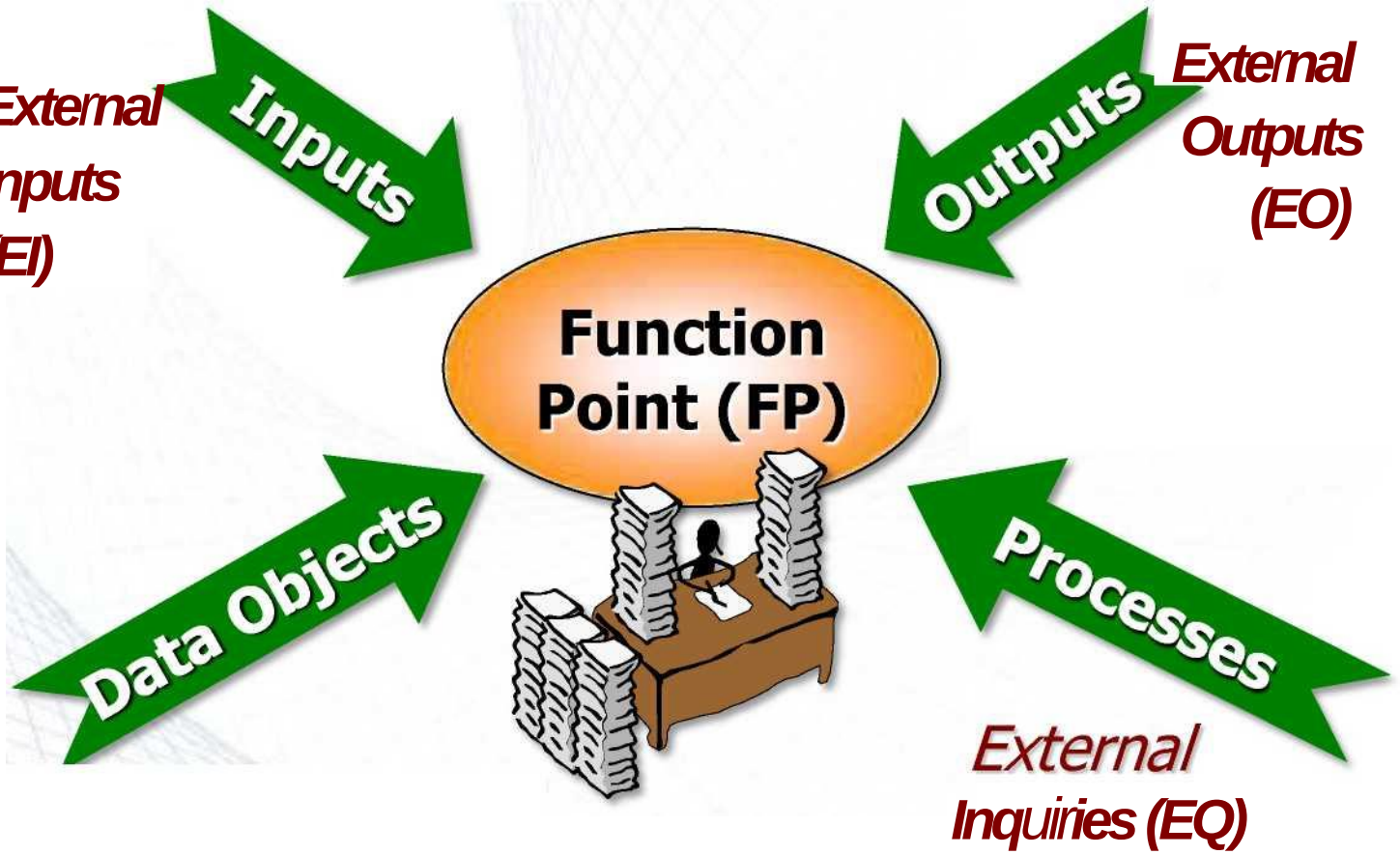
Function-Oriented Metrics

- **Function Point (FP) is a weighted measure of software functionality.**
- The idea is that a product with more functionality will be larger in size.
- Function-oriented metrics are indirect measures of software which focus on functionality and utility.
- The first function-oriented metrics was proposed by Albrecht (1979~1983) who suggested a productivity measurement approach called the **Function Point (FP)** method.
- Function points (FPs) measure the amount of functionality in a system based upon the ***system specification.***
  - Estimation before implementation!

Function Point (FP) Standards

Standard	Name	Content
ISO/IEC 14143-1:2000	IFPUG 4.1	Unadjusted functional size measurement method -- Counting practices manual IFPUG Function Points VS 4.1 (unadjusted)
ISO/IEC 20926:2004	IFPUG 4.2	Unadjusted functional size measurement method -- Counting practices manual IFPUG Function Points VS 4.2 (unadjusted)
ISO/IEC 19761:2003	COSMIC-FFP	A functional size measurement method COSMIC Full Function Points Vs. 2.2
ISO/IEC 20968:2002	Mark II Function Point Analysis	Counting Practices Manual Mark II Function Points

Function Point (FP) /1



External Interface files (EIF)

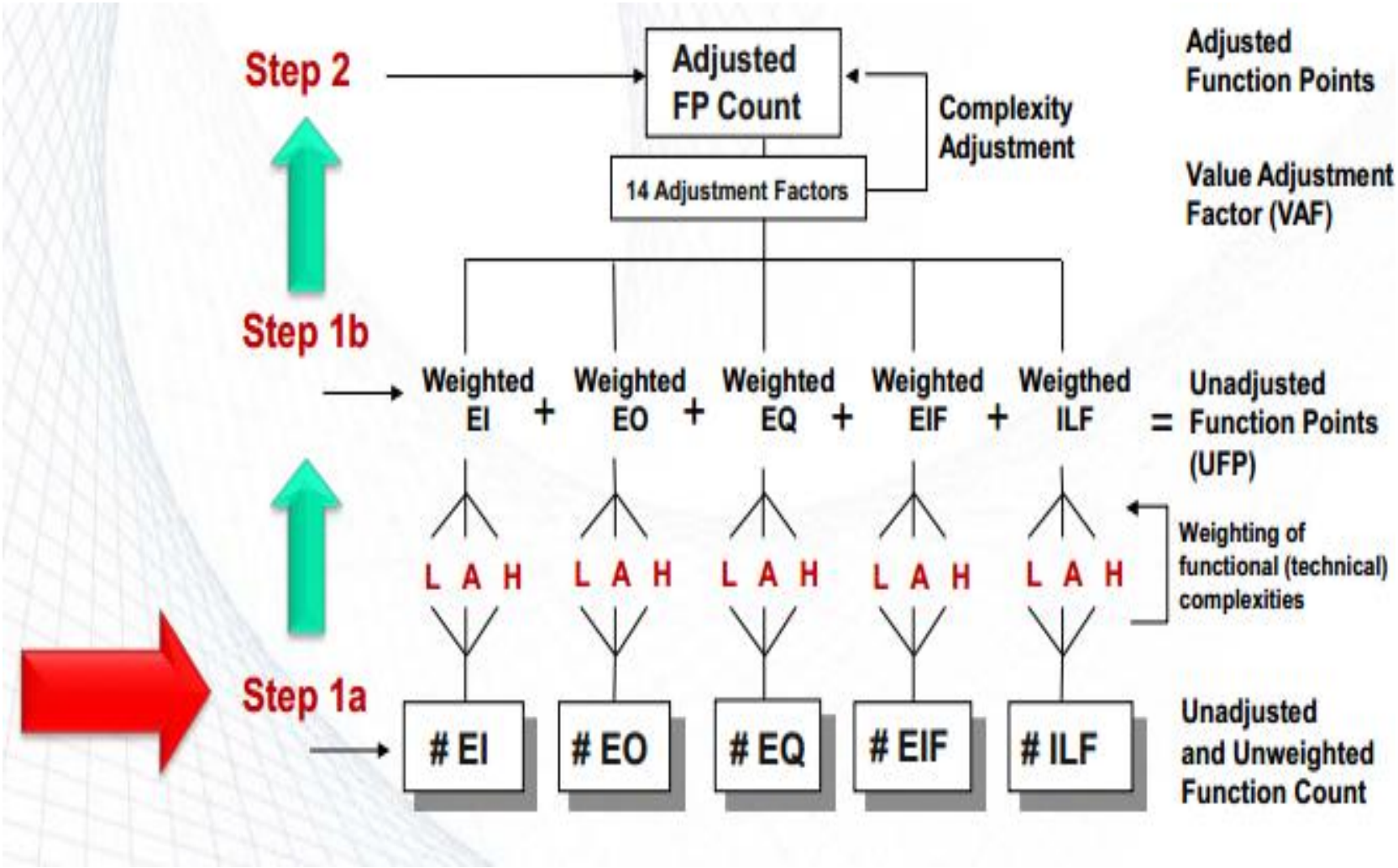
Internal Logic files (ILF)

Function Point (FP) *(cont.)*

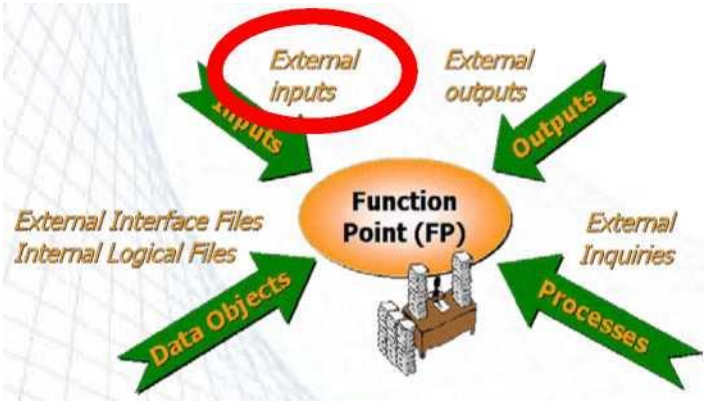
- Function Point (FP) is a weighted measure of software functionality.
- FP is computed in two steps:
  1. Calculating ***Unadjusted Function point Count (UFC)***.
  2. Multiplying the UFC by a ***Value Adjustment Factor (VAF)***
- The final (adjusted) Function Point is:
  $FP = UFC \times VAF$

IFPUG Terminology

FP Counting Summary



1. External Inputs (EI)



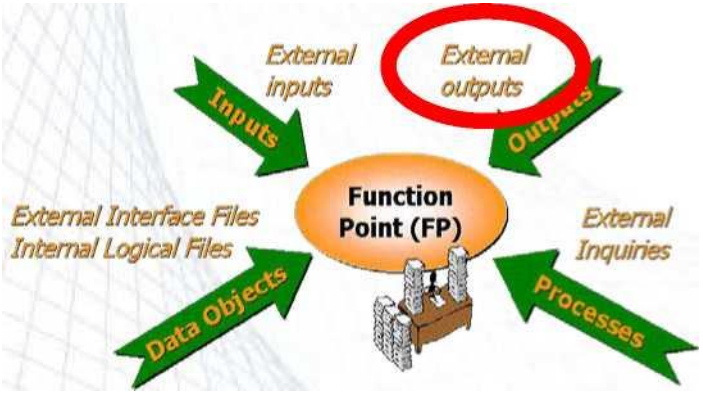
External Inputs - IFPUG Definition:

- An external input (EI) is an elementary process that processes data or control information that comes from outside the application boundary
- The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system

Example:

- Data entry by users
- Data or file feeds by external applications

2. External Outputs (EO)



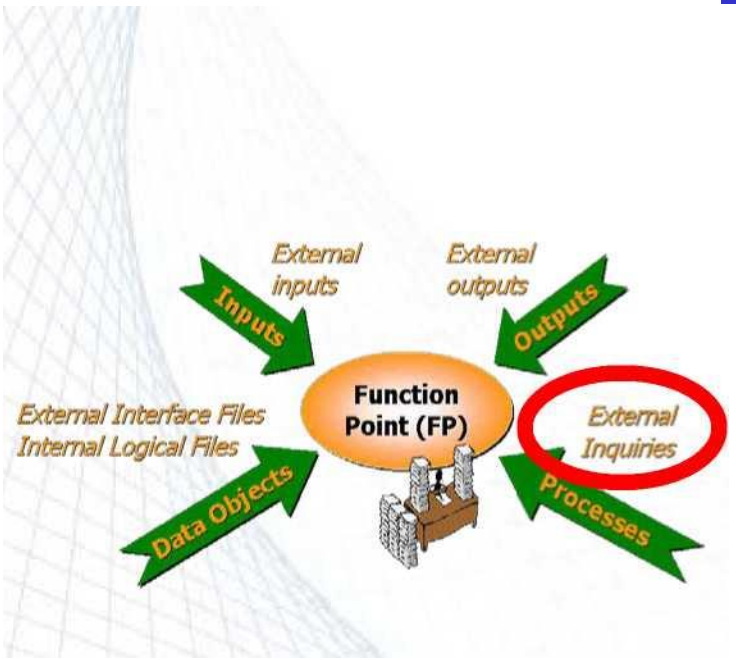
External Outputs - IFPUG Definition:

- An external output (EO) is an elementary process that sends data or control information outside the application boundary
- The primary intent of an external output is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information
- The processing logic must contain at least one mathematical formula or calculation, create derived data, maintain one or more ILFs, or alter the behavior of the system

Example:

- Reports created by the application being counted, where the reports include derived information

3. External Inquiries (EQ)

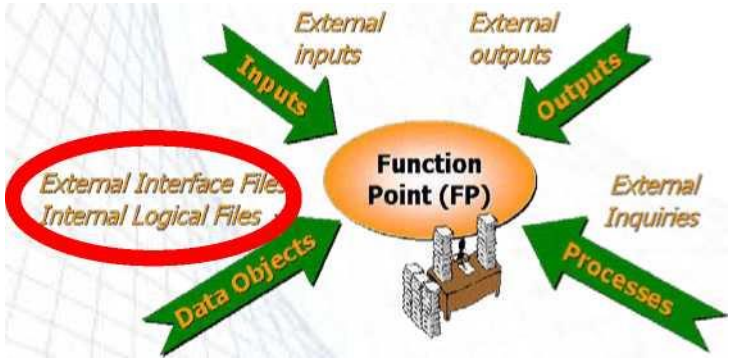


- ***External Inquiries - IFPUG Definition:***
  - An external inquiry (EQ) is an elementary process that sends data or control information outside the application boundary
  - The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF or EIF
  - The processing logic contains no mathematical formulas or calculations, and creates no derived data
  - No ILF is maintained during the processing, nor is the behavior of the system altered

Example:

- Reports created by the application being counted, where the report does not include any derived data

4. Internal Logical Files (ILF)

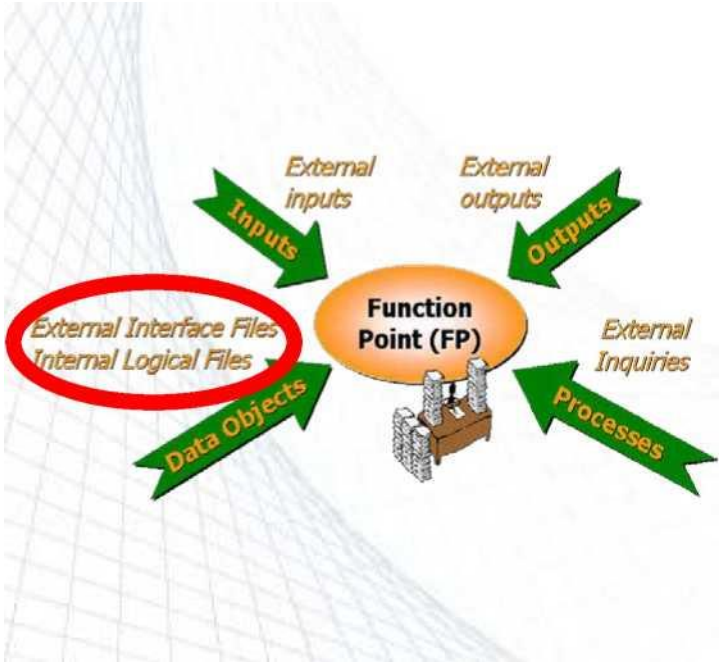


- ***Internal Logical Files - IFPUG Definition:***
  - An ILF is a user-identifiable group of logically related data or control information maintained within the boundary of the application
  - The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted

Example:

- Tables in a relational database
- Application control information, perhaps things like user preferences that are stored by the application

5. Ext. Interface Files (EIF)

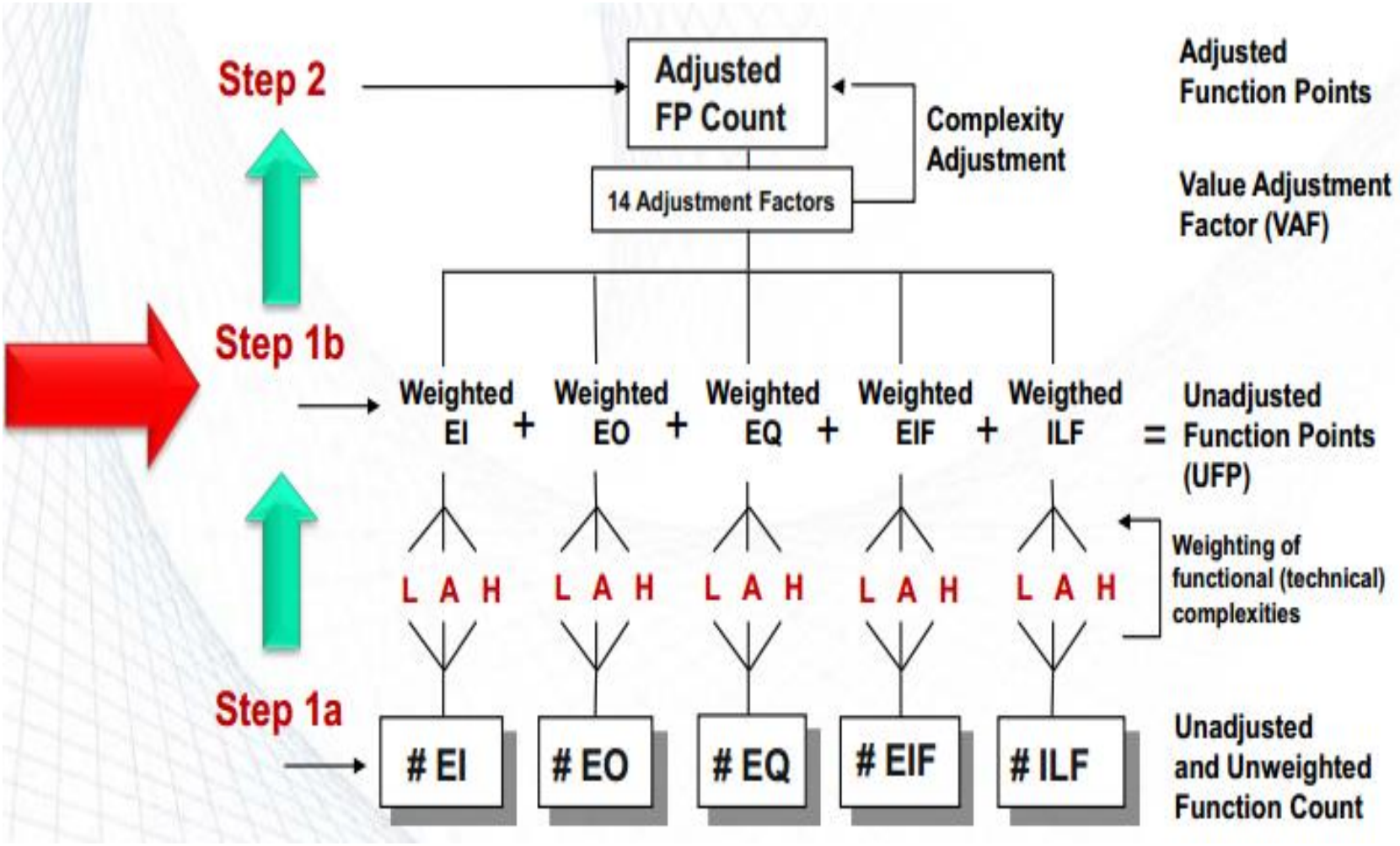


- ***External Interface files -IFPUG Definition:***
  - An external interface file (EIF) is a user identifiable group of logically related data or control information referenced by the application, but maintained within the boundary of another application
  - The primary intent of an EIF is to hold data referenced through one or more elementary processes within the boundary of the application counted
  - This means an EIF counted for an application must be in an ILF in another application

Example:

- As for ILF, but maintained in different systems

FP Counting Summary



Unadjusted FP Count (UFC)

- A complexity rating is associated with each count according to ***function point complexity weights,*** below:

$$UFC = 4\,N_{E1} + 5\,N_{EO} + 4\,N_{EQ} + 7\,N_{EIF} + 10\,N_{ILF}$$

Element	Complexity Weighting Factor		
	Low (Simple)	Average	High (Complex)
External inputs (NEI)	3	4	6
External outputs *(NEO)*	4	5	7
External inquiries *(NEQ)*	3	4	6
External interface files *(NEIF)*	5	7	10
.Internal logical files (NILF)	7	10	15

FP Counting: Weighting of Technical Complexity

Elements	Complexity Weighting Factor			
	low	average	high	Sum
External Inputs (EI)	__ x 3 = _____	__ x 4 = _____	__ x 6 = _____	_____
External Outputs (EO)	__ x 4 = _____	__ x 5 = _____	__ x 7 = _____	_____
External Inquiries (EQ)	__x 3 = _____	__ x 4 = _____	__ x 6 = _____	_____
External Interface Files (EIF)	__ x 5 = _____	__ x 7 = _____	__ x10 = _____	_____
Internal Logical Files (ILF)	__ x 7 = _____	__ x10 = _____	__ x15 = _____	_____
Unadjusted Function Points (UFP)				

Function Types - Complexity Assessment

How to assess complexity?

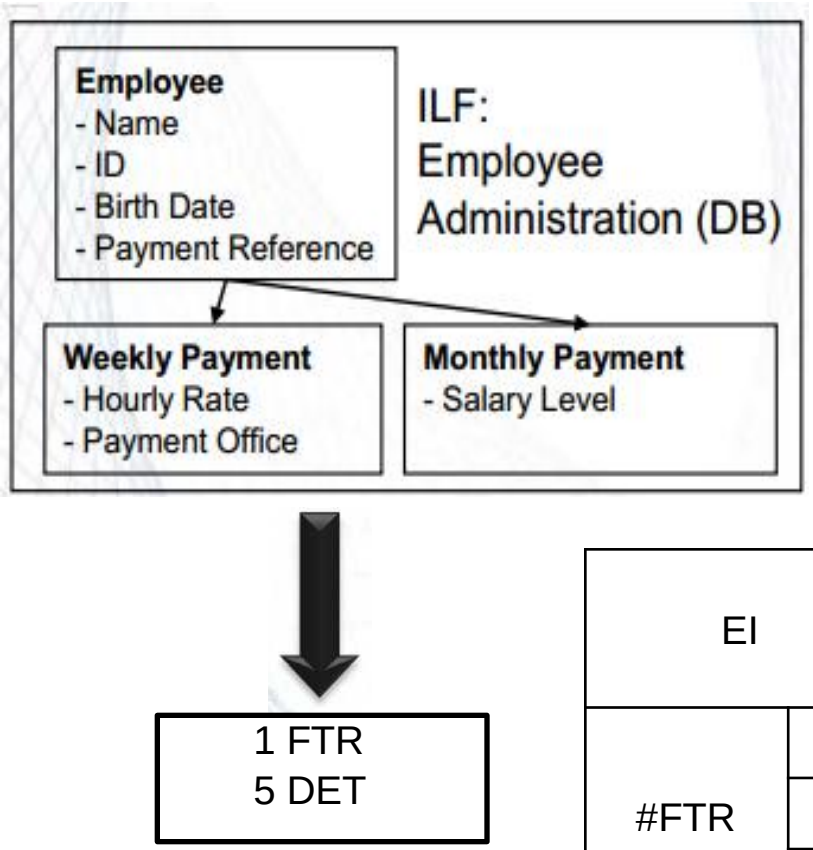
<table>
<tr><th rowspan="2"></th><th colspan="2">Data Function Types</th><th colspan="3">Transaction Function Types</th></tr>
<tr><th>Internal Logical Files (ILF)</th><th>External Interface Files (EIF)</th><th>External Input (EI)</th><th>External Output (EO)</th><th>External Inquiry (EQ)</th></tr>
<tr><td rowspan="2">Elements evaluated for technical complexity assessment</td><td colspan="2">REcord Types (RET): User recognizable sub groups of data elements within an ILF or an EIF. It is best to look at logical groupings of data to help identify them.</td><td colspan="3">File Type Referenced (FTR): File type referenced by a transaction. An FTR must be an Internal Logical File (ILF) or External Interface File (EIF).</td></tr>
<tr><td colspan="5">Data Element Types (DET): A unique user recognizable, non-recursive (non-repetitive) field containing dynamic information. If a DET is recursive then only the first occurrence of the DET is considered not every occurrence.</td></tr>
</table>

Complexity Assessment: EI

- Identify number of File Types Referenced (FTR)
- Identify number of Data Element Type (DET)
- Determine complexity weight (→used to calculate FP count)

EI		#DET		
		1-4	5-15	> 15
#FTR	1	low (3)	low (3)	average (4)
	2	low (3)	average (4)	high (6)
	> 2	average (4)	high (6)	high (6)

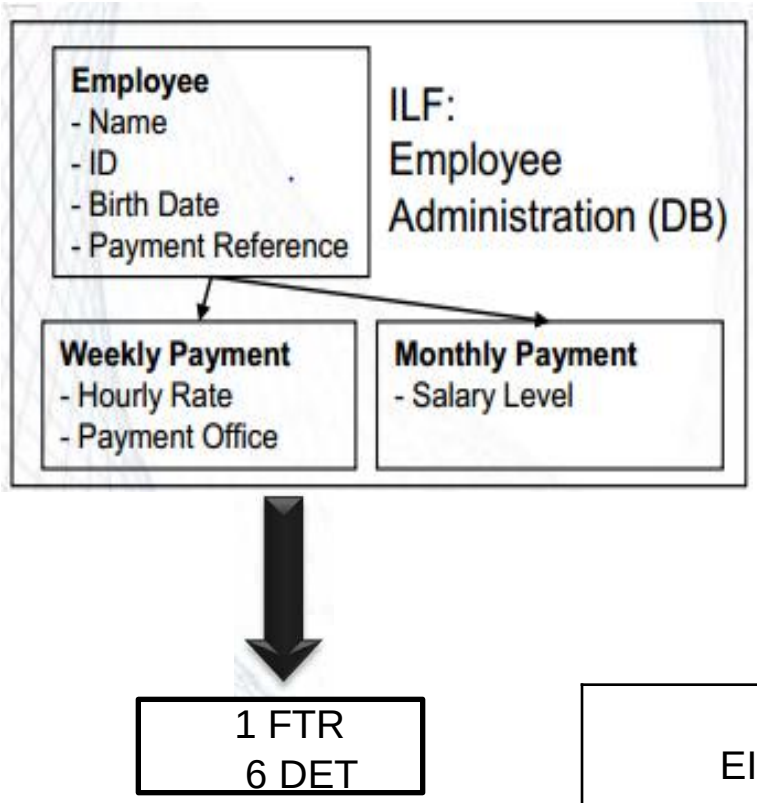
External Input - Example



- Enter a new employee with Monthly Payment:
  - Name
  - ID
  - Birth Date
  - Payment Reference
  - Salary Level

EI		#DET		
		1-4	5-15	> 15
#FTR	1	low (3)	low (3)	average (4)
	2	low (3)	average (4)	high (6)
	> 2	average (4)	high (6)	high (6)

EI - Example



- Enter a new employee with Weekly Payment:
  - Name
  - ID
  - Birth Date
  - Payment
  - Reference
  - Hourly
  - Rate
  - Payment
  - Office

EI		#DET		
		1-4	5-15	> 15
#FTR	1	low (3)	low (3)	average (4)
	2	low (3)	average (4)	high (6)
	> 2	average (4)	high (6)	high (6)

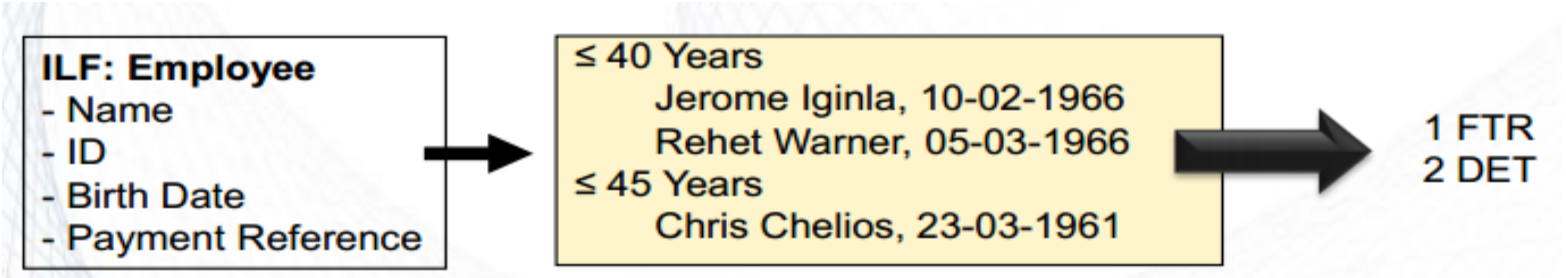
Complexity Assessment: EO

- Identify number of File Types Referenced (FTR)
- Identify number of Data Element Type (DET)
- Determine complexity weight (→ used to calculate FP count)

EO		#DET		
		1-5	6-19	> 19
#FTR	1	low (4)	low (4)	average (5)
	2-3	low (4)	average (5)	high (7)
	> 3	average (5)	high (7)	high (7)

EO - Example

- Report of all Employees containing Names and Birth Dates, sorted by age.



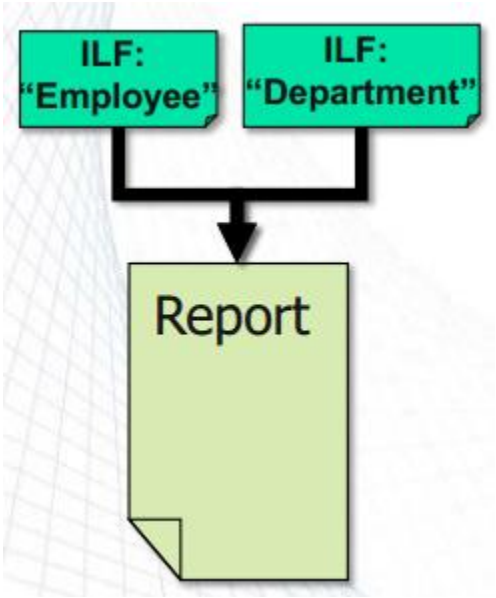
EO		#DET		
		1-5	6-19	> 19
#FTR	1	low (4)	low (4)	average (5)
	2-3	low (4)	average (5)	high (7)
	> 3	average (5)	high (7)	high (7)

Complexity Assessment: EQ

- Identify number of File Types Referenced (FTR)
- Identify number of Data Element Type (DET)
- Determine complexity weight (→used to calculate FP count)

EQ		#DET		
		1-5	6-19	> 19
#FTR	1	low (3)	low (3)	average (4)
	2-3	low (3)	average (4)	high (6)
	> 3	average (4)	high (6)	high (6)

EQ - Example



- Report of all employees belonging to Department X containing Names, Birth Dates, and showing the Department Name
- Files (ILF): Employee, Department
- 2 FTR: Employee, Department
- 3 DET: Name (Employee), Birth Date (Employee), Department Name (Department)

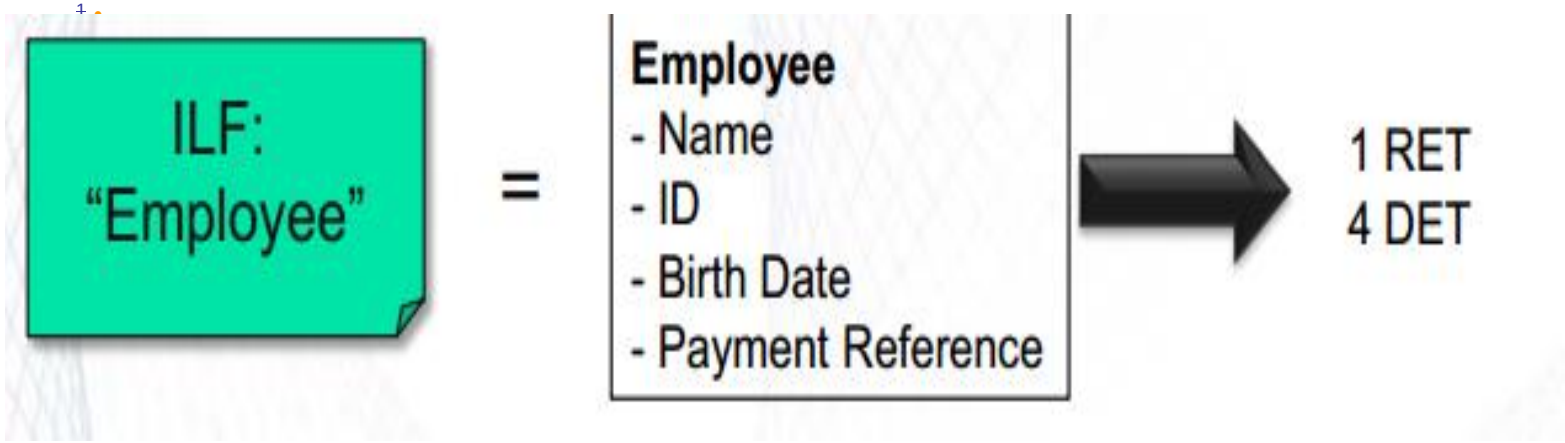
EQ		#DET		
		1-5	6-19	> 19
#FTR	1	low (3)	low (3)	average (4)
	2-3	low (3)	average (4)	high (6)
	> 3	average (4)	high (6)	high (6)

Complexity Assessment: ILF

- Identify number of Record Types (RET)
- Identify number of Data Element Type (DET)
- Determine complexity weight (→used to calculate FP count)

ILF		#DET		
		1-19	**20-50**	**> 50**
#RET	**1**	low (7)	low (7)	average (10)
	2-5	low (7)	average (10)	high (15)
	> 5	average (10)	high (15)	high (15)

ILF - Example



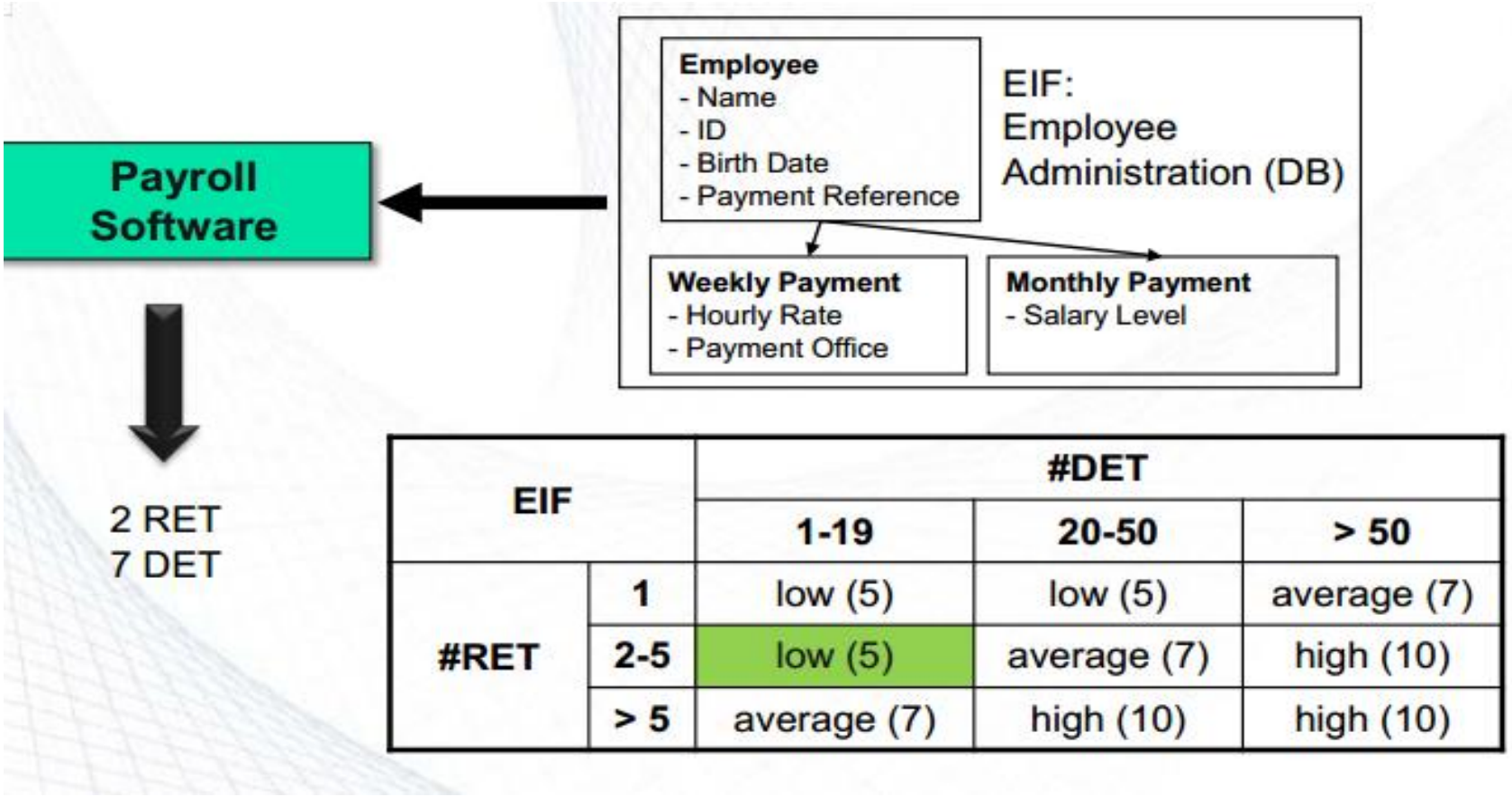
ILF		#DET		
		1-19	20-50	> 50
#RET	1	low (7)	low (7)	average (10)
	2-5	low (7)	average (10)	high (15)
	> 5	average (10)	high (15)	high (15)

Complexity Assessment: EIF

- Identify number of Record Types (RET)
- Identify number of Data Element Type (DET)
- Determine complexity weight (^ used to calculate FP count)

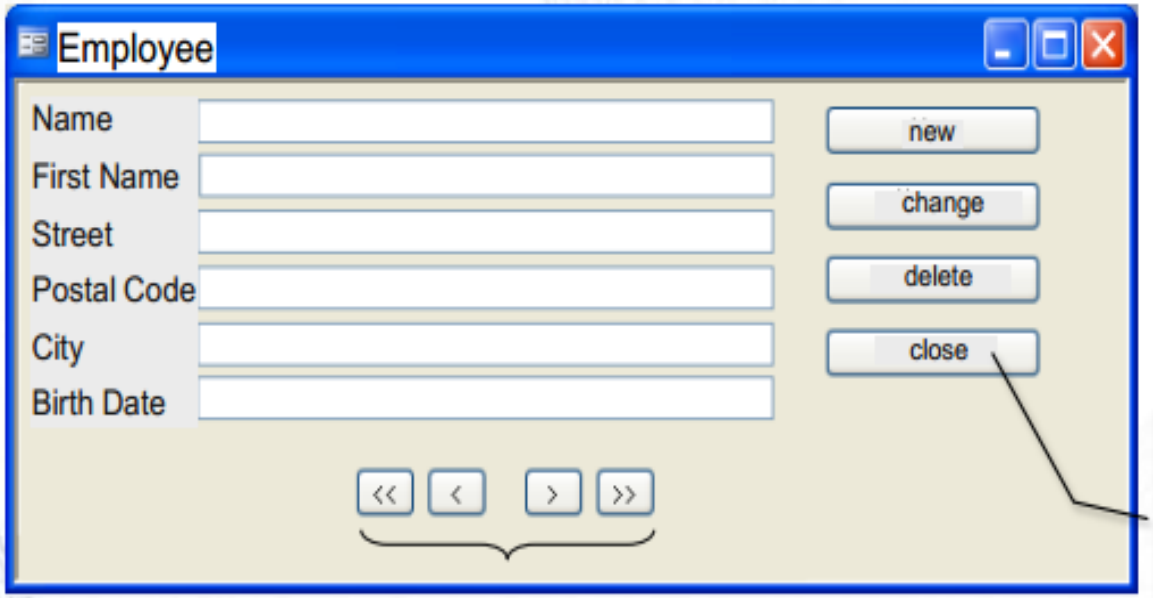
EIF		#DET		
		1-19	20-50	> 50
#RET	1	low (5)	low (5)	average (7)
	2-5	low (5)	average (7)	high (10)
	> 5	average (7)	high (10)	high (10)

EIF - Example



EIF		#DET		
		1-19	20-50	> 50
#RET	1	low (5)	low (5)	average (7)
	2-5	low (5)	average (7)	high (10)
	> 5	average (7)	high (10)	high (10)

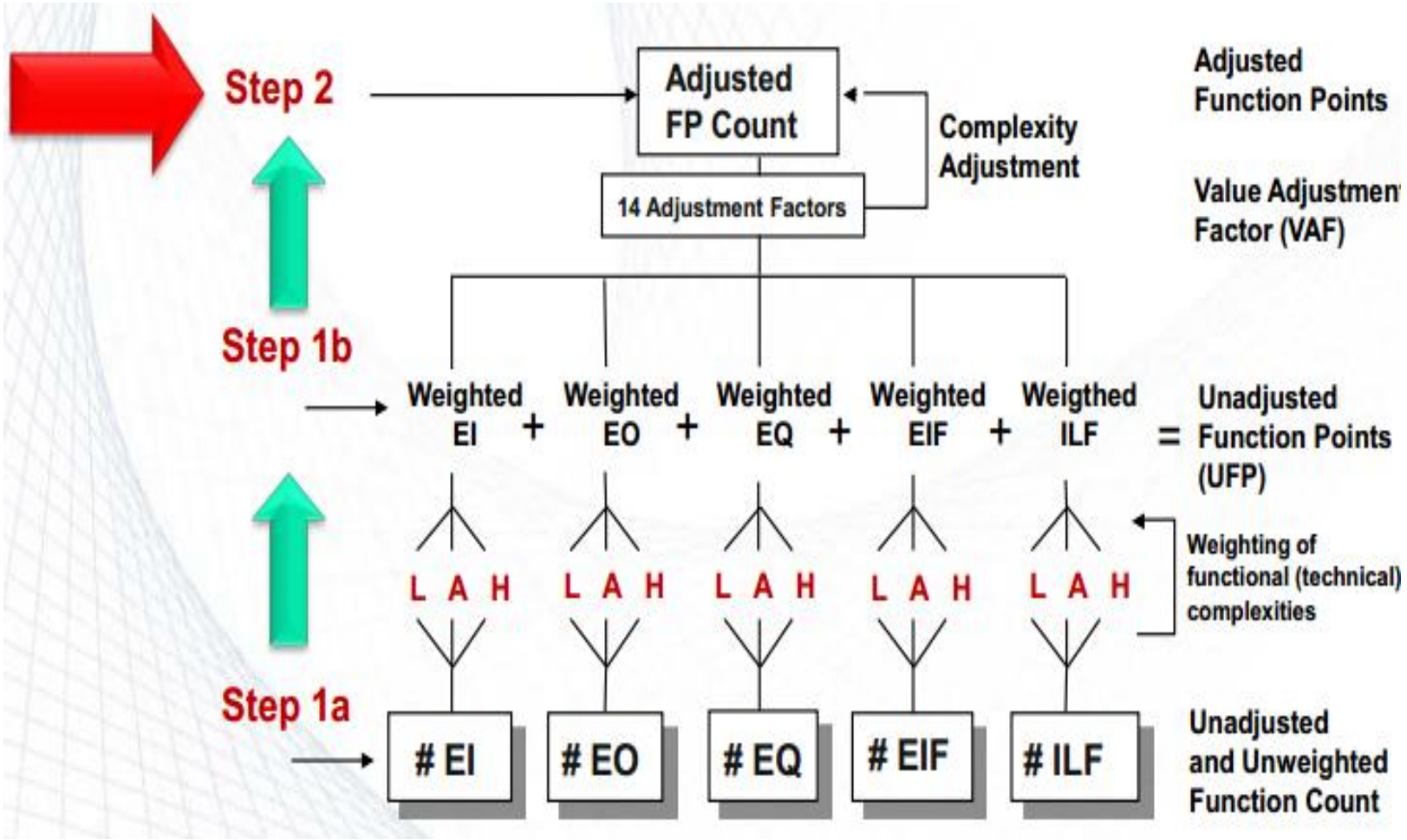
FP Counting Example - GUI



External Input (new, 1 FTR, 7 DET, low) = 3 FP
External Input (change, 1 FTR, 7 DET, low) = 3 FP
External Input (delete, 1 FTR, 7 DET, low) = 3 FP
External Inquiry (navigate, 1 FTR, 7 DET, low) = 3 FP

12 FP

FP Counting Summary



Value Adjustment Factor (VAF) /1

- Value Adjustment Factor *(VAF)* (aka. Technical complexity factors) is a weighted sum of 14 components, given below:

F1	Data Communications	F8	On-line Update
F2	Distributed Data Processing	F9	Complex Processing
F3	Performance	F10	Reusability
F4	Heavily Used Configuration	F11	Installation Ease
F5	Transaction Rate	F12	Operational Ease
F6	On-line Data Entry	F13	Multiple Sites
F7	End-User Efficiency	F14	Facilitate Change

Value Adjustment Factor (VAF)

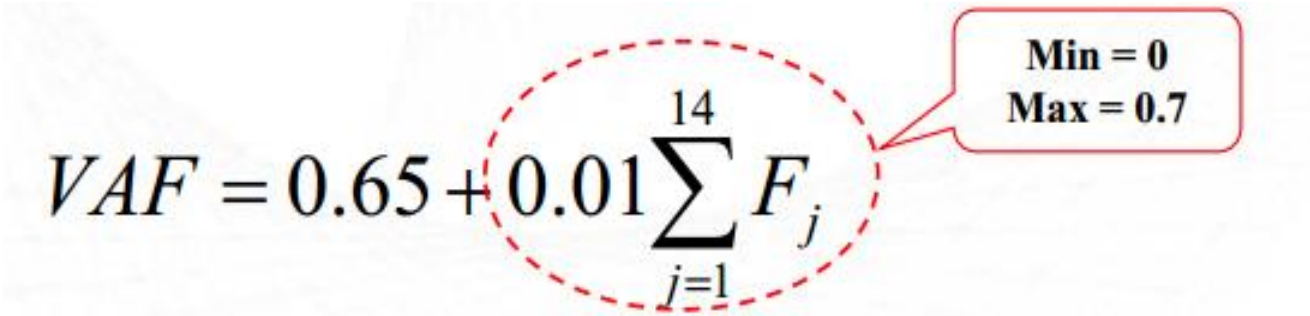
- **Data Communications:** The data and control information used in the application are sent or received over communication facilities.
- **Distributed Data Processing:** Distributed data or processing functions are a characteristic of the application within the application boundary.
- **Performance:** Application performance objectives, stated or approved by the user, in either response or throughput, influence (or will influence) the design, development, installation and support of the application.
- **Heavily Used Configuration:** A heavily used operational configuration, requiring special design considerations, is a characteristic of the application.
- **Transaction Rate:** The transaction rate is high and influences the design, development, installation and support.
- **On-line Data Entry:** On-line data entry and control information functions are provided in the application.
- **End-User Efficiency:** The on-line functions provided emphasize a design for

Value Adjustment Factor (VAF)

- **On-line Update:** The application provides on-line update for the internal logical files.
- **Complex Processing:** Complex processing is a characteristic of the application.
- **Reusability:** The application and the code in the application have been specifically designed, developed and supported to be usable in other applications.
- **Installation Ease:** Conversion and installation ease are characteristics of the application. A conversion and installation plan and/or conversion tools were provided and tested during the system test phase.
- **Operational Ease:** Operational ease is a characteristic of the application. Effective start-up, backup and recovery procedures were provided and tested during the system test phase.
- **Multiple Sites:** The application has been specifically designed, developed and
  supported to be installed at multiple sites for multiple organizations.
- **Facilitate Change:** The application has been specifically designed, developed and supported to facilitate change.

Value Adjustment Factor (VAF) /2

- Each component is rated from 0 to 5, where 0 means the component is not relevant to the system and 5 means the component is essential.
- VAF can then be calculated as:



- VAF varies from 0.65 (if all *Fj* are set to 0) to 1.35 (if all *Fj* are set to 5)
- Final Function Point is: $FP = UFC \times VAF$

Function Points (IFPUG)

Example 1

FP Example /1 a

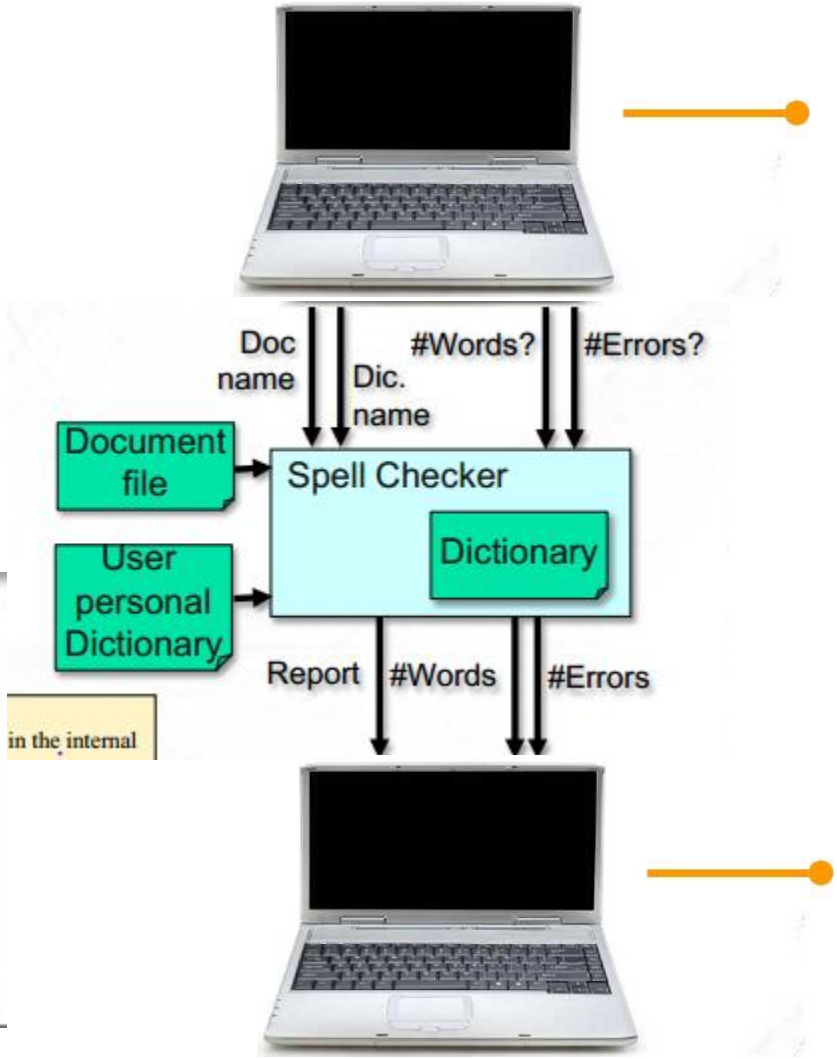
Specification: Spell Checker

- 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 /1 b

Spell Checker

Convert spec to a diagram



Specification:

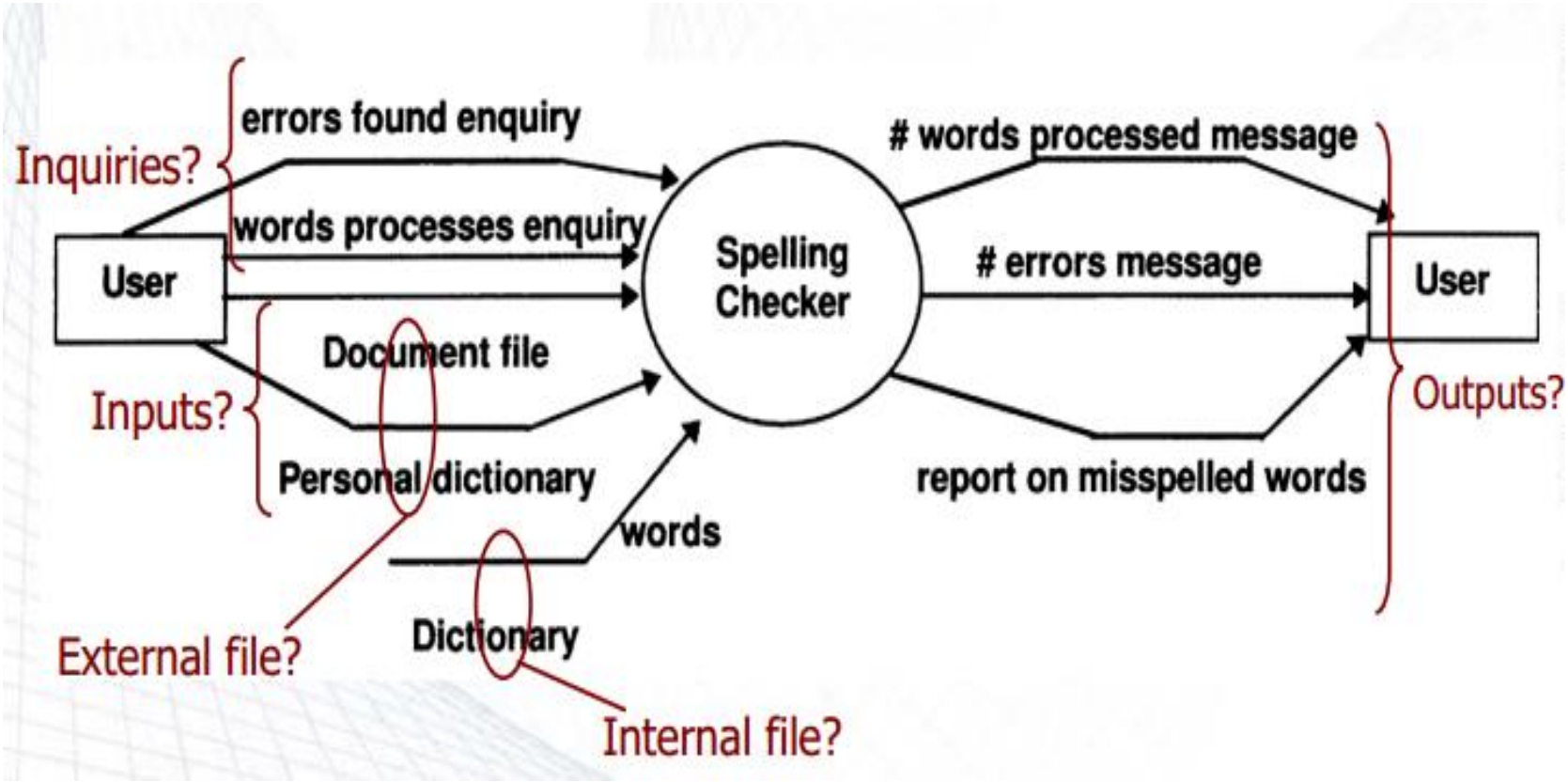
1. Checks all words in a document by comparing them to a list of words in the internal dictionary and an optional user-defined dictionary
2. After processing the document sends a report on all misspelled words to standard output
3. On request from user shows number of words processed on standard output
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

in the internal

Last req. cannot be shown

FP Example /1 c

Spell Checker: Convert spec to a diagram



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

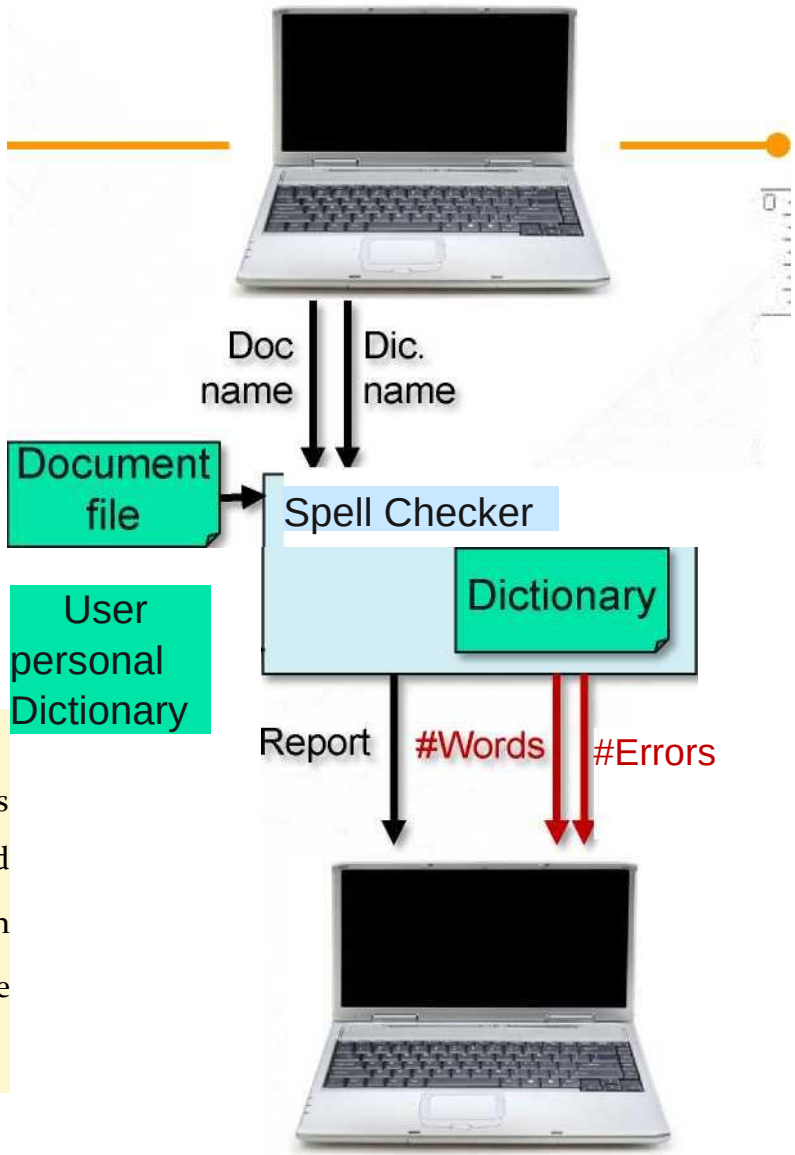
2 After processing the document sends a report on all misspelled words to standard output

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



FP Example /1e

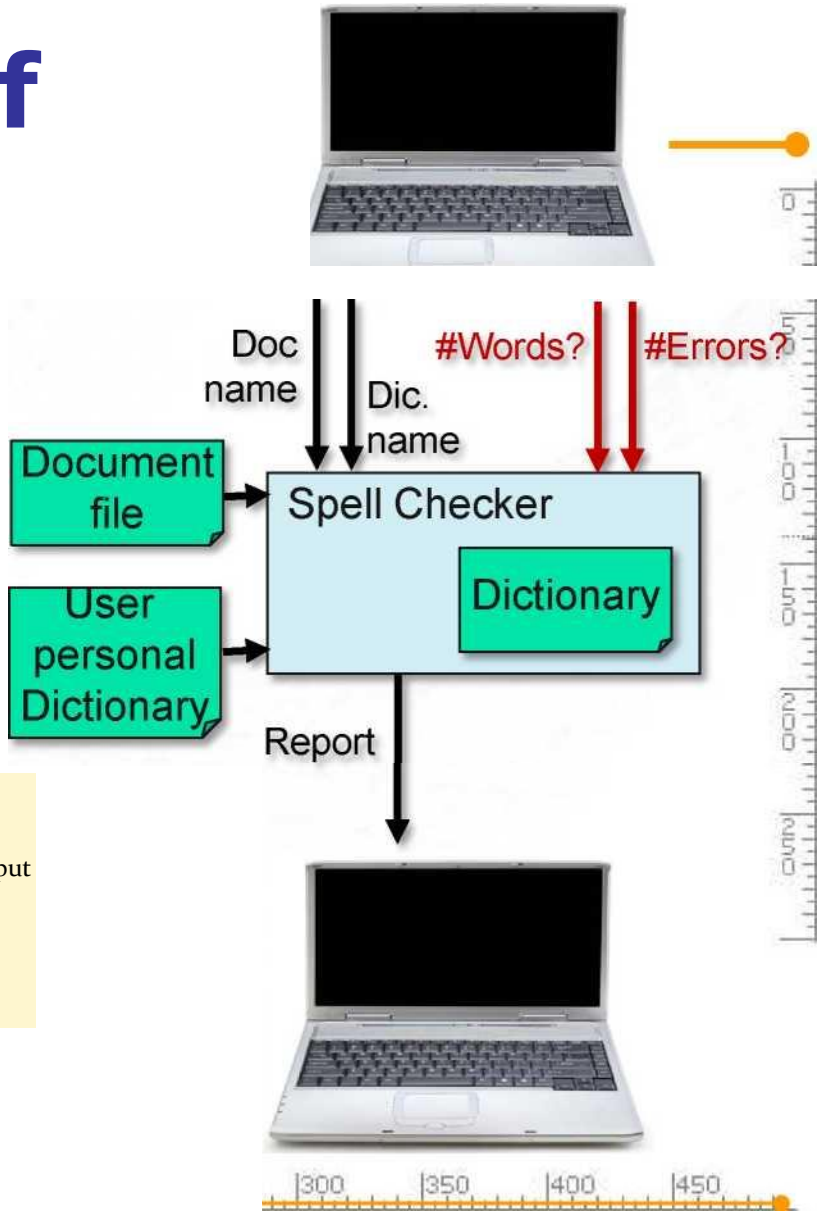
Element	Weighting Factor		
	Simple	Average	Complex
External inputs (**EI**)	3	4	6
External outputs *(EO)*	4	5	7
External inquiries (**EQ**)	3	4	6
External interface files *(EIF)*	5	7	10
Internal logical files *(ILF)*	7	10	15

- $N_{ei}=2$ (number of EI): document name, user-defined dictionary name
- $N_{eo}=3$ (number of EO): misspelled word report, number-of-words-processed message, number-of-errors-so-far message
- $N_{eq}=0$ (number of EQ): --
- $N_{eif}=2$ (number of EIF): document file, personal dictionary
- $N_{ilf}=1$ (number of ILF): dictionary

FP Example /1f

Solution B:

- EI: Doc. name + Dic. name -> 2
- EO: Report -> 1
- EQ: #Words? + #Errors? 2
- 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

FP Example /1g

Element	Weighting Factor		
	Simple	Average	Complex
External inputs (*EI*)	3	4	6
External outputs (*EO*)	4	5	7
External inquiries (*EQ*)	3	4	6
External interface files (*EIF*)	5	7	10
Internal logical files (*ILF*)	7	10	15

- $N_{EI}=2$ (number of EI): document name, user-defined dictionary name
- $N_{EO}=1$ (number of EO): misspelled word report
- $N_{EQ}=2$ (number of EQ): number-of-words-processed message, number-of-errors-so-far message
- $N_{EIF}=2$ (number of EIF): document file, personal dictionary
- $N_{ILF}=1$ (number of ILF): dictionary

FP Example /1 h

- Suppose that

  $F_1=F_2=F_6=F_7=F_8=F_{14}= 3;$

  $F_4=F_{10}=5$

  and the rest are zero.

$$VAF = 0.65 + 0.01\sum_{j=1}^{14} F_j$$

Solution A:

Solution B:

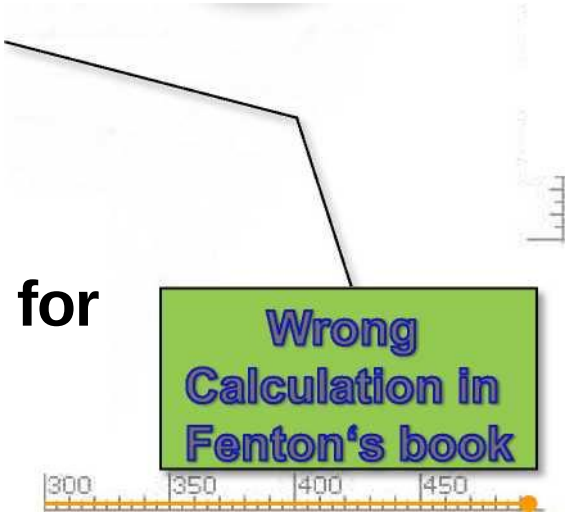
FP Example /1I - Discussion

Which solution is correct?

- Solution A?
- Solution B?
- None of them? (cf. Textbook)

Answer: Solution A is correct!

Why? The difference is because of correct interpretation of EO and EQ for the requirement #5



Function Points (IFPUG)

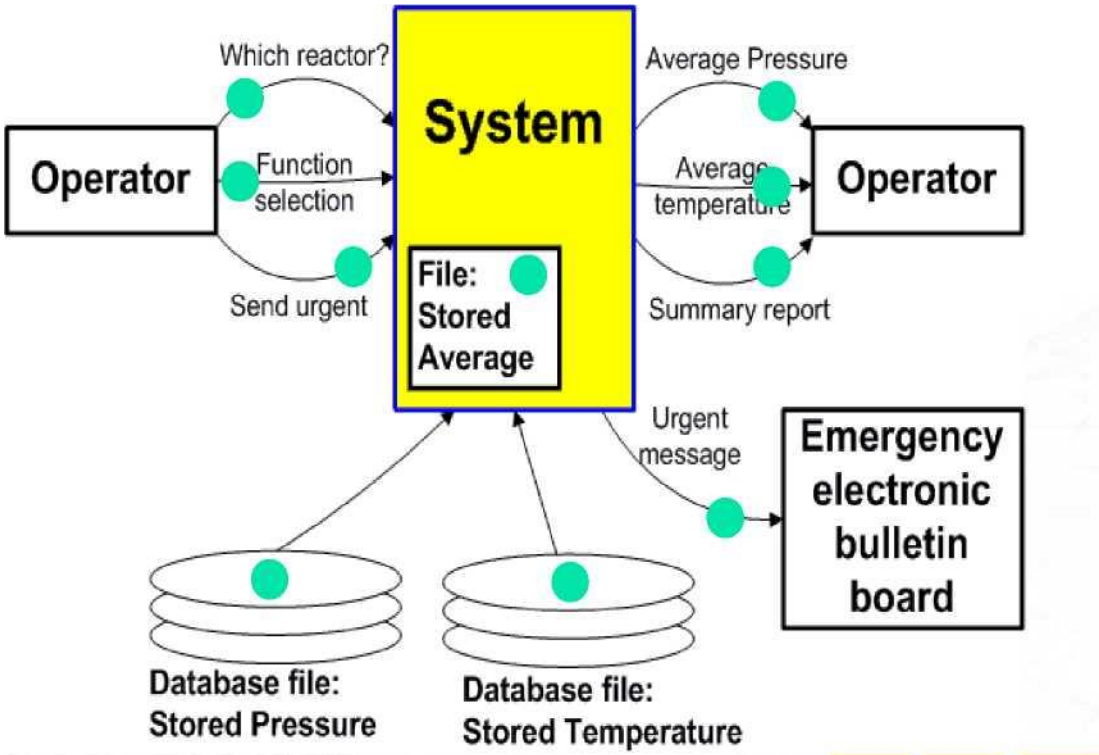
Example 2

FP: Example /2a

- 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.

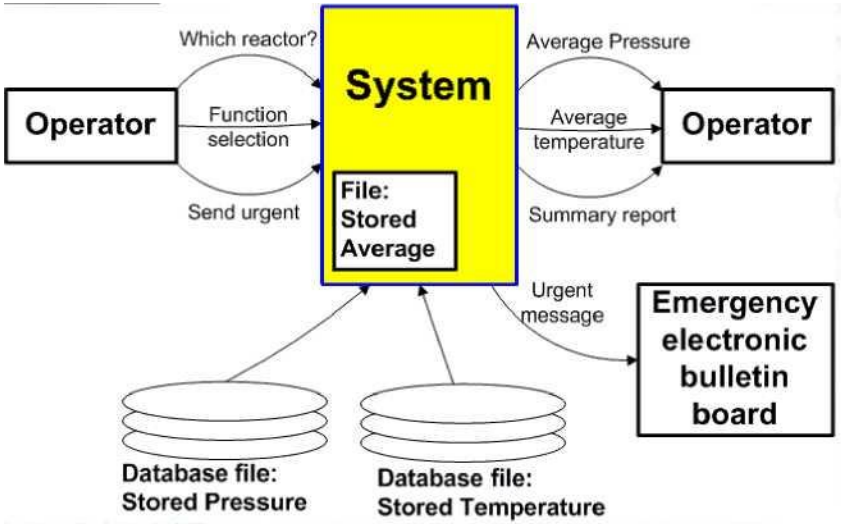
FP: Example /2b

EI EO EQ EIF ILF



EI EO EQ EIF ILF

FP: Example /2c



- N_{E1} 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_{1LF} number of internal logical files **1**

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

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

Review: Software Size

- Size measurement must reflect ***effort***, ***cost*** and ***productivity***.
- Defining software size in terms of length, functionality, and complexity, each capturing a key aspect of software size.
- Basic measure for length is LOC.
- Basic measure for functionality is Function Point (FP).
- FP is computed in two steps:
  1) **Calculating an** *Unadjusted Function point Count (UFC).*
  2) **Multiplying the UFC by a** *Value Adjustment Factor (VAF).*

  The final (adjusted) Function Point is:

  $$FP = UFC \times VAF$$
- FP variations: feature point, object point, use-case point.

FP vs. LOC /1

- A number of studies have attempted to relate LOC and FP metrics (Jones, 1996).
- The average number of source code statements per function point has been derived from case studies for numerous programming languages.
- Languages have been classified into different levels according to the relationship between LOC and FP.

FP vs. LOC *12*

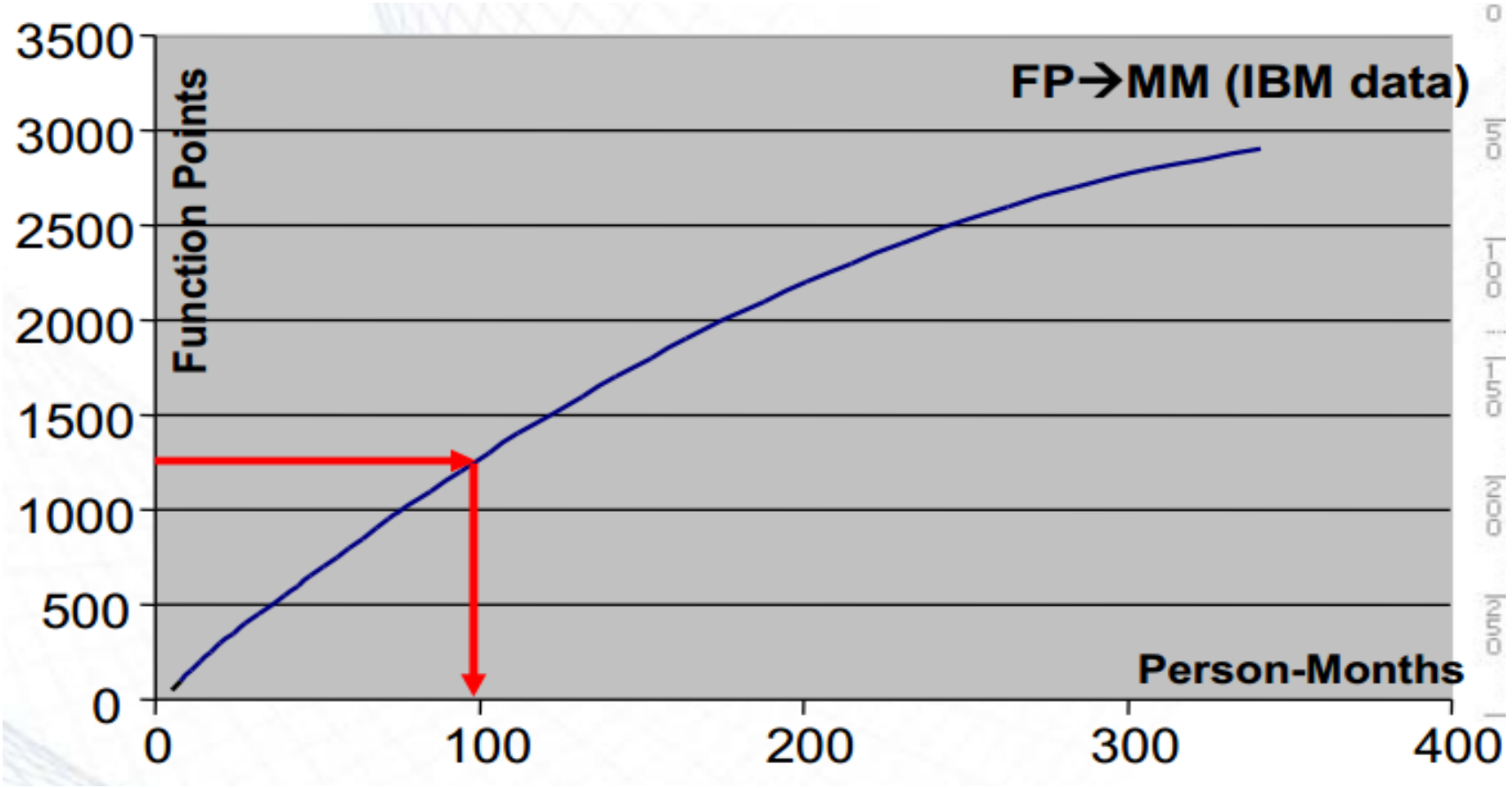
Language	Typical SLOC per FP
1GL Default Language	320
2GL Default Language	107
3GL Default Language	80
4GL Default Language	20
Code generators	15
Assembler	320
C	148
Basic	107
Pascal	90

FP vs. LOC / 3

Language	Typical SLOC per FP
C#	59
C++	60
PL/SQL	46
Java 2	60
Visual Basic	50
Delphi	18
HTML 4	14
SQL	13
Excel	47

Effort Estimation with FP

Effort estimation based on organization- specific data from past projects.



FP: Advantages - Summary

- Can be counted before design or code documents exist
- Can be used for estimating project cost, effort, schedule early in the project life-cycle
- Helps with contract negotiations
- Is standardized (though several competing standards exist)

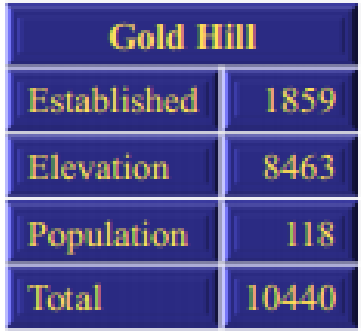
FP: Critics

"Guesstimate" Instead of Estimate!

- FP is a subjective measure: affected by the selection of weights by external users.
- Function point calculation requires a full software system specification. It is therefore difficult to use function points very early in the software development lifecycle.
- Physical meaning of the basic unit of FP is unclear.
- Unable to account for new versions of I/O, such as data streams, intelligent message passing, etc.
- Not suitable for “complex” software, e.g., real-time and embedded applications.

A Quote from John Munson

- "The most important thing we must know about derived metrics is that we cannot add apples and oranges. A new metric simply cannot be derived from two or more simple metrics. We recognize this fact in several aspects of our daily life. We do not recognize this fact in our analysis of software engineering problems. Consider the case of the city of Gold Hill, Colorado. Outside the town there is a sign that reads:"
- This is really a funny sign. But it is exactly how we define FP!!



Software Engineering Measurement (by John C. Munson)

Feature Point

Feature Point /1

- 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.
- Feature points accommodate applications in which the "algorithmic complexity" is high such as real-time, process control, and embedded software applications.
- For conventional software and information systems functions and feature points produce similar results. For complex systems, feature points often produce counts about %20~%35 higher than function points.

Feature Point *12*

- Feature points are calculated the same way as FPs with the additions of *algorithms* as an additional software characteristic.
- Counts are made for the five FP categories, i.e., number of external inputs, external outputs, inquiries, internal files, external interfaces, plus:
  - **Algorithm (*Na*):** A bounded computational problem such as inverting a matrix, decoding a bit string, or handling an interrupt.

Feature Point /3

- Feature points are calculated using:

Number of external inputs	x4
Number of external outputs	x5
Number of external inquiries	x4
Number of external interface files	X7←
Number of internal files	x7
Algorithms	x3←

$$UFeC = 4N_{EI} + 5N_{EO} + 4N_{EQ} + 7N_{EIF} + 7N_{ILF} + 3N_{A}$$

- The *UFeC* used in the function point calculation to calculate the feature points.

Object Point

Object Point /1

- Object points are used as an initial measure for size way early in the development cycle, during feasibility studies.
- An initial size measure is determined by counting the number of *screens*, *reports*, and third-generation *components* that will be used in the application.
- Each object is classified as *simple*, *medium*, or *difficult*.

Object Point /2

- Object point complexity levels for screens.

Number of views contained	Number and source of data tables		
	Total <4 <2 servers <2 clients	Total <8 2-3 servers 3-5 clients	Total 8+ >3 servers >5 clients
< 3	Simple	Simple	Medium
3 - 7	Simple	Medium	Difficult
8+	Medium	Difficult	Difficult

Object Point / 3

- Object point complexity levels for reports.

Number of views contained	Number and source of data tables		
	Total <4 <2 servers <2 clients	Total <8 2-3 servers 3-5 clients	Total 8+ >3 servers >5 clients
0 - 1	Simple	Simple	Medium
2 - 3	Simple	Medium	Difficult
4+	Medium	Difficult	Difficult

Object Point / 3

- The number in each cell is then weighted and summed to get the object point.
- The weights represent the relative effort required to implement an instance of that complexity level.
  - **Complexity level for object point:**

Object Type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component	-	-	10

New object points = (object points) x ***(100 -r) /100***

Assuming that ***% r*** of the objects will be reused from previous projects.

How to Calculate OP?

1. Estimate the number of “screens” of the system and the approximate number of data entries for each screen (called view).
2. Estimate the number of “reports” that will be generated by the system (including any output file, writing to database, etc.) and the approximate number of data entries for each report (called view).
3. For each “screen” and “report” and their corresponding views, use Tables (previous pages) to determine whether that screen or report is “simple”, “medium” or “difficult”.
4. Weight the screens and reports using the weight table and sum them up to get the OP number.
5. If you have any acquired component, give it the weight of 10 and add it to the OP.
6. If you have any part of your system reused from a previous generation define a percentage of reuse and then adjust the value of OP accordingly

Effort Estimation with OP

- Formula:

  Effort [Person-Month] = OP / PROD

- How to measure PROD?

Developer Experience/Skills	Very Low	Low	Average (Nominal)	High	Very High
PROD	4	7	13	25	50

- Example:

  Effort [Person-Month] = 13 / 13 = 1 Assuming average experience and 0% reuse.

Example: OP

- Suppose that you have
  - 4 screens, 2 of them simple (weight 1) and 2 of them medium (weight 2)
  - 3 reports, 2 of them simple (weight 2) and 1 medium (weight 5),

  then the total number of OP is:

  $$OP = 2x1 + 1x2 + 2x2 + 1x5 = 13$$

- If you have any acquired component, give it the weight 10 and add it to the OP.
- If you have any part of your system reused from a previous generation, define a percentage of reuse (say 10%) and then adjust the value of OP accordingly.

$$OP_{new} = 13 \text{ x } (100\text{-}10)/100 = 11.7$$

Example 2

- Determine object points for a smart vending machine assuming that the system is average size (total number of clients-servers is less than 8) using the following data:

Screens	*Views (Data Items)*
User Screens	
S1. Buying Screen	Amount of coins to insert, selection
Administrator Screens	
S2. Inventory Update	Inventory input update (20 data items)
S3. Change Update	Change infrared input update (4 data items)

Example 2 (cont'd)

Reports	***Views (Data Items)***
User Reports	
R1. Display money entered	Amount of money entered
R2. Not enough money entered	Insufficient funds
R3. Returning change	Amount of change
R4. Out of stack message	Out of stock
R5. No change message	No change
R6. Selection Approved	Release selection signal
Administrator Reports	
R7. Machine Report	Inventory (20 data items), change information (4 data items)

Example 2 (cont'd)

Simple	*Medium*	*Difficult*
S1	*S3*	*S2*
R1 R2 R3 R4 R5 R6	*None*	*R7*

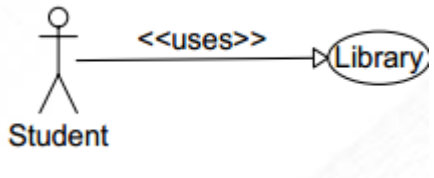
OP = (1X1) + (1X2) + (1X3) + (2X6) + (5X0) + (8X1) OP = 26

Effort = 26/13 = 2 person month

Use-Case Point

Use-Case Point /1

- *Function Point* is a method to measure software size from a *requirements* perspective.
- *Use-Case* is a method to develop *requirements*. Use Cases are used to validate a proposed design and to ensure it meets all requirements.



Question: How to use Use-Cases to measure function point and vice-versa?

Use-Case Point / 2

Question: How to use Use-Cases to measure function point and vice-versa?

Two methods:

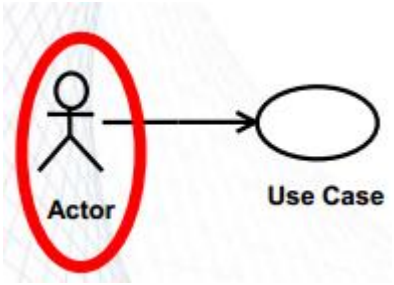
1. Identify and weight *actors* and *use-cases*
2. Count the inputs, outputs, files and data inquiries from use-cases (using the *use-case definition* and *activity diagram*).

Use Case Point /3

- Use Case Point Estimation has its origin in the work by Gustav Karner (1993)
- Process:

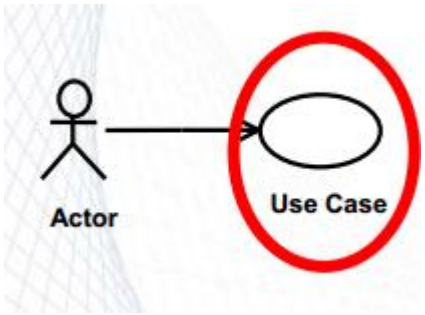
1. Identify and weight Actors (→UAW)
2. Identify and weight Use Cases (→UUCW)
3. Calculate Unadjusted Use Case Points (→UUCP = UAW + UUCW)
4. Calculate Value Adjustment Factor (→VAF)
5. Calculate (Adjusted) Use Case Points →UCP = UUCP * VAF

Use Case Point: Actors



- For each Actor, determine whether the actor is simple, average or complex.
  - **A *simple actor* represents another system with a defined Application Programming Interface (API). [weight: 1]**
  - **An *average actor* is either another system that interacts through a protocol (HTTP, FTP, user-defined, etc.), a data store (files, DBs, etc.) or a person interacting through a text-based interface. [weight: 2]**
  - **A *complex actor* is a person interacting through a graphical user interface (GUI). [weight: 3]**

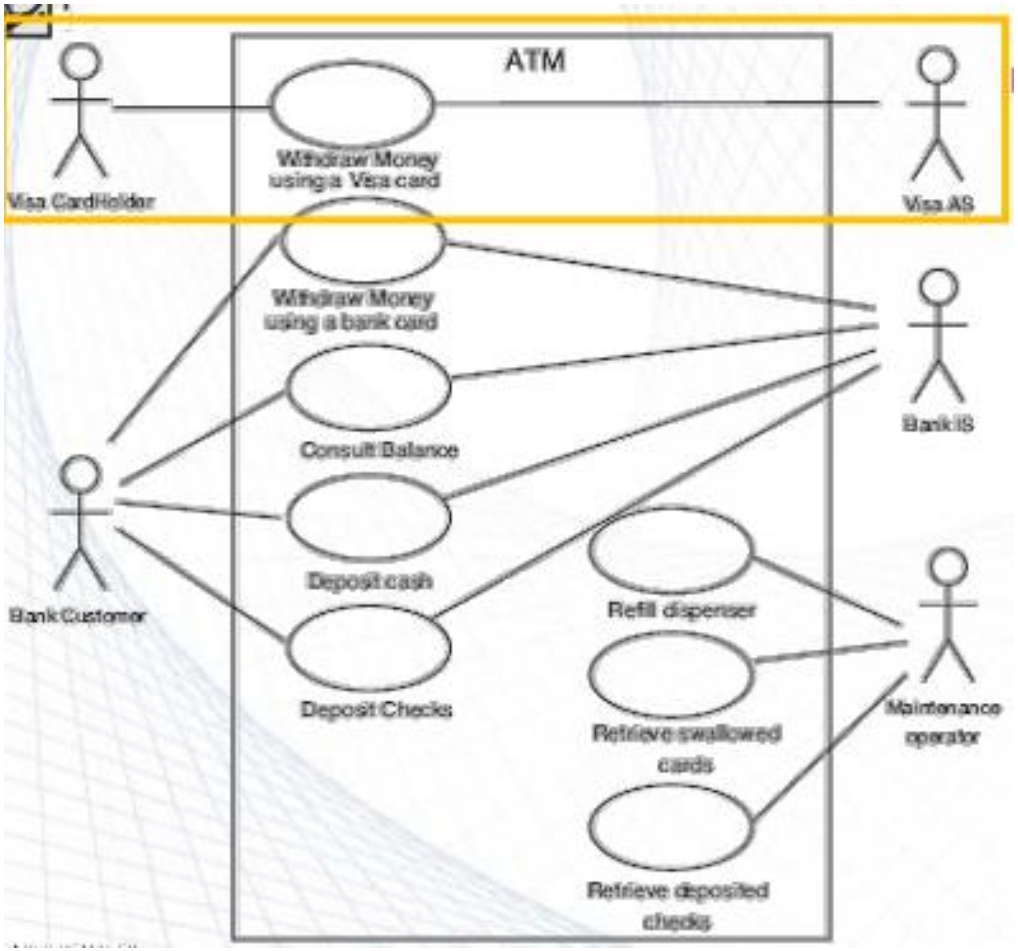
Use Case Point: Use Cases



- For each Use Case, determine whether it is simple, average or complex.
  - A **simple use case** contains up to 3 transactions. [weight: 5]
  - An **average use case** contains 4-7 transactions. [weight: 10]
  - A **complex use case** contains more than 7 transactions. [weight: 15]

- ***Note:*** The original Use Case Point method suggested to count only transactions of the main (success) scenarios. Some researchers propose to include in the counting also alternate (extensions) and exception (error) transaction sequences.

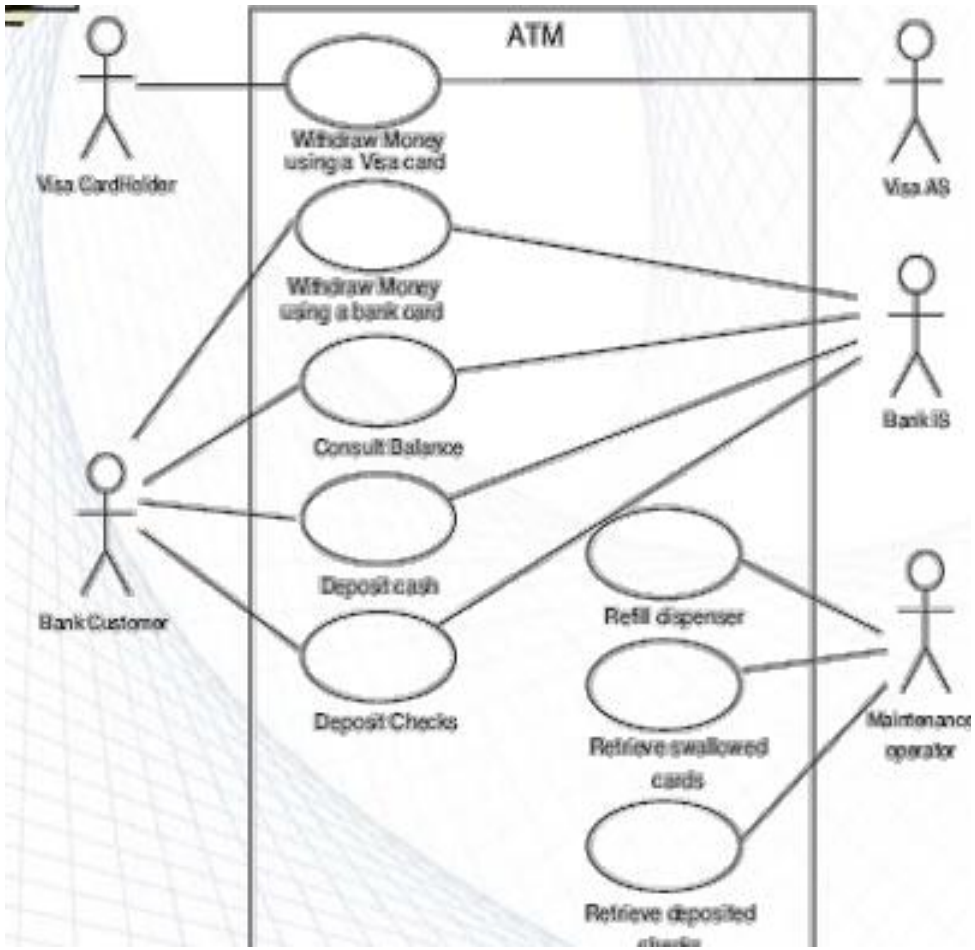
Example: ATM



- Actors:
  - Visa Cardholder: complex (interacts via **GUI)**
  - Visa AS:

    average (interacts via communication protocol)

Example: ATM

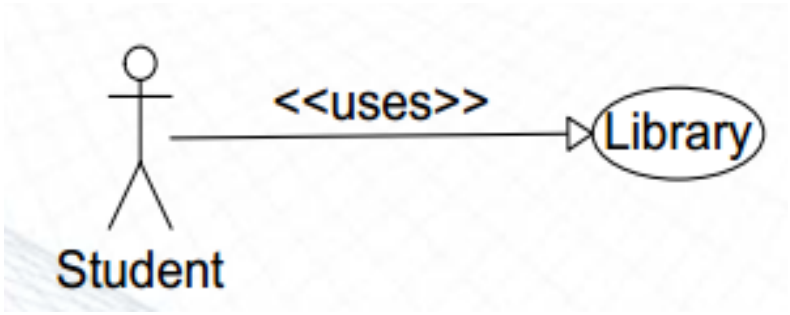


- Use Case:
  - "Withdraw money using a Visa card"

    complex (more than 7 transactions, even without counting extensions or exceptions)
- UAW = 3 + 2 = 5
- UUCW= 15

Use-Case Point / 2

- We must count the inputs, outputs, files and data inquiries from use-cases.
- Function points become evident using the *use-case definition* and *activity diagram* for the use-case. Each step within the activity diagram can be a transaction (inputs, outputs, files and data inquiries).



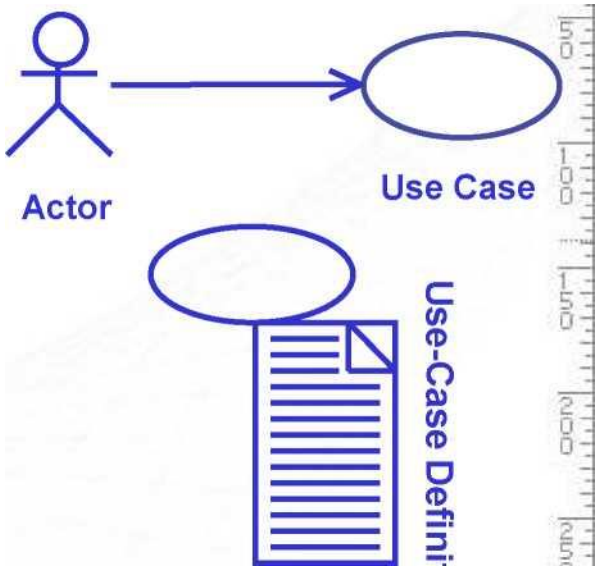
Examining Use Case Scenario

- Use Case might be further refined via Activity and Interaction Diagrams
- Examining Use Case definition and Activity and/or Interaction diagrams can also lead to FP count.
- How?

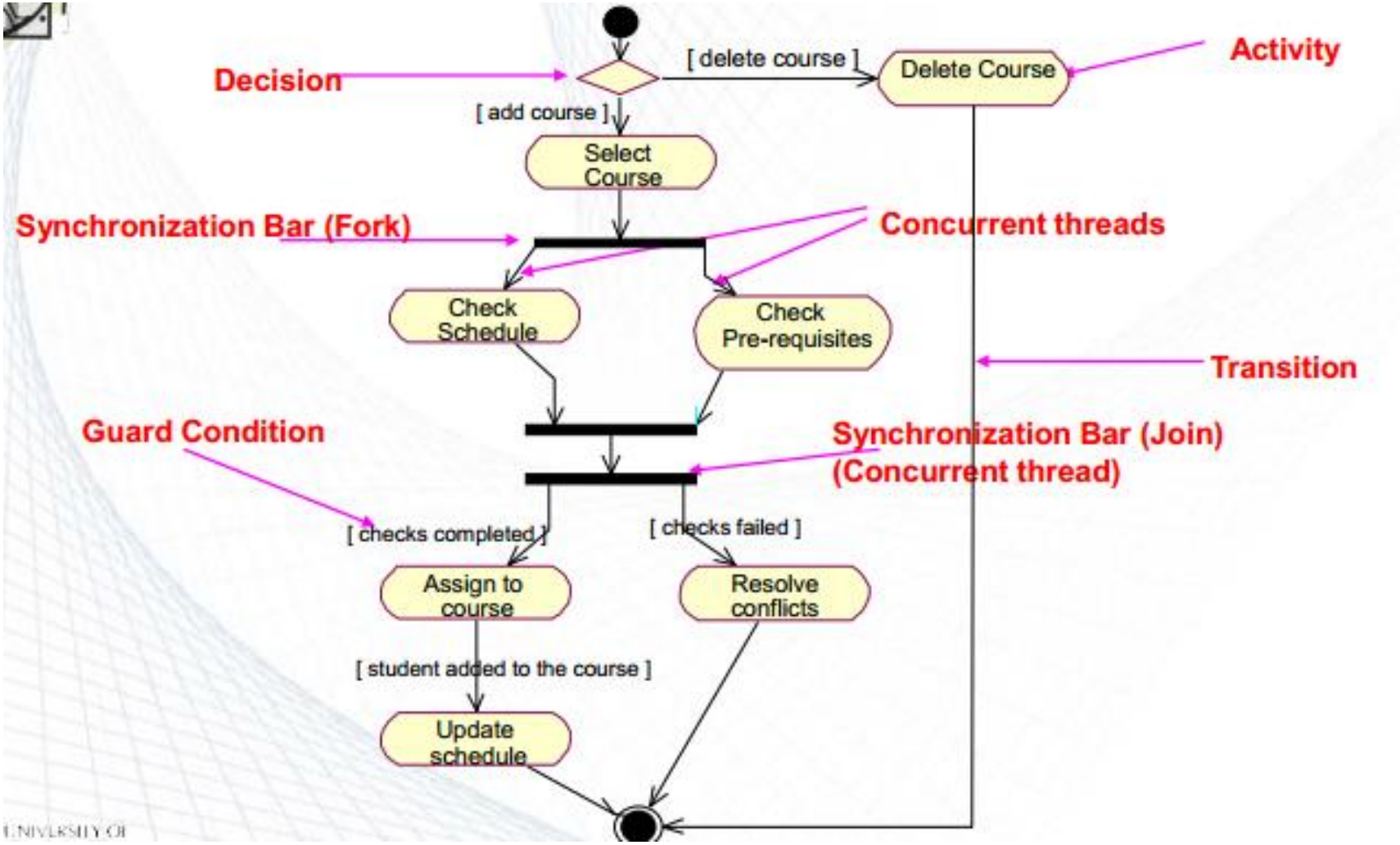
Use-Case Definition

- A typical *use-case definition* document consists of:

- Use Case name
- Actor name
- Objective
- Preconditions
- Results (Post-conditions)
- Detailed description (actions & responses)
- Exceptions and alternative courses



Example: Activity Diagram



Example 1: Use-Case Scenario

But these are not independent from the "Find" input, so they are actually data attributes for "Find"

Use-Case: Library Scenario:

1. The user may enter a book's ISBN number, student ID, or student name ***(input × 3)***
2. The user will press "Find" ***(input × 1)***
3. If the user enters a book ISBN number ***(input × 1)***

- The system will display information related to that book and write the results to a file ***(output × 1) (internal file × 1)***

4. If the user entered a student name or student ID ***(input × 2)***

- The system will return a list of all books on the waiting list for that student and write the results to a file ***(output × 1) (internal file × 1)***
- The user can select one book from the list ***(external inquiry × 1)***
- The system will search the database by ISBN number ***(external file × 1)***
- The system will display information related to that book and available date and will write the results to a file ***(output × 1) (internal file × 1)*** But these are not independent from the "Find" input, so they are actually data attributes for "Find"

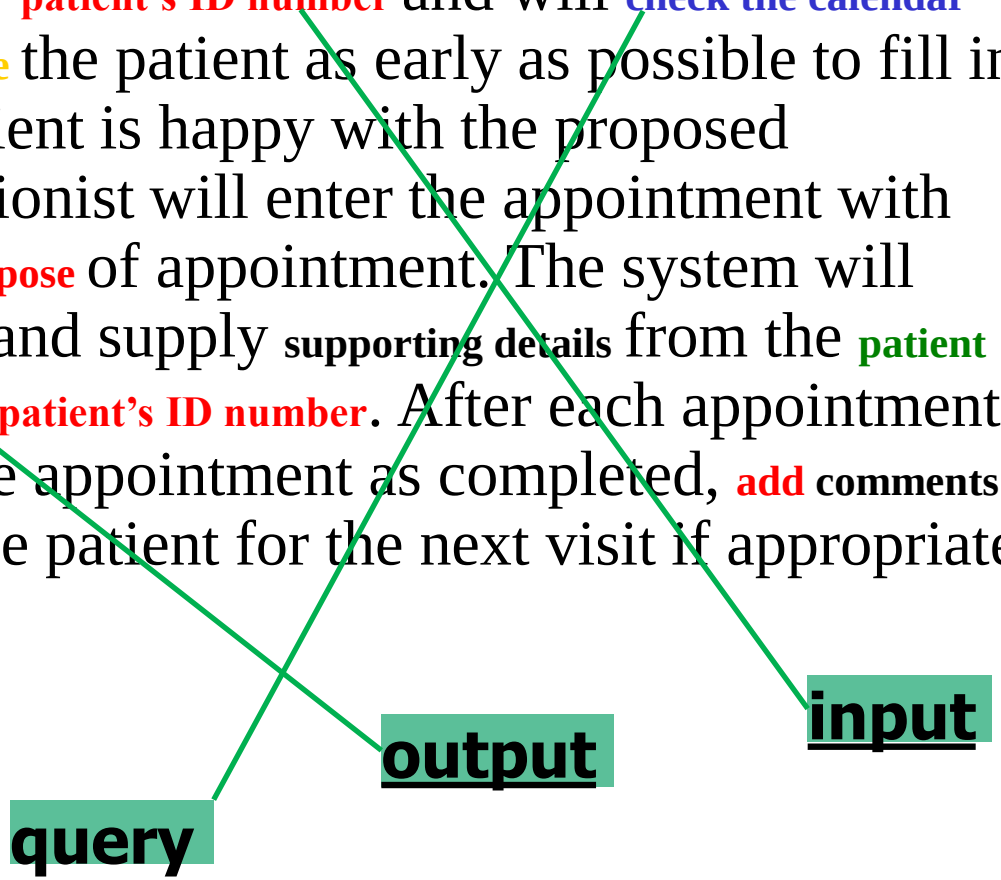
Together they will qualify as 1 output

Example 2

- Dr. X, a recent graduate from a medical university, is starting her medical practice in a small town. She is planning to hire a receptionist. She approaches software company Y to build a software system to manage the patients' appointments. The following is her problem description (next two slides).
- Suppose that you are the analyst in charge of estimating the cost of this system and you base your estimation of the function point (FP) count for this system.
- Calculate the unadjusted function point count (UFC) for the system assuming the average weight for all entries.

Example 2 (cont'd)

- When a patient calls for an appointment, the receptionist will ask the **patient's name** or **patient's ID number** and will **check the calendar** and will try to **schedule** the patient as early as possible to fill in vacancies. If the patient is happy with the proposed **appointment**, the receptionist will enter the appointment with the **patient name** and **purpose** of appointment. The system will verify the **patient name** and supply **supporting details** from the **patient records**, including the **patient's ID number**. After each appointment the Dr.X will **mark** the appointment as completed, **add comments**, and then schedule the patient for the next visit if appropriate



Example 2 (cont'd)

- The system will answer queries by **patient name**, by **patient ID** and by **date**. Supporting details from the **patient's records** are **displayed** along with the **appointment information**. The receptionist can **cancel** appointments. The receptionist can **print** out a **notification list** for making **reminder calls** 2 days before appointments. The system includes the patient's phone numbers from the **patient records**. The receptionist can also print out **daily** and **weekly work schedules** with all the patients.

Note that we have to again ask ourselves whether these qualify for EI, EO, EQ, EIF and ILF.

Note that we have to again ask ourselves whether these qualify for EI, EO, EQ, EIF and ILF.

Example 2 (cont'd)

Type	Description	No.	Weight	Weighted value
Inputs (EI)	Patient name, Patient ID number, Appointment completed, Appointment purpose, Cancel appointment	5	4	20
Outputs (EO)	Comments, Calendar, Supporting details, Appointment Information, Notification List, Daily schedule, Weekly schedule,	7	5	35
Queries (EQ)	Check calendar, Query by name, Query by ID, Query by date, Verify patient, Available Appointment	6	4	24
Internal files (ILF)	Patients' data record	1	10	10
External files (EIF)	--	0	7	0
		Total UFC		89

Recent Approaches

- Recent approaches to size measurement and estimation include:
  - Decomposition
  - Expert opinion
  - Analogy
  - Class-code expansion

1. Decomposition

- In many cases, the measurement or estimation problem is too complex to be considered in one piece.
- As a result, the measurement or estimation problem is decomposed or partitioned into smaller, more manageable problems.
- By decomposing the problem, measurement or estimation can by performed in a stepwise fashion.

(Remember the GQM technique)

2. Expert Opinion /1

- Expert opinion refers to predictions made by experts based on past experience.
- The technique is based on combination of individual predictions.
- The result is a group estimate arrived at by consensus. The group estimate is typically a better overall estimate than any individual prediction.
- In general, the expert opinion approach to estimation can result in very accurate estimates, however it is entirely dependent on the experience of the expert.

2. Expert Opinion *12*

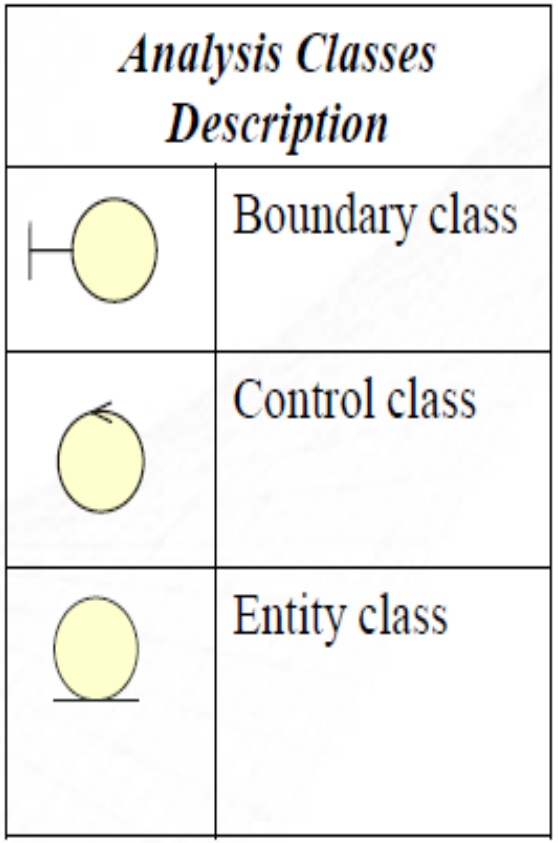
- Example: Wideband Delphi Technique
  - A group of experts is each given the program's specifications and an estimation form.
  - They meet to discuss the product and any estimation issues.
  - They then each anonymously complete the estimation forms
  - The estimates are given to the estimate coordinator, who tabulates the results and returns them to the experts.
  - Only each expert's personal estimate is identified; all others are anonymous.
  - The experts meet to discuss the results, revising their estimates as appropriate.
  - The cycle continues until the estimates converge to an acceptable range.

3. Analogy

- Analogy is a more formal approach to expert opinion also referred to as the Fuzzy-Logic Method or Case-based Method.
- Estimators compare the proposed project with one or more past project cases. Differences and similarities are identified and used to adjust the estimate.
- The estimator will typically identify the type of application, establish an initial prediction, and then refine the prediction within the original range.
- The accuracy of the analogy approach is dependent on the availability of a case-base of project information.

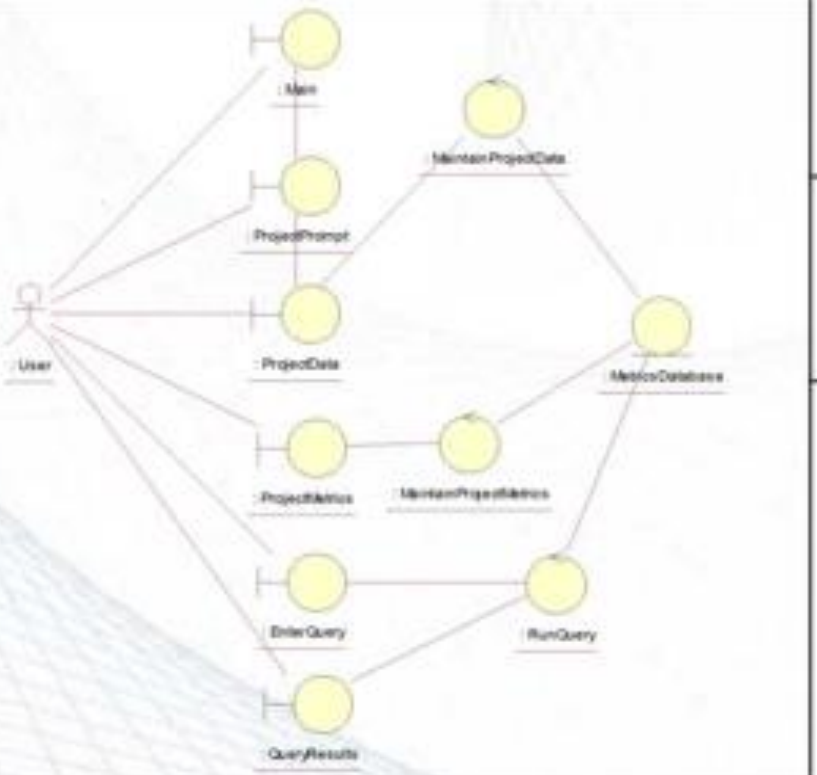
4. Class-Code Expansion

- In the design phase, boundary classes are usually expanded by the ratio of 1 to 2 and control classes are expanded by the ratio of 2 to 3, respectively. The entity classes are usually expanded to a subsystem having 4 to 8 design classes.
- An average design class in Java has 20 methods, each method having 10 to 20 lines of code.



Example

- Using class-code expansion

Model	*Description*	
		Boundary class
		Control class
		Entity class

Example (cont'd)

- Calculate total software size in LOC, assuming total size is 1.5 x methods LOC
- Calculate total development cost.

Analysis classes	*No.*
Boundary class	6
min (X 1)	6
max (X 2)	12
Control class	3
min (X 2)	6
max (X 3)	9
Entity class	1
min (X 4)	4
max (X 8)	8

Total classes	*Total Operations*	*Total LOC*
Min 16	Min 320	1.5 X 320 X 10 = 4,800 (minimum)
Max 29	Max 580	1.5 X 320 X 20 = 9,600
		1.5 X 580 X 10 = 8,700
		1.5 X 580 X 20 = 17,400 (maximum)

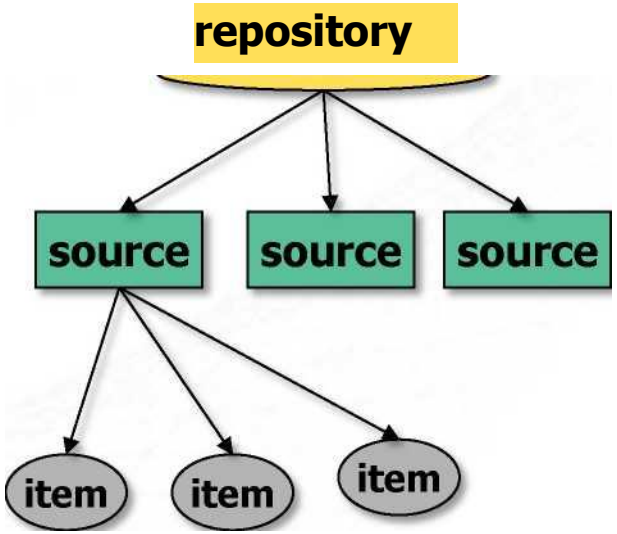
Measuring Software Reuse

Software Size: Reuse /1

- Reuse measures how much of a product was copied or modified.
- Measurement of size must also include some method of counting reused products.
- It is difficult to define formally what is meant by reused code. Whole programs can be reused without modification, but it is more common to reuse some part of code (a module, function, or procedure).
- The ***extent of reuse*** is defined as follows (Software Productivity Consortium, 1995):
  - **Reused verbatim:** code in the unit was used without any changes
  - **Slightly modified:** fewer than 25% of lines of code in the unit were modified
  - **Extensively modified:** 25% or more of lines of code were modified
  - **New:** none of the code comes from a previously constructed unit

Software Size: Reuse /2

- Reuse needs a *repository of reusable components*
- Reuse metrics
  - ***Reuse Level:*** percentage of different items coming from a given source in a repository.
  - ***Reuse Frequency:*** percentage of references to items from a given source in a repository.
  - ***Reuse Density:*** normalized number of items from a given source in a repository.



Software Size: Comparison

Criteria	*LOC*	*Halstead's*	*FP*	*OP*
Applicable to early stages (requirement phase) of software development?				
Subjective measure?				
Derivable from system requirements?				
Language dependable?				
Has clear and understandable physical meaning?				