CSE 4621 / SWE 5620
Software Metrics

Measuring Internal Product Attributes: Size

Chapter -8-

Khaled Slhoub, PhD

Software Size

- Size is a fundamental software characteristic of interest.
- We will look at different ways to measure the size of software development
- We will examine:
  - Length
  - Functionality
  - Complexity

Properties Of Software Size

- Like physical entities, software products can be described in terms of their size
- Size alone cannot directly indicate external attributes such as effort, productivity, and cost.
- Size measures are very useful:
  - Test effort
  - Maintenance effort
  - Development time
  - Development resources
  - Productivity (Size/Effort)
  - Defect density (Defect count/Size)
  - Cost estimation

Code includes source code, intermediate code, byte code, and even executable code.

Properties Of Software Size

Florida Tech

… there is a major problem with the lines-of-code measure: it is not consistent because some lines are more difficult to code than others …. One solution to the problem is to give more weight to lines that have more 'stuff' in them.

CONTE ET AL. 1986

Those who reject a measure because it does not provide enough information may be expecting too much of a single measure.

Software Length

Florida Tech

- The most common view of program length is as a count of the number of lines of source code (LOCs). In large source code, KLOC is used too.
- Some LOCs are different from others
- What to count is a major issue:
  - blank lines
  - comment lines
  - data declarations
  - lines that contain several separate instructions
- We must explain how each of the above is handled
- Language differences is another major issue

Software Length

- Consideration must be given to the intended use of the metric:
  - effort and schedule estimation
  - defect density estimation (defects/KLOC)
  - disk space estimation
  - code complexity
  - pages of formatted source listings

Jones reports that one count can be as much as five times larger than another, simply because of the difference in counting technique (Jones 2008)

Software Length

There is some general consensus that blank lines and comments should not be counted. Conte et al. define an LOC as any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This definition specifically includes all lines containing program headers, declarations, and executable and nonexecutable statements (Conte et al. 1986). Grady and Caswell report that Hewlett-Packard defines an LOC as a non-commented source statement: any statement in the program except for comments and blank lines

Grady and Caswell 1987

Software Length

- To stress the fact that an LOC according to this definition is actually a non-commented line (NCLOC)
- Also called *effective lines of code*
- **Useful** when comparing actual components code and evaluating the growth of systems over time
- **Problem** is valuable size information is lost when comment lines are not counted
  - program size is important in deciding how much computer storage is required
  - how many pages are required for a print-out

The **number of comment lines of program text** (CLOC) be measured and recorded separately. Total size (LOC) = NCLOC + CLOC. or CLOC/LOC

Software Length

Florida Tech

- Other code size measures try to take into account the way code is developed and run
- Sometimes programs are full of data declarations and header statements, and there is very little code that actually executes
  - issue with testing
- Measure the number of executable statements (ES)
  - counts separate statements on the same physical line as distinct
  - ignores comment lines, data declarations, and headings
- **Problem** is the amount of code can be significantly different from the amount of code actually written

Software Length

- programming team may write drivers, stubs, prototypes, and “scaffolding” for development and testing, but these programs are discarded when the final version is tested and turned over to the customer.
- Here, the developers want to distinguish the amount of delivered code from the amount of developed code.
- The number of delivered source instructions (DSI)
- Unlike ES, DSI includes data declarations and headings as source instructions

Software Length

- Refer to the simple program in Figure 8.1 (next slide) of our text:
  - 30 count of the number of physical lines (including blank lines)
  - 24 count of all lines except blank lines and comments
  - 20 count of all statements except comments
  - 17 count of all lines except blank lines, comments, declarations and headings
  - 13 count of all statements except blank lines, comments, declarations and headings
  - 6 count of only executable statements, not including exception conditions

Even with a small program, the difference between least and most compact counting methods can be a factor of 5:1

Software Length - Example

In order for LOC count to be a useful metric within an organization, the organization must agree on

- what to count
- how to count and
- use a coding standard consistently

```
with TEXT_IO; use TEXT_IO;
procedure Main is

        --This program copies characters from an input
        --file to an output file. Termination occurs
        --either when all characters are copied or
        --when a NULL character is input

        NullchAr, Eof: exception;
        Char: CHARACTER;
        Input_file, Ouptu_file, Console: FILE_TYPE;
Begin
        loop
                Open (FILE => Input_file, MODE => IN_FILE,
                                NAME => "CharsIn");
                Open (FILE => Output_file, MODE = > OUT_FILE,
                                NAME => "CharOut");
                Ge (Input_file, Char);
                if END_OF_FILE (Input_file) then
                        raiseEof;
                elseif Char = ASCII.NUL then
                        raiseNullchar;
                else
                        Put(Output_file, Char);
                endif;
        end loop;
exception
        whenEof => Put (Console, "no null characters");
        whenNullchar => Put (Console, "null terminator");
end Main
```

FIGURE 8.1 A simple program.

Software Length

For a very full discussion of Line of Code counts, see:

Park, Robert E., *Software Size Measurement: A Framework for Counting Source Statements*, CMU/SEI-92-TR-20, 1992. Available for download from

https://resources.sei.cmu.edu/asset_files/TechnicalReport/1992_005_001_16082.pdf

Halstead's Approach

- In 1977, Maurice Halstead introduced a collection of software measures referred to as Software Science.
- To the best of our knowledge, the measures have not been validated, even by Halstead.
- Halstead viewed a program P as a collection of tokens: operations and operands
  - number of unique operators: μ_1
  - number of unique operands: μ_2
  - total occurrences of operators: N_1
  - total occurrences of operands: N_2

Halstead's Approach

- Halstead's derived measures:
  - Length of P: $N = N_1 + N_2$
  - Vocabulary of P: $\mu = \mu_1 + \mu_2$
  - Volume of P: $V = N \times \log_2 \mu$
  - Other measures attempt to estimate difficulty and effort to generate P. See text.

Halstead's Approach - Example

```
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;
}
```

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

Halstead's Approach

- Many problems with Halstead's measures have been identified:
  - No evidence of validation
  - No consensus on meaning of terms, e.g., volume, difficulty
  - Unclear relationship between real world and mathematical model

Alternative Code Size Measures

Florida Tech

- Two other measures of software length
  - The number of characters of storage required for the source code
  - The number of characters (CHAR) in the source code
  - If A is the average number of characters per line of code, then:

$$CHAR = A \times LOC$$

Dealing with Nontextual or External Code

- A complicating factor today is the automatic generation of code:
  - iconic programming
  - Visual Basic type languages
  - Power builder type development systems, i.e., use of a template to create database interfaces.
- Does it make sense to use LOC counts in determining the length of OO software?
  - What about using counts of object classes and methods instead?
  - We will discuss OO metrics in detail later in the course.

Design Size

- Counting design elements rather than LOCs
  - the abstractions used to express the design
  - design methodology
  - artifacts developed
  - the level of abstraction
- Procedural design
  - lowest level: count the number of procedures and functions, and count number of arguments
  - higher level: count the number of packages and subsystems in terms of the number functions and procedures
  -

Design Size

- Object-oriented designs
  - Packages:
    - Number of subpackages, number of classes, interfaces (Java), or abstract classes (C++)
  - Design patterns:
    - Number of different design patterns used in a design
    - Number of design pattern realizations for each pattern type
    - Number of classes, interfaces, or abstract classes that play roles in each pattern realization
  - Classes, interfaces, or abstract classes:
    - Number of public methods or operations, number of attributes
  - Methods or operations:
    - Number of parameters, number of over-loaded and over-ridden versions of a method or operation

Design Size

- Object-oriented designs
  - object-oriented design size measures is the number methods in a class (WMC)
  - the number of attributes in each methods
  - the number of design patterns

Requirements Size

- Requirements and specification documents generally combine text, graphs, and special mathematical diagrams and symbols
- A use case analysis may consist of a UML use case diagram along with a set of use case scenarios that may be expressed as either text or as UML activity diagrams
  - The nature of the presentation depends on the particular style, method, or notation used.
- Because a requirements analysis often consists of a mix of document types, **it is difficult to generate a single size measure**.

Requirements Size

- There are obvious atomic elements in a variety of requirements and specification that can be counted:
  - One simple size measure can be enforced is counting the number of pages (commonly used in industry)
  - Use case diagrams: Number of use cases, actors, and relationships of various types
  - Use case: Number of scenarios, size of scenarios in terms of steps, or activity diagram model elements
  - Domain model (expressed as a UML class diagram): Number of classes, abstract classes, interfaces, roles, operatoins, and attributes

Functional Size Measures

Florida Tech

- Many argue that size is misleading, and that the amount of functionality inherent in a product paints a better picture of product size
  - works well when estimation start early

- An intuitive notion of the amount of functionality contained in a delivered product or in a description of how the product is supposed to be
  - Function Points (FPs)
  - COCOMO II approach

Estimation before implementation

Function Points

Florida Tech

Motivation: a method for estimating which

- Relates closely to customer-desired functionality, expressed in business terms.
- Exists independent of the implementation language.
- Can be used early in the project (before a design is determined).

Function points (FPs) indicate the amount of functionality a system provides, as described by a specification, rather than the number of source statements the programmers write

Function points (FPs) measure the amount of functionality in a system based upon the *system specification*.

Function Points Basics

- Function Point (FP) is a weighted measure of software functionality.

- FP is computed in two steps:

  1) Calculating ***Unadjusted Function Point Count*** (***UFC***).

  2) Calculating ***Technical Complexity Factor (TCF)***

  Then, final Function Point is

$$\mathbf{FP = UFC \times TCF}$$

Function Points Basics (cont'd)

Florida Tech

First, we compute an **unadjusted function point count (UFC)** from the number of "items" of the following types:

- **External inputs** (e.g., transaction types), items that enter the boundary of the system that cause processing to take place.
- **External outputs** (e.g., types of reports), items that leave the system boundary after processing has occurred.
- **External inquiries** (e.g., types of interactive inquiries processable by the system), unique inquiries that require an immediate response.
- **External files** (e.g., interfaces to other systems, including files accessed by the system but not modified by it, basically external sources of information required for its processing).
- **Internal files**: Logical master files in the system (e.g. DB tables, internal files).

Function Points Basics (cont'd)

- The complexity of each of the four types of externals and the internal logical file(s) items assigned a subjective "complexity" rating on a three-point ordinal scale: *simple*, *average*, or *complex*. Their counts are weighted correspondingly and then summed to determine the **UFC**, "function point count"

		Function				
		Outputs	Inquiries	Inputs	Internal Files	External Files
Complexity	Simple	4	4	3	7	5
	Average	5	5	4	10	7
	Complex	7	6	6	15	10

multiplying the number of items in a variety by the weight of the variety and summing over all 15:

$$\text{UFC} = \sum_{i=1}^{15} (\text{Number of items of variety } i) \times (\text{weight}_i)$$

Function Points Basics (cont'd)

Florida Tech

Function Point Complexity Weights

Elements	Complexity Weighting Factor			
	Simple	Average	Complex	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 Point Count (UFC) =				

Function Points Basics (cont'd)

Components of the Technical Complexity Factor

Second, TCF, called also "value adjustment factor" (VAF), is determined by assessing the impact of 14 factors relating to the operation of the system such as performance, ease of use, data communication, etc.

TCF components as listed in text. The list varies from source to source:

F1	Reliable back-up and recovery
F2	Data communications
F3	Distributed Functions
F4	Performance
F5	Heavily used configuration
F6	Online data entry
F7	Operational ease
F8	Online update
F9	Complex interface
F10	Complex processing
F11	Reusability
F12	Installation ease
F13	Multiple sites
F14	Facilitate change

The degree of influence of each of the 14 components is rated as:

0 - None
1 - Incidental
2 - Moderate
3 - Average
4 - Significant
5 - Essential

$$TCF = 0.65 + 0.01 \sum_{i=1}^{14} F_i$$

i = 1 to 14
0.65 <= TCF <= 1.35

The final calculation of FPs

$$FP = UFC \times TCF$$

Function Points – Example 1

Florida Tech

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

Function Points – Example 1

Specification: Spell Checker

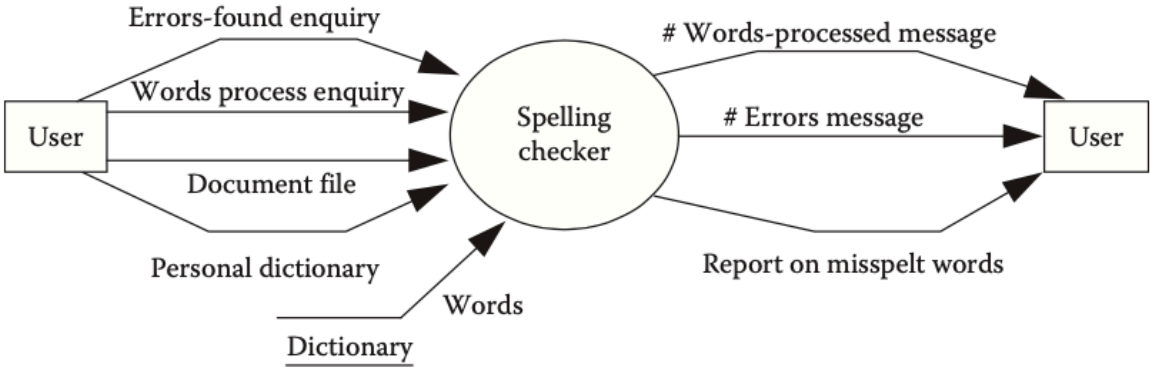
If we assume that the complexity for each item is *average*, then the UFC is

UFC = 4A + 5B + 4C + 10D + 7E = 58

If instead we learn that the dictionary file and the misspelled word report are considered *complex*, then

UFC = 4A + (5× 2 + 7× 1) + 4C + 10D + 10E = 63

Spell-checker spec: The checker accepts as input a document file and an optional personal dictionary file. The checker lists all words not contained in either of these files. The user can query the number of words processed and the number of spelling errors found at any stage during processing.



A = # external inputs = 2, B = # external outputs = 3, C = # inquiries = 2, D = # external files = 2, and E = # internal files = 1

- The two external inputs are: document file-name, personal dictionary-name.
- The three external outputs are: misspelled word report, number-of-words-processed message, number-of-errors-so-far message.
- The two external inquiries are: words processed, errors so far.
- The two external files are: document file, personal dictionary.
- The one internal file is: dictionary.

Function Points – Example 1

If instead we learn that the dictionary file and the misspelled word report are considered *complex*, then

UFC = 4A + (5× 2 + 7× 1) + 4C + 10D + 10E = 63

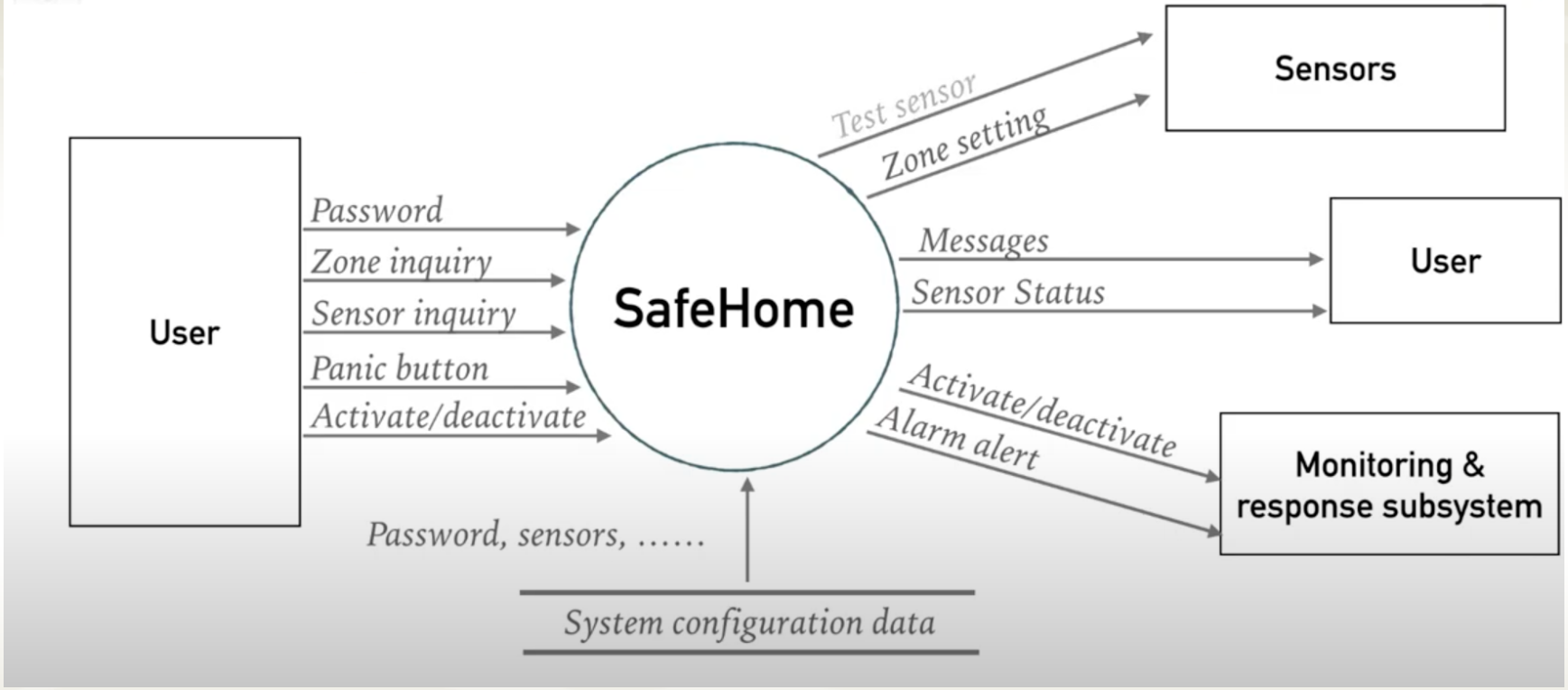
FP computation for the spelling checker, we evaluate the technical complexity factor. After having read the specification, it seems reasonable to assume that F_3, F_5, F_9, F_{11}, F_{12}, and F_{13} are 0, that F_1, F_2, F_6, F_7, F_8, and F_{14} are 3, and that F_4 and F_{10} are 5. Thus, we calculate the TCF as

$$\text{TCF} = 0.65 + 0.01 \sum_{i=1}^{14} F_i$$

TCF = 0.65 + 0.01(18 + 10) = 0.93
Since UFC is 63, then

FP = UFC × TCF
FP = 63 × 0.93 = 59

Function Points – Example 2



A = # External Inputs = 3
B = # External outputs = 2
C = # Inquiries = 2
D = # External files = 4
E = # Internal files = 1

3A + 4B + 3C + 5D + 7E = 50

UFC = 50

FP = UFC x TCF
FP = 50 x 0.93 = 46.4

$$TCF = 0.65 + 0.01 \sum_{i=1}^{14} F_i$$

if we assume that all Fi = 2
then, 14 x 2 = 28
TCF = 0.65 + (0.01 x 28)
TCF = 0.93

Function Points – Example 3

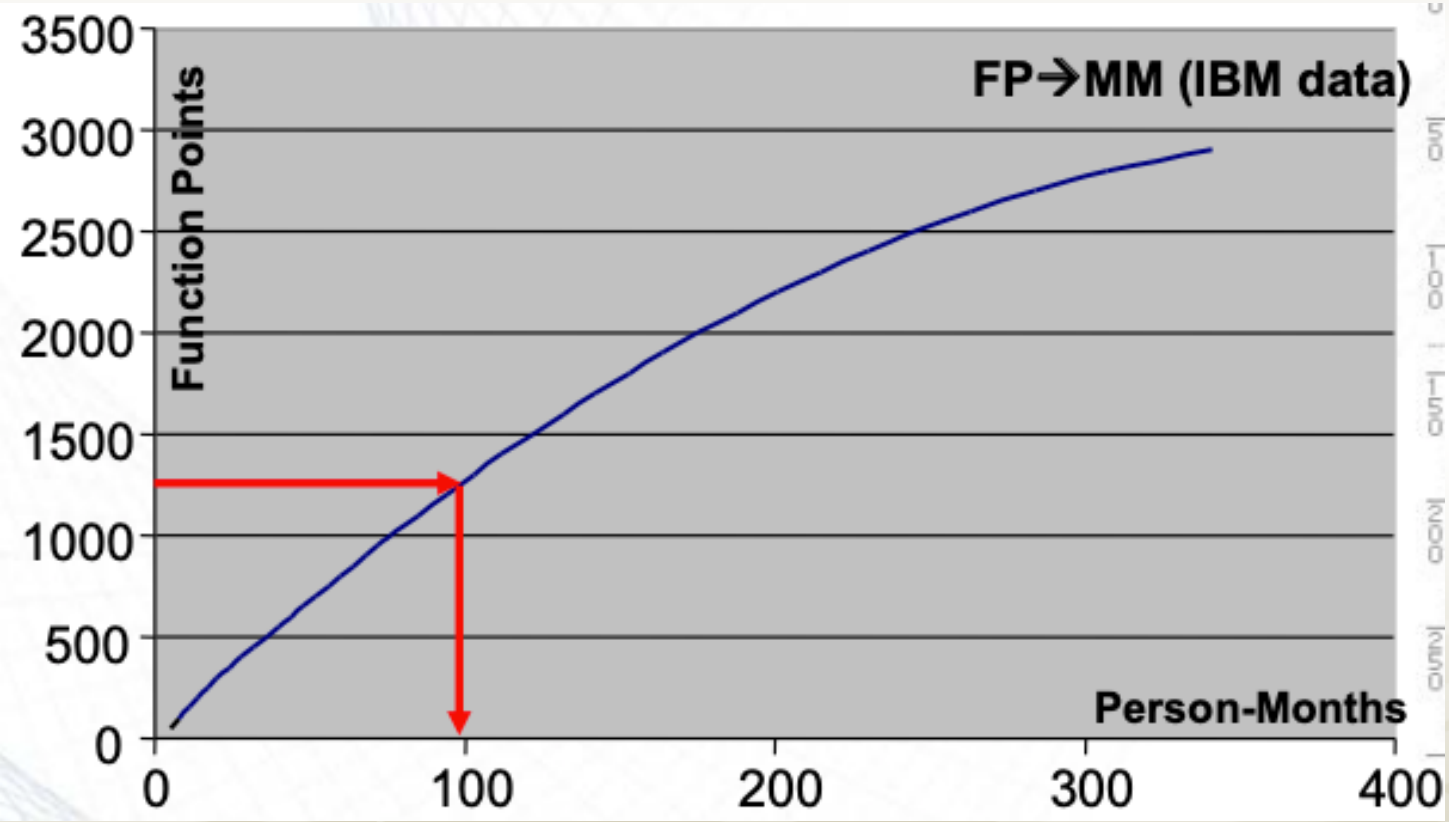
Florida Tech

Function Point Determination From Requirements – an Example

8.8 PERFORMANCE STATISTICS	FUNCT	COUNT	COMPLEXITY	FP
8.8.1 The System shall be able to monitor and display real-time throughput rates for NASCOM blocks.				
a. The throughput rate shall be displayed by site, by port, and for the entire NTS.	Output	1	4	4
b. The throughput rate shall be displayed as a count based on an operator-specific sampling interval. This sampling time interval shall be specific in seconds.	Input	1	3	3
c. The peak traffic rate, based upon the operator-specified sampling time interval, shall be displayed with a corresponding time stamp.	Output	1	4	4
Total Unadjusted Function Points				**11**

Effort Estimation with FP

Effort estimation based on organization specific data from past projects



Function Point Advantages

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

Function Point - Current Status

Florida Tech

1979: Proposed by Albrecht, at IBM

1983: Gained popularity, Albrecht's

1986: International FP user group established (http://www.ifpug.org)

1990~2004: IFPUG, Guidelines to FP measurement

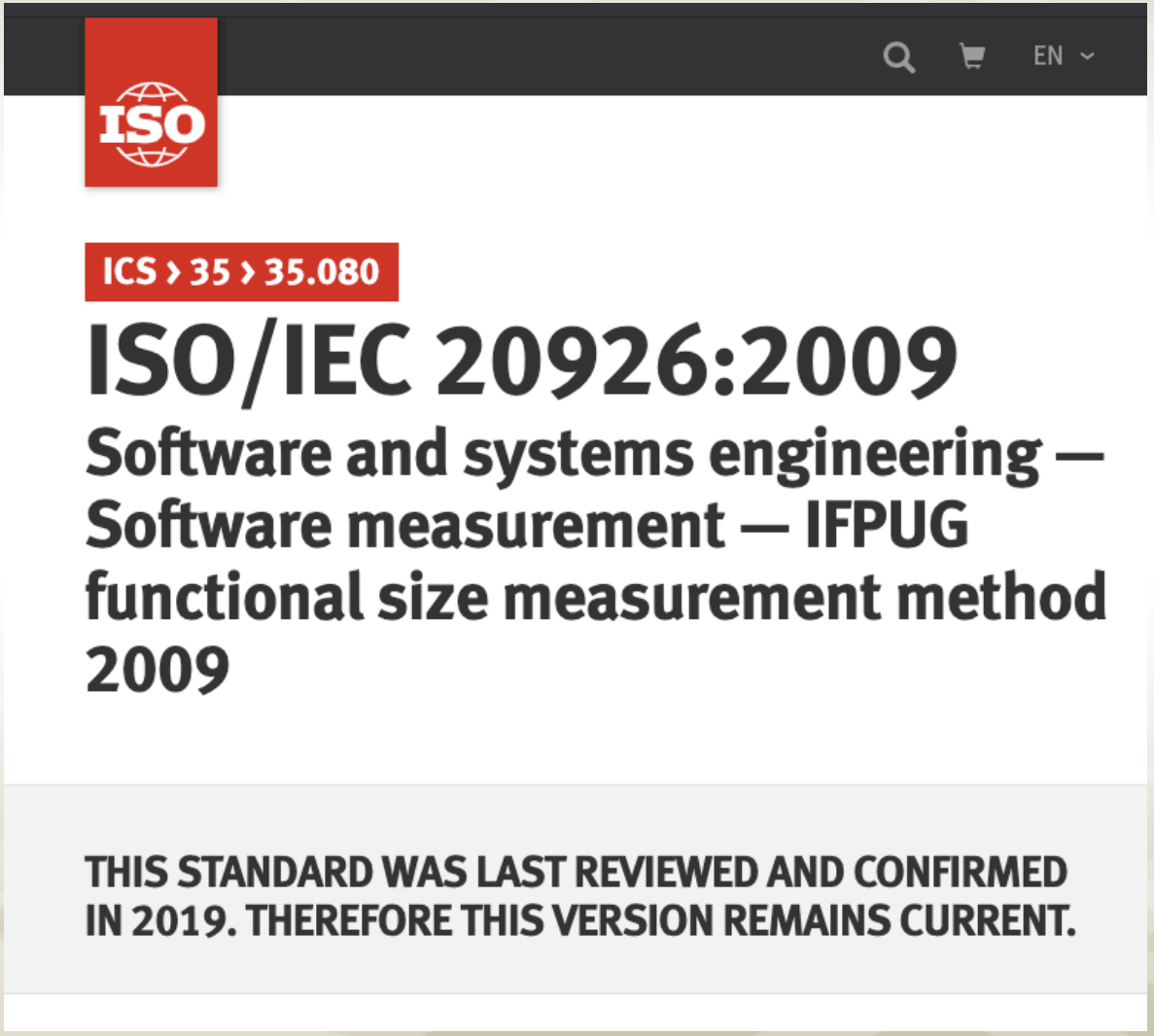
1998: ISO 14143-FSM: Functional Size Measurement

1999: IFPUG, FP counting practice manual, version 4.1

2004: IFPUG, FP counting practice manual, version 4.2

2010: IFPUG. FP counting practice manual, version 4.3

Function Point - SO/IEC 20926:2009



Function Point Limitations

1. Problems with the subjectivity in the TCF
2. Problems with double counting
3. Problems with accuracy
   - Some studies show that TCF does not improve resource estimates
4. Problems with changing requirements - not unique to FPs
   - Function Point counts can increase 400 to 2000% over the lifecycle
   - Requires a full spec
5. Problems with differentiating specified items
   - Reproducibility problems - difficult to automate
6. Problems with application domain
   - Originally developed for data processing
   - How do the rules apply to real-time systems or to games?
7. Problems with subjective weightings
   - Choice of weights was made at IBM over 20 years ago
8. Problems with measurement theory
   - Weights and TCF components are on an ordinal scale with the counts are on a ratio scale, so the linear combination is meaningless from a measurement theory view.