OFFICE OF THE DIRECTOR OF NATIONAL INTELLIGENCE

Estimating SW Costs from Requirements using Objective Function Points Research

Overall Classification

UNCLASSIFIED

Overview

- From the Congressional Notification: Data Science Tradecraft and Standards Initiative:
  - `*"This initiative, in concert with ODNI's Augmenting Intelligence Using Machines (AIM) Strategy, will ensure we maintain the IC's high standards and preserve confidence in the **analytic processes** as we increase the use of machines and **automated methods**"*
- ODNI is collaborating with community on developing more effective/accurate cost/schedule software estimation using **automated methods**
  - Investigating alternatives to estimating Software (SW) via Source Lines Of Code (SLOC) counting, by exploring alternative estimating methods using Function Points (FP) for estimating SW development efforts
    - » This drove the need for automated methods to count/capture FPs
    - » Similar to the UCC standards in SLOC counting where tool would be Open Source
  - This effort focused the IC to initiate an Objective Function Point (OFP) Counting capability into the government managed tool suite (UCC-G) that is requested for each IC MSA program acquisition via CDRL
    - » Automatically calculating OFP's based on International Function Point User Group (IFPUG) documented standards
      - Currently analyzes C, C++, C#, Java and Java Script languages

What are Objective Function Points (OFPs)?

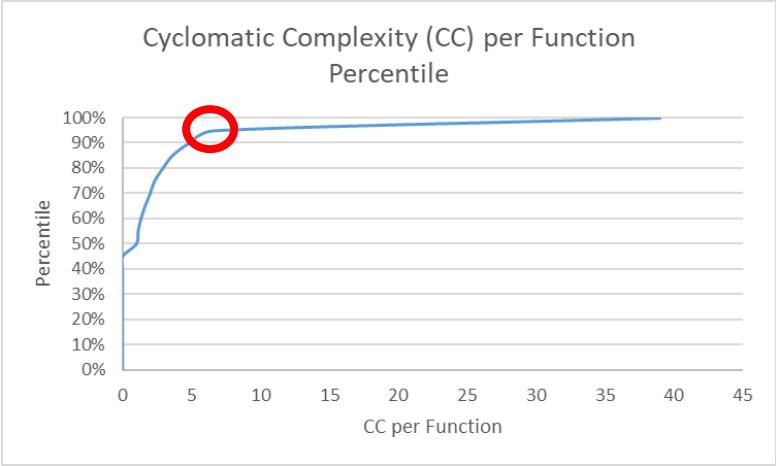
- Developing an Automated Objective Function Point Counter
  - Developing a standard automated approach to counting OFPs to avoid subjective estimates
    - » Standard Function Points (FP) require Function Point experts to derive
    - » Used UCC tool baseline since it already parses through most SW languages and is Open Source
  - Current Function Point estimates use IFPUG tables such as:

1. External Input (EI): Functions that move data into the application without presenting data manipulation.
2. External Output (EO): Functions that move data to user and presents some data manipulation.
3. External Inquiries (EQ): Functions that move data to user without presenting data manipulation.
4. Internal Logical Files (ILF): The logic in the form of fixed data managed by the application using External Input (EI)
5. External Interface Files (EIF): The logic in the form of fixed data used by the application but did not run in it

  - OFPs capture the sizing needed to assess the effort and are based from the IFPUG weights
    - » Cyclomatic Complexity determines which IFPUG table to use

Complexity Mapping to OFPs

- To test the OFPs, we used NASA's General Missions Analysis Tool (GMAT) software
  - Open Source code
- To calculate the OFPs from GMAT code, we can pull out the following data:
  - The Cyclomatic Complexity shows that most of the GMAT code falls less than 10
  - This will drive the OFPs to be the lowest values from the IFPUG tables
- Things to consider on OFPs:
  - OFPs are derived from the actual source code
  - Every Function gets an OFP associated to it
  - There are more developed Functions in the code than what the Function Point experts can predict
  - This will lead to higher OFPs than FPs
    - » Assuming FPs were generated from requirements by an FP expert



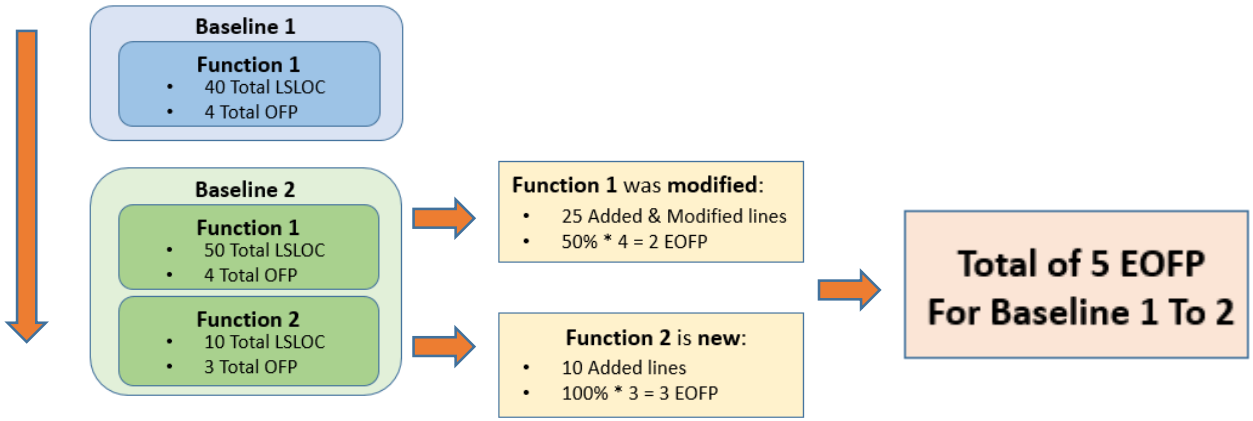
McCabe Complexity Values	McCabe Complexity Definitions	Traditional Function Point Mapping Definitions
1 – 10	• Structured and well written code • High Testability • Cost and Effort is Less	• External Input (EI): Functions that move data into the application without presenting data manipulation.
11 – 20	• Complex Code • Medium Testability • Cost and Effort is Medium	• External Output (EO): Functions that move data to user and presents some data manipulation. • External Inquiries (EQ): Functions that move data to user without presenting data manipulation.
21 – 40	• Very Complex Code • Low Testability • Cost and Effort is Medium	• External Interface Files (EIF): The logic in the form of fixed data used by the application but did not run in it
> 40	• Difficult to test • Very High Cost and Effort	• Internal Logical Files (ILF): The logic in the form of fixed data managed by the application using External Input (EI)

OFPs and Requirements

- 5 Different small test cases were performed with a FP expert
  - FP Expert was provided in all cases UML Sequence and Class Model diagrams
  - All 5 cases had less than a **10**% error between the automated OFPs and the predicted FPs
- This led to a large test case using a subset of the GMAT Formal SW Requirements document
  - The GMAT Formal SW Requirements document is composed of:
    - » Application Control, Resource and Command Objects Functional Requirements
    - » External Interface, Environmental, Computer Resource and Test System Requirements
  - **INITIAL TEST**: Using a subset of the GMAT requirements document (see backup slide for details), the FP expert calculated FPs that showed ~ **200**% error from the OFPs
  - **FINAL TEST**: After providing the FP expert the Unified Modeling Language (UML) documentation, the FP expert calculated FPs that showed ~ **2**% error from the OFPs
- This shows the Uncertainty of estimates are due to the Maturity of the Requirements provided to the estimator
  - Phase A requirements are being refined/defined/matured while some are not specified until the end of Phase B
  - As the requirements get more mature, the Uncertainty of the estimate will go down

Using OFPs to Capture Effort

- OFPs capture the total effort of a baseline as though it was ALL NEW code
- How do we use these if we are trying to capture the effort between baselines or in AGILE's case "Sprints" or "Increments"?
  - We needed a new metric that can utilize the UCC DIFF capabilities
- Created a measure to capture development called **Effective Objective Function Points (EOFPs)**
  - EOFPs are computed by comparing the source code of two baselines
  - After the code is divided into modules / classes or each code baseline, the code is compared and the number of ADDED, MODIFIED and DELETED Logical Source Lines Of Code (LSLOC) is determined for each function within each module
  - For each function, the number of ADDED + MODIFIED lines (as a percentage of total lines) is multiplied by the OFP to determine the EOFP for that function.
  - **The EOFP for the module is simply the sum of the EOFP for all functions**



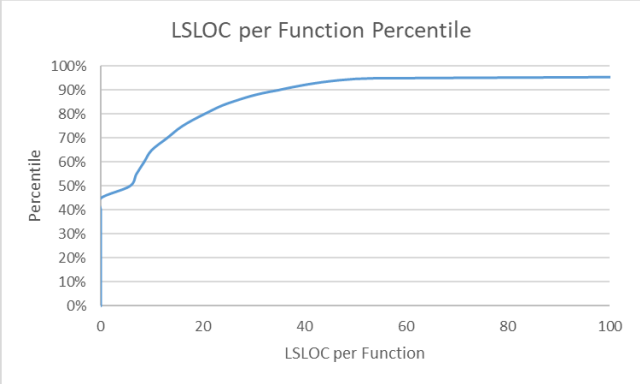
NASA's General Missions Analysis Tool (GMAT) Example

- Results from comparison between GMAT C++ code baselines 2017a and 2018a

	2017a	2018a
Physical Source Lines Of Code (PSLOC)	457,112	535,661
Logical Source Lines Of Code (LSLOC)	290,674	318,173
Delta LSLOC	27,499	
Modules (Classes)	3,104	3,191
OFPs	70,953	74,099
Delta OFPs	3,146	
DIFF Results		
NEW LSLOC	9.3%	
DELETED LSLOC	1.0%	
MODIFIED LSLOC	0.7%	
UNMODIFIED LSLOC	89.0%	
EOFPS	3,023.3	

- Here are some high level GMAT metrics:
  - Total # EOFPs = 3,023.3
  - # Changed/New Modules = 211
  - # Changed/New Functions = 1,473
  - EOFPs / (Changed/New Modules) = 14.33
  - EOFPs / (Changed/New Functions) = **2.05**
  - (Changed/New Functions)/(Changed/New Modules) = **6.98**

- Other potential metrics:
  - EOFPs / Hours
  - LSLOC / Function
    - » Previous UCC counter reported by file and NOT by Function



  - Objective Maintainability Index
    - » GMAT = 135.67 **High Maintainability**

Maintainability Index

85 and more:	**good maintainability**
65-85:	**moderate maintainability**
under 65:	**difficult to maintain**

bad pieces of code (big, uncommented, unstructured) the MI value can be even negative

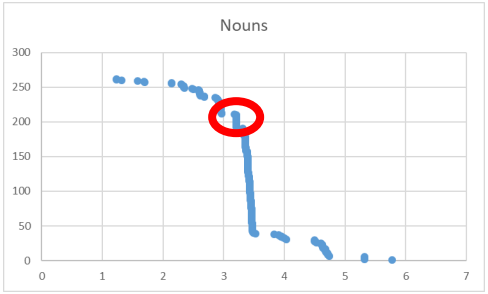
We are opening the door to new metrics to estimate effort

Potential of Using Requirements Documents

- Software Requirement Documents contain nouns and verbs
  - Object Oriented Theory:
    - » Nouns become Modules/Classes
    - » Verbs become Process Functions
- Using Natural Language Toolkit (NLTK) to automatically extract nouns and verbs
    - » This is free open source on the unclass and class side
- NLTK parses out the nouns and verbs from the requirements very well
- In order to identify key words that correlate to effort, we need to calculate weights for the nouns and verbs
  - These weights would be derived by scoring them against the rest of the requirement document
- Due to long runtime with NLTK algorithms when scoring, we investigated using Lucene in place of NLTK to compute Scoring between Module / Class names and individual requirements
  - Lucene combines Boolean model (BM) of Information Retrieval with Vector Space Model (VSM) of Information Retrieval - documents "approved" by BM are scored by VSM
  - Lucene is open source available on high and low side
  - Calibrated Lucene Scoring model to properly map Key Words to Requirements
    - » Calibration involved many hours of many different test cases and individually comparing results
- Runtime of Lucene Scoring outperformed NLTK Scoring

What can we do with Scoring?

- How can we use the Scoring of Nouns and Verbs to help estimate the number of Functions?
  - Following the basic principle from Object Oriented Analysis:
    - » Nouns are potential Modules (classes)
- By using the Scores from the Unique Nouns, we could estimate the number of Modules (Classes)
  - The # of Unique Nouns equals the # of Modules (Classes) when:
    - » Score of **3** or more (see plot of right)
    - » As previously noted, there are **211** GMAT Module (Classes)



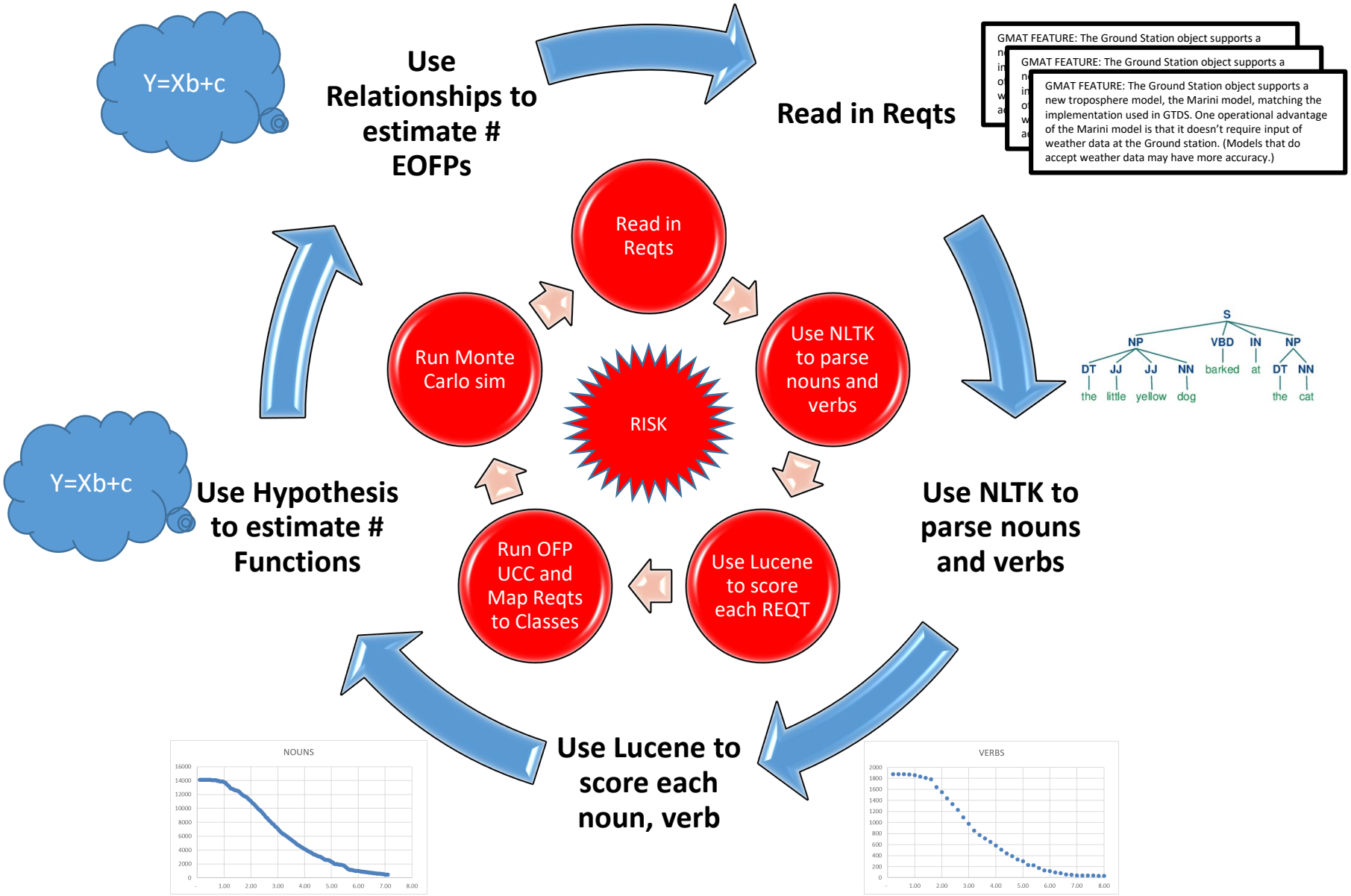
- # Functions currently do not trend to Unique Verbs
  - Verbs such as "Get or "Set" are used many times
  - Requires more research
- As previously noted, there are **6.98** Functions per Module (Class) in GMAT
  - Currently observing other programs in this range **(6 – 7)**
  - Multiplying the 211 Total Changed and New Modules by 6.98, we get **1473** GMAT Functions
- Next challenge is to convert to EOFPs
  - As previously noted, there are **2.05** EOFPs/ Total Changed and New Functions in GMAT
    - » Currently observing other programs in this range **(2 – 3)**
  - **Total Estimated EOFPs = 2.05 * 1473 = 3,019.7**
  - **Total UCC EOFPs = 3,023.3**

Estimating Risk from Requirements

- This Monte Carlo approach provides a Risk Range for the estimate
  - If Story Points are available from data collection efforts, the Story Points map to each requirement where each EOFP maps to Code
    - » Story Points are a subjective means in Agile to relate size to effort
  - If Story Points are NOT available, subjectively map each requirement to the Code Functionality
- Challenge has always been correlating requirements to Function Points
- Scoring paves the way for Monte Carlo approach to correlate requirements to Function Points

- This allows Automapping between Code to Requirements
  - Each requirement can now map to a section of Code as well as EOFPs
  - Thus producing ranges for each requirement
- This bounds the estimate with ranges from the Monte Carlo simulation runs

Summary of Sizing Estimating Process



Overview of Accomplishments

- Accomplishments:
  1. Need an Automated and Objective method to capture sizing
     Objective Function Point (OFP) tool provides this solution √
  2. Need an Automated method to pull nouns and verbs from Requirement documents
     NLTK tool provides this solution √
  3. Need to Correlate code to Requirement descriptions
     Lucene Scoring tool provides a quick solution for each word combination √
  4. Need to isolate specific development to specific SW Requirements
     OFP DIFF tool isolates the development that relates to stories/specific Requirements √
  5. Need to capture new Sizing metric to relate specific development to specific Requirements
     OFP DIFF tool now reports Effective OFPs (EOFPs) √
  6. Need to Map Code to Requirements
     Built a Mapping tool to read in Classes and Methods/Functions and map them to Requirements √
  7. Need to estimate EOFPs based from specific Requirements
     Investigating various hypotheses that will provide confidence in estimating EOFPs √
  8. Need to convert EOFPs to hours
  9. Need to collect more program baselines

POCs

- Govt POC: Michal Bohn MICHALB6@dni.gov
- Presenter: Paul Cymerman pcymerman@quaternion-consulting.com

BACKUP

BACKUP

GMAT Requirements for Unit Level Testing

Groundtrack Plot*	FRR-42.1.0	The system shall allow the user to choose among the following objects as the central body of a ground track plot:
	FRR-42.1.1	1) Default Celestial Body
	FRR-42.1.2	2) User-defined Body
	FRR-42.2.0	The Ground Track Plot shall draw the longitude and latitude time-history for the following object types:
	FRR-42.2.1	1) Spacecraft
	FRR-42.2.3	2) Groundstation
	FRR-42.3.0	The system shall display icons on the ground track to indicate the locations of the following object types:
	FRR-42.3.1	1) Spacecraft
	FRR-42.3.2	2) Groundstation
	FRR-42.4.0	The system shall display object labels next to the icons for the following object types:
	FRR-42.4.1	1) Spacecraft
	FRR-42.4.2	2) Groundstation
	FRR-42.5.0	The system shall allow the user to define the data plotting options for a ground track plot:
	FRR-42.5.1	1) The number of integration steps to skip between plot points
	FRR-42.5.2	2) The number of plot points to collect before updating a ground track plot
	FRR-42.5.3	3) The number of plot points to retain and redraw during propagation and animation.
	FRR-42.6.0	The system shall allow the user to specify how data is drawn to Ground Track Plots during iterative processes such as differential correction, optimization, and estimation. The following options shall be available:
	FRR-42.6.1	1) Show all iterations/perturbations
	FRR-42.6.2	2) Show current iteration/perturbation only
	FRR-42.6.3	3) Show solution only
	FRR-42.7.0	The system shall allow the user to specify a texture map using the following options
	FRR-42.7.1	1) Use default texture map for central body
	FRR-42.7.2	2) Use user-defined texture map.
	FRR-42.8	The system shall optionally display or not display a configured ground track plot
	FRR-42.9	The Ground Track Plot shall display the epoch in UTC Gregorian format
	FRR-42.10	The system shall allow the user to animate the Ground Track Plot after a run is complete
	FRR-42.11	The system shall display the latitude and longitude values when the cursor is placed over a GroundTrackPlot.

Objective Function Point Counting Process on Existing Source Code

- Parse source code for relevant metrics:
  - Modules (classes, or file names for non-OO code and code outside of classes)
    - » Class inheritance
    - » Associations between modules
  - Methods / Functions
    - » Cyclomatic Complexity
  - Attributes (class level variables)
- Cyclomatic Complexity for each method / function is used to determine a which OFP table to use (EI, EO/EQ, ELF, ILF)
  - Complexity < 11 = EI table
  - Complexity < 21 = EO/EQ table
  - Complexity < 41 = ELF table
  - Complexity >= 41 = ILF table
- Use metrics gathered in step 1 to determine which row to choose in the OFP table
  - Class Inheritances (+1) corresponds to OFP RET (Record Element Type)
  - Class Associations correspond to OFP FTR (File Type Reference)
  - Class Attributes correspond to OFP DET (Data Element Type)
  - The number of inheritances, associations, and attributes for a module tells us which row to select in the OFP tables.
    - » The average of the RET/DET and FTR/DET ratios gives us a low, average, or high risk, corresponding to the three rows in each OFP table, so we simply use that knowledge to pick the FP number from our selected table
- Now we have an Objective Function Point number for each method/function. Total them all together and we have the OFP for the source code

Determine Function Point Complexity

- To account for the interfaces in the design of the code, Function Point Theory captures these interfaces through 5 different pieces of data:
  1. External Input (EI): Functions that move data into the application without presenting data manipulation.
  2. External Output (EO): Functions that move data to user and presents some data manipulation.
  3. External Inquiries (EQ): Functions that move data to user without presenting data manipulation.
  4. Internal Logical Files (ILF): The logic in the form of fixed data managed by the application using External Input (EI)
  5. External Interface Files (EIF): The logic in the form of fixed data used by the application but did not run in it
- Based on the 5 above types and the calculated RET, FTR and DET, the Complexity value can be attained by using the look-up tables on the right
  - » **These are standard IFPUG tables**

Table 1 Function Complexity Matrix (Shared table between EI)

Files Type Referenced (FTR)	Data Elements Types (DETs)		
	1-4	5-15	Greater than 15
Less than 2	Low (3)	Low (3)	Average (4)
2	Low (3)	Average (4)	High (6)
Greater than 2	Average (4)	High (6)	High (6)

Table 2 Function Complexity Matrix (Shared table between EO , EQ)

Files Type Referenced (FTR)	Data Elements Types (DETs)		
	1-5	6-19	Greater than 19
Less than 2	Low (4)	Low (4)	Average (5)
2 or 3	Low (4)	Average (5)	High (7)
Greater than 3	Average (5)	High (7)	High (7)

Table 3 Function Complexity Matrix (Shared table ILF)

Record Element Type (RET)	Data Elements Types (DETs)		
	1-19	20-50	51 or More
1	Low (7)	Low (7)	Average (10)
2 to 5	Low (7)	Average (10)	High (15)
6 or More	Average (10)	High (15)	High (15)

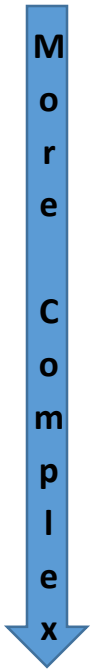
Table 4 Function Complexity Matrix (Shared table EIF)

Files Type Referenced (FTR)	Data Elements Types (DETs)		
	1-19	20 - 50	51 or More
1	Low (5)	Low (5)	Average (7)
2 to 5	Low (5)	Average (7)	High (10)
6 or More	Average (7)	High (10)	High (10)

Cyclomatic Complexity Approach

- UCC already collects Cyclomatic Complexity (CC)
- Objective Function Point (OFP) uses CC as a proxy for the complexity-related inputs to Function Point calculations

McCabe Complexity Values	McCabe Complexity Definitions	Traditional Function Point Mapping Definitions
1 – 10	• Structured and well written code • High Testability • Cost and Effort is Less	• External Input (EI): Functions that move data into the application without presenting data manipulation.
11 – 20	• Complex Code • Medium Testability • Cost and Effort is Medium	• External Output (EO): Functions that move data to user and presents some data manipulation. • External Inquiries (EQ): Functions that move data to user without presenting data manipulation.
21 – 40	• Very Complex Code • Low Testability • Cost and Effort is Medium	• External Interface Files (EIF): The logic in the form of fixed data used by the application but did not run in it
> 40	• Difficult to test • Very High Cost and Effort	• Internal Logical Files (ILF): The logic in the form of fixed data managed by the application using External Input (EI)



Simple Example

- Run counter on a small program that has only 2 Classes resulting in 2 files
- File 1 has 2 Methods/Functions
- File 2 has 1 Method/Function
- Below are results from the Function Point tool:

	RET	FTR	DET	CC	OFP
FILE/Method1.1	2	1	10	10	3
FILE/Method1.2	3	2	20	20	7
FILE/Method2.1	5	3	35	45	10
TOTAL					20

  - Where:
    - » RET: Record Element Type
    - » FTR: File Type Reference
    - » DET: Data Element Type
    - » CC: Cyclomatic Complexity
    - » OFP: Objective Function Points

McCabe Complexity Values	McCabe Complexity Definitions	Traditional Function Point Mapping Definitions
1 – 10	• Structured and well written code • High Testability • Cost and Effort is Less	• External Input (EI): Functions that move data into the application without presenting data manipulation.
11 – 20	• Complex Code • Medium Testability Cost and Effort is Medium	• External Output (EO): Functions that move data to user and presents some data manipulation. • External Inquiries (EQ): Functions that move data to user without presenting data manipulation.
21 – 40	• Very Complex Code • Low Testability • Cost and Effort is Medium	• External Interface Files (EIF): The logic in the form of fixed data used by the application but did not run in it
> 40	• Difficult to test • Very High Cost and Effort	• Internal Logical Files (ILF): The logic in the form of fixed data managed by the application using External Input (EI)