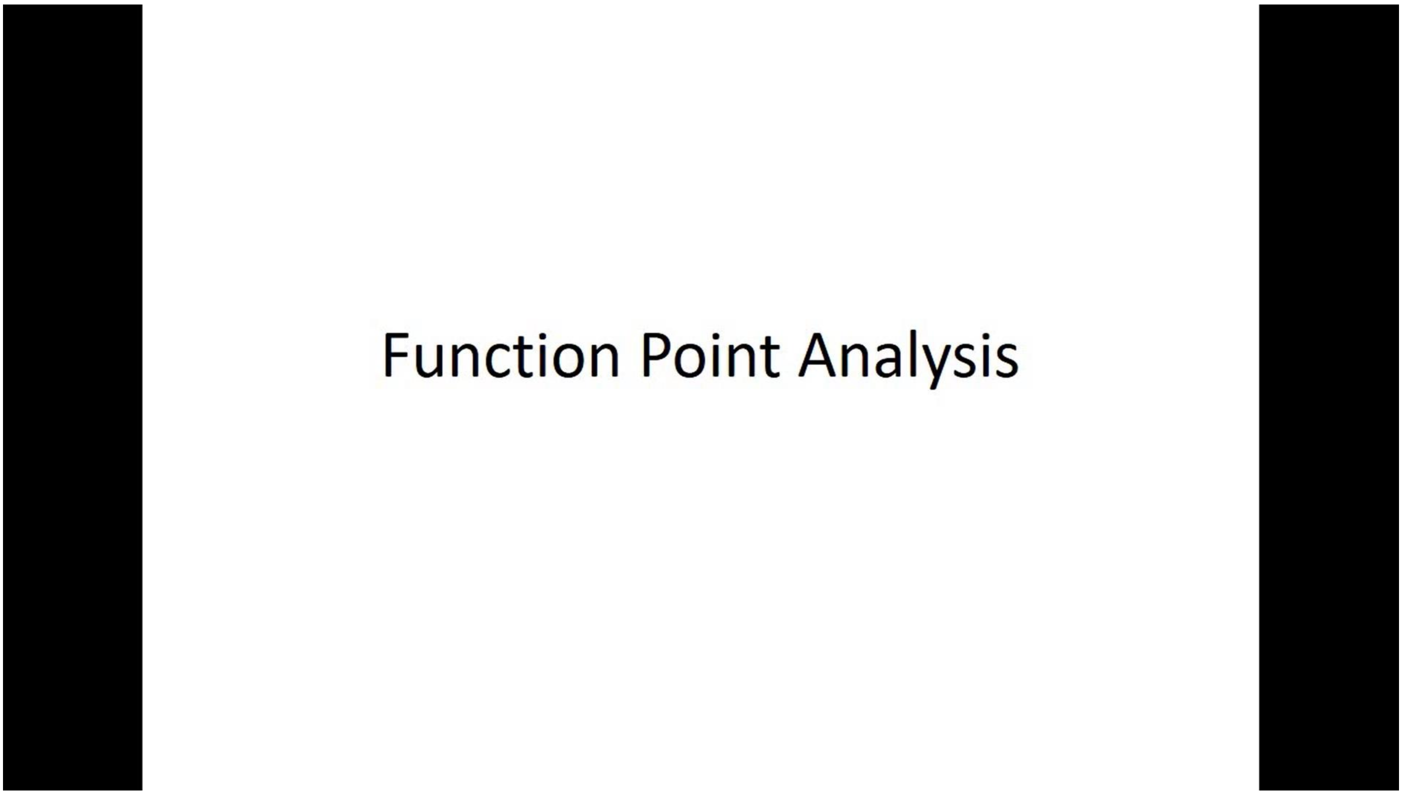
Software Architecture and Quality Engineering

Vikas Bajpai



Function Point Analysis

Function Point Analysis

What is Function Point Analysis (FPA)?

- It is designed to estimate and measure the time, and thereby the cost, of developing new software applications and maintaining existing software applications.

- It is also useful in comparing and highlighting opportunities for productivity improvements in software development.

- It was developed by A.J. Albrecht of the IBM Corporation in the early 1980s.

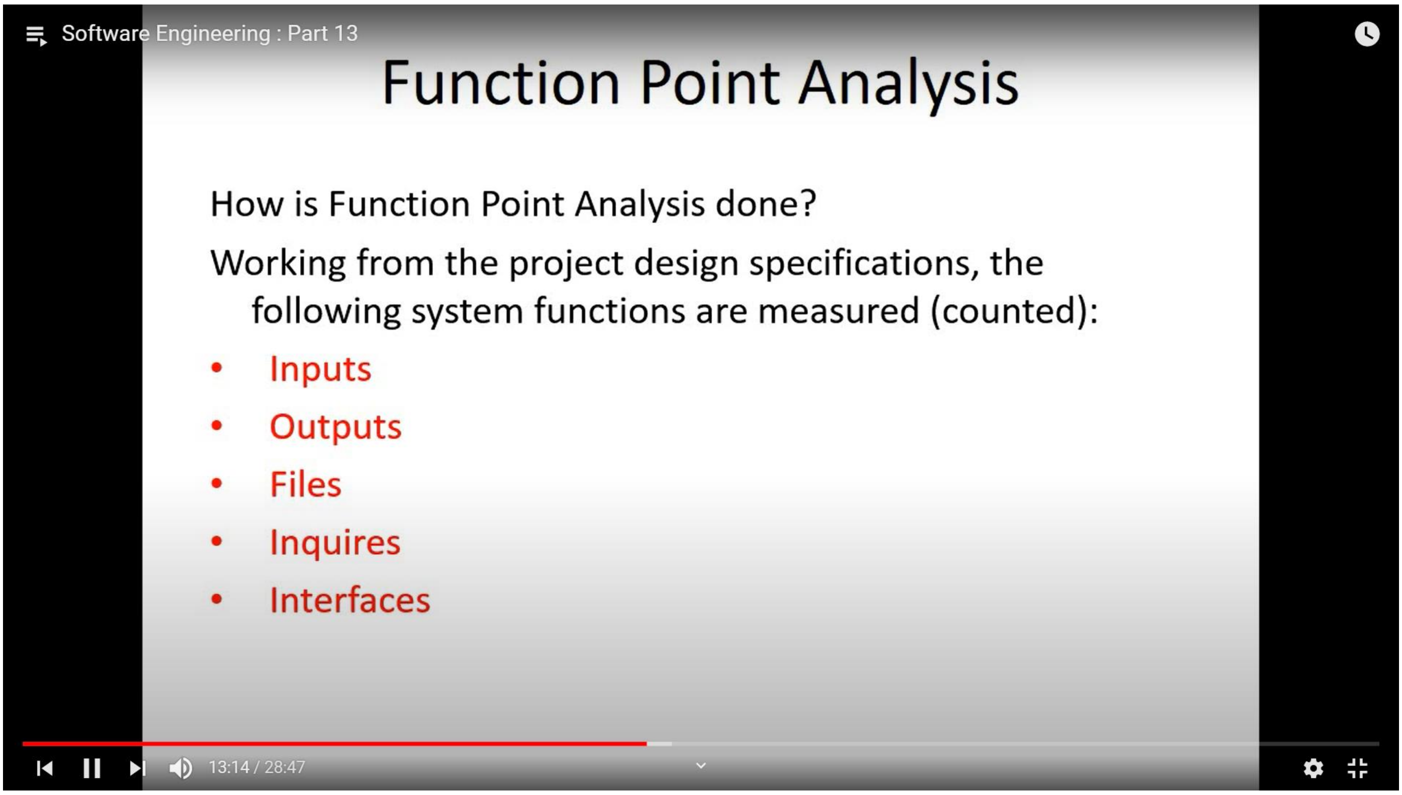
- The main other approach used for measuring the size, and therefore the time required, of software project is lines of code (LOC) – which has a number of inherent problems.

Function Point

- It can easily estimate the size of a software product directly from the problem specification

- Conceptual idea is : size of a software ∞ the number of different functions or features it supports.

- Software product supporting many features would be of larger size

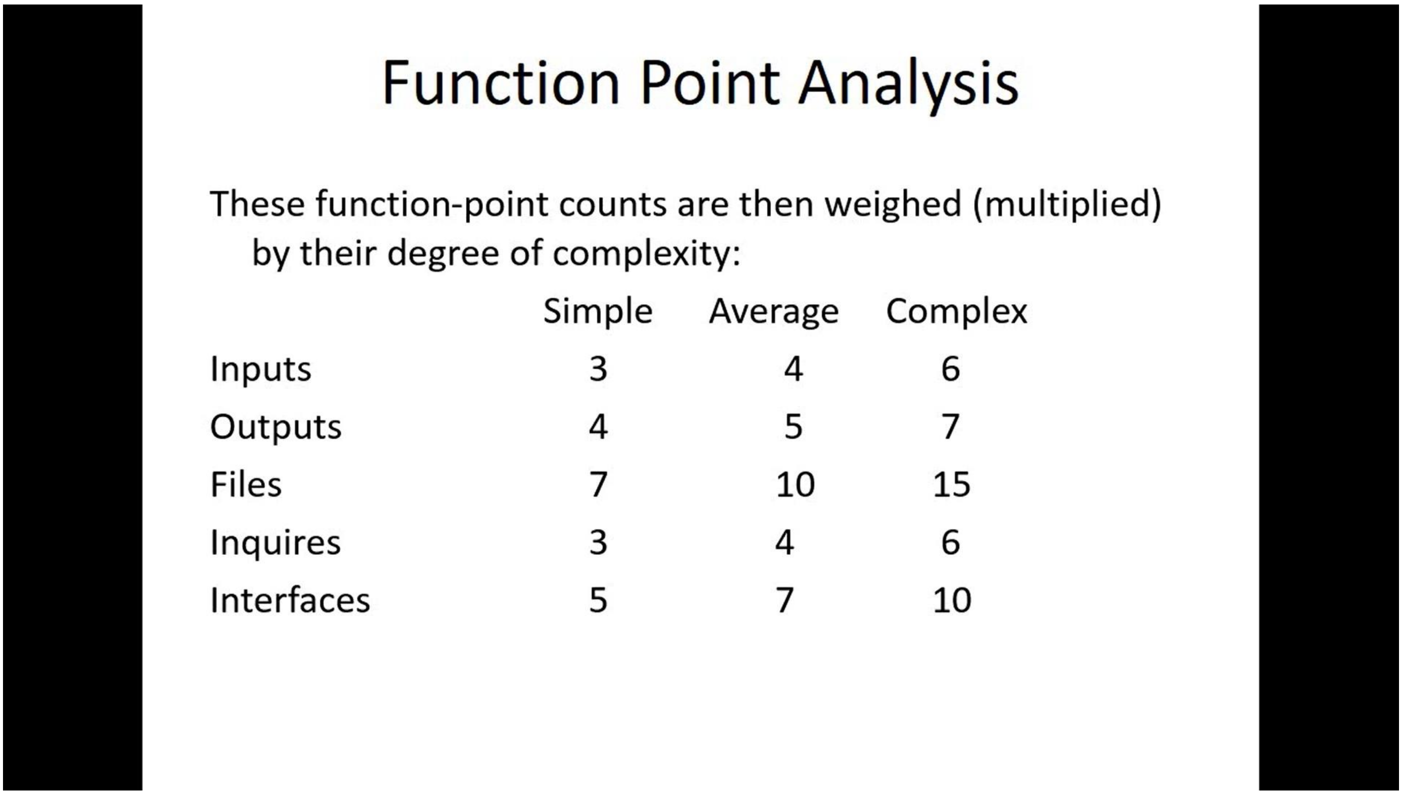


Function Point Analysis

How is Function Point Analysis done?

Working from the project design specifications, the following system functions are measured (counted):

- Inputs
- Outputs
- Files
- Inquires
- Interfaces



Function Point Analysis

These function-point counts are then weighed (multiplied) by their degree of complexity:

	Simple	Average	Complex
Inputs	3	4	6
Outputs	4	5	7
Files	7	10	15
Inquires	3	4	6
Interfaces	5	7	10

Function Point Analysis

A simple example:

inputs

3 simple X 2 = 6

4 average X 4 = 16

1 complex X 6 = 6

outputs

6 average X 5 = 30

2 complex X 7 = 14

files

5 complex X 15 = 75

inquiries

8 average X 4 = 32

interfaces

3 average X 7 = 21

4 complex X 10 = 40

Unadjusted function points 240

For average case…..

UFP = (No. of inputs)*4 + (No. of outputs)*5 + (No. of inquiries)*4 + (No. of files)*10 + (No. of interfaces)*10

Function Point Analysis- TCF

In addition to these individually weighted function points, there are factors that affect the project and/or system as a whole. These are the technical complexity factors. There are a number 14) of these factors that affect the size of the project effort, and each is ranked from “0”- no influence to “6”- essential or strong influence.

The following are some examples of these factors:

- Is high performance critical?
- Is the internal processing complex?
- Is the system to be used in multiple sites and/or by multiple organizations?
- Is the code designed to be reusable?
- Is the processing to be distributed?
- High transaction rates?
- Throughput?
- Response time?
- and so forth . . .

Function Point Analysis

Continuing our example . . .

Complex internal processing	= 3
Code to be reusable	= 2
High performance	= 4
Multiple sites	= 3
Distributed processing	= 5
Project adjustment factor or Degree of influence	= 17

Adjustment calculation:

FP = UFP * DI

Adjusted FP = Unadjusted FP X [0.65 + (adjustment factor X 0.01)]

= 240 X [0.65 + (17 X 0.01)]

= 240 X [0.82]

= 197 Adjusted function points

Function Point Analysis

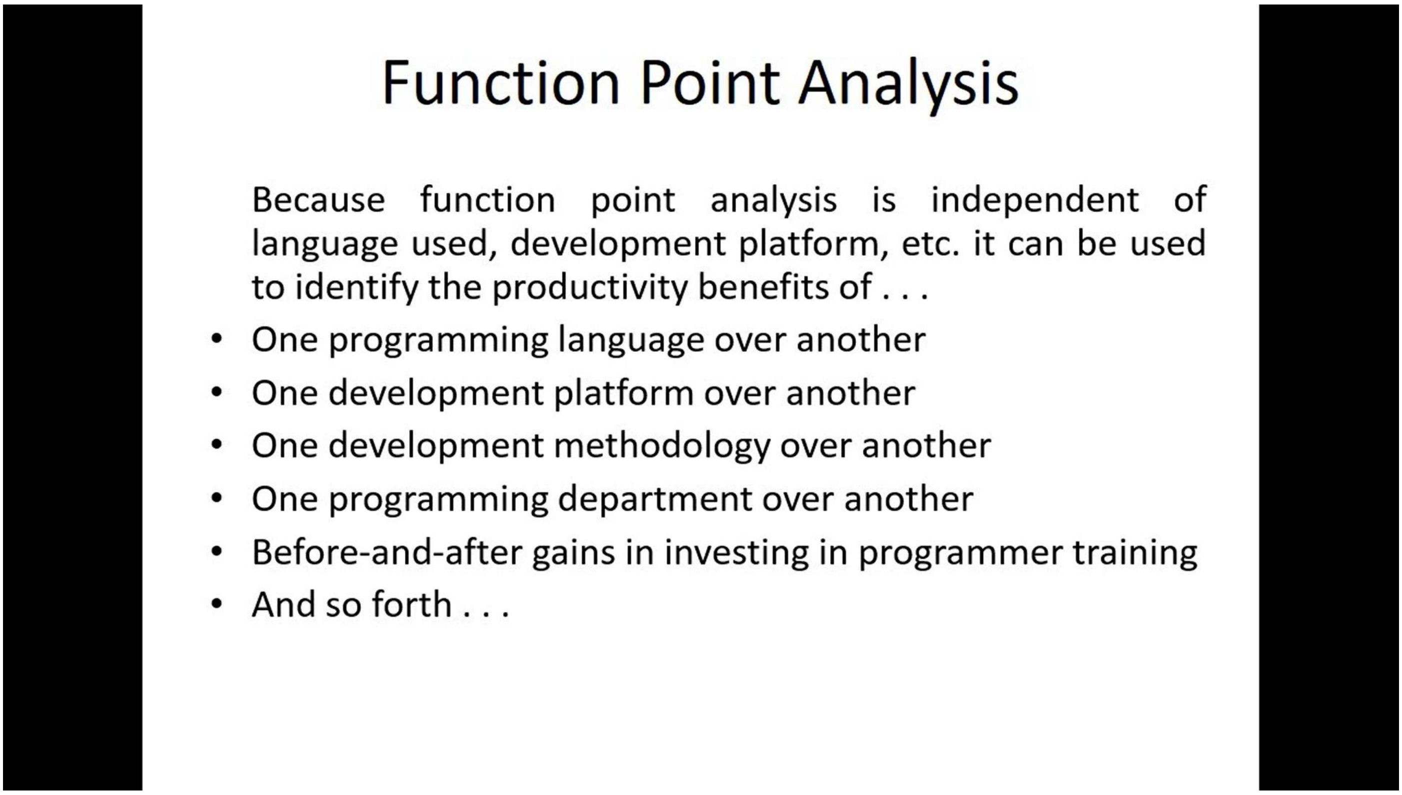
But how long will the project take and how much will it cost?

- Suppose, programmers in an organization average 18 function points per month. Thus . . .

  197 FP divided by 18 = 11 man-months

- If the average programmer is paid $5,200 per month (including benefits), then the [labor] cost of the project will be . . .

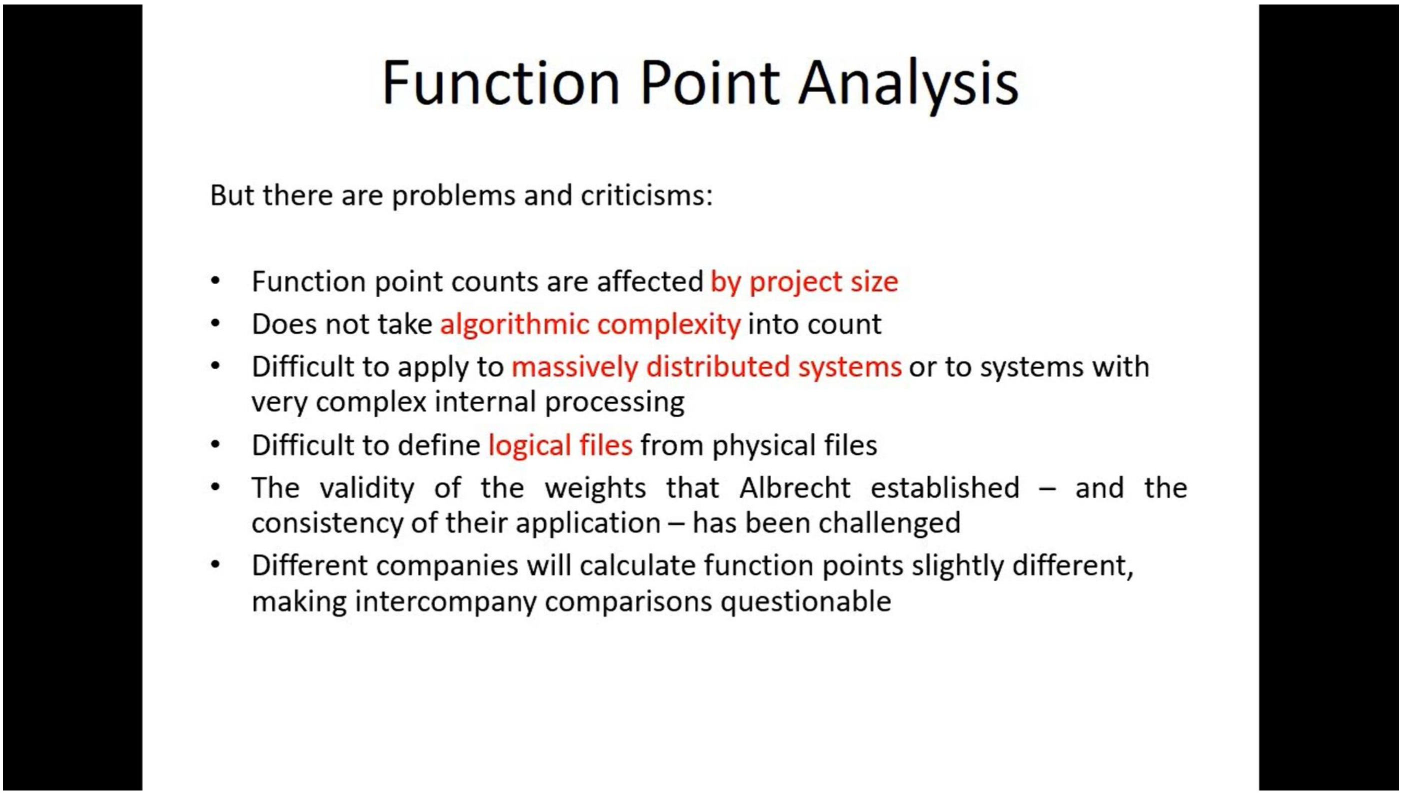
  11 man-months X $5,200 = $57,200



Function Point Analysis

Because function point analysis is independent of language used, development platform, etc. it can be used to identify the productivity benefits of . . .

- One programming language over another
- One development platform over another
- One development methodology over another
- One programming department over another
- Before-and-after gains in investing in programmer training
- And so forth . . .



Function Point Analysis

But there are problems and criticisms:

- Function point counts are affected by project size
- Does not take algorithmic complexity into count
- Difficult to apply to massively distributed systems or to systems with very complex internal processing
- Difficult to define logical files from physical files
- The validity of the weights that Albrecht established – and the consistency of their application – has been challenged
- Different companies will calculate function points slightly different, making intercompany comparisons questionable

Typical SW Function-Oriented Metrics

1) errors per FP (thousand lines of code)

2) defects per FP

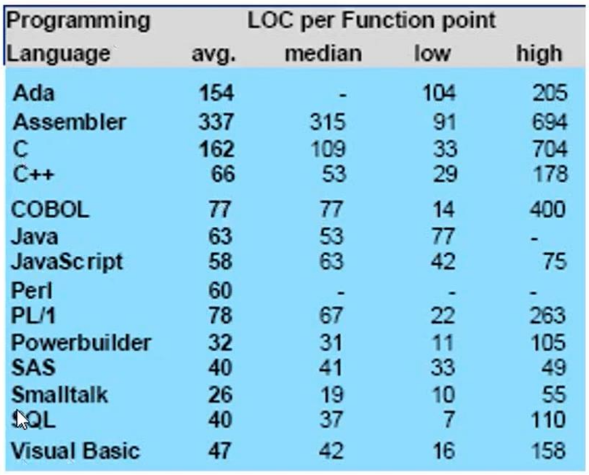
3) $ per FP

4) pages of documentation per FP

5) FP per person-month

Comparing

Comparing LOC and FP



Programming	LOC per Function point			
Language	avg.	median	low	high
Ada	154	-	104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
Java	63	53	77	-
JavaScript	58	63	42	75
Perl	60	-	-	-
PL/1	78	67	22	263
Powerbuilder	32	31	11	105
SAS	40	41	33	49
Smalltalk	26	19	10	55
SQL	40	37	7	110
Visual Basic	47	42	16	158

Representative values developed by QSM