Quaternion CONSULTING

New Design Complexity Interaction with Automatic Objective Function Points

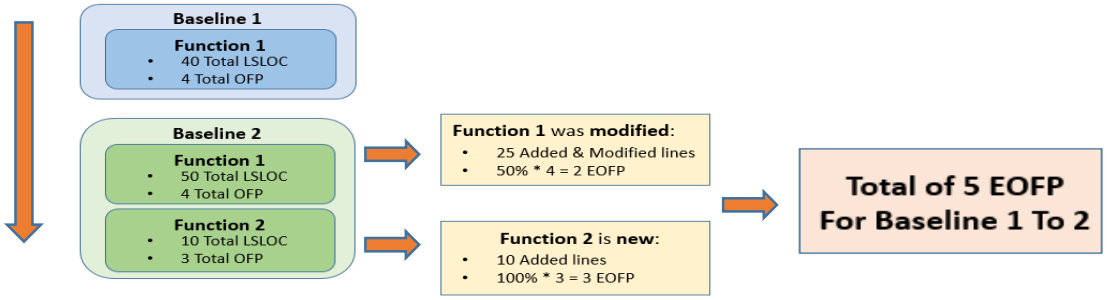
Paul Cymerman, Joe VanDyke, Ian Brown
September 16th, 2020

Overview

- In support of the Office of the Director of National Intelligence (ODNI),we continued to advance Objective Function Point (OFP) Counting capability into the government managed tool suite (UCC-G) that is requested for each Intelligence Community Major System Acquisition (MSA) program acquisition via Contract Data Requirements List (CDRL)
  - Automatically calculating OFP's based on International Function Point User Group (IFPUG) documented standards
    - » Currently analyzes C, C++, C#, Java, Java Script, SQL, and XML languages
- Published in the Global Journal of Computer Science and Technology, Volume 20, Issue 3, Version 1.0, 2020:
  - https://globaljournals.org/GJCST_Volume20/1-Design-Complexity-for-Objective.pdf
- Submitted article to next IFPUG MetricViews Fall 2020 edition
  - This article is an edited version of an article originally published
  - https://www.ifpug.org/metricviews/

Why Objective Function Points (OFPs)?

- The Automated Objective Function Point Counter creates a standard automated approach to counting OFPs to avoid subjective estimates
  - Standard Function Points (FP) require Function Point experts to derive
  - Used UCC-G tool baseline since it already parses through most SW languages and is Open Source
- 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
  - **The EOFP for the Module is simply the sum of the EOFP for all functions**



Is Design Complexity the Same Cyclomatic Complexity?

- The Answer is **NOT EXACTLY**!
- Cyclomatic Complexity (CC) is a software metric used as a limiting function for measuring the complexity of routines during program development
  - This complexity is specific to the ongoing development of routines during overall program development
- McCabe references this as Design Complexity (DC) of the Module but it **does not** address architectural complexity of software design
- The more interactions between objects and the more associations between classes there are, the higher will be the complexity
  - Both the abstract level of the class as well as the physical level of the objects are taken into consideration
  - That would be called the DC of the architecture
- The following statements from Richard Seidl captures the following rational behind DC:
  - "UML Design Complexity metrics can be defined as the relationship of entities to relationships. The size of a set is determined by the number of elements in that set. The complexity of a set is a question of the number of relationships between the elements of that set. The more connections or dependencies there are relative to the number of elements, the greater the complexity."
  - "The more interactions and associations there are between objects and classes, the greater the dependency of those objects and classes upon one another. This mutual dependency is referred to a coupling. Classes with a high coupling have greater domain impacts"

Richard Seidl "Modeling Metrics for UML Diagrams", 2010

Architecture Design Complexity (DC)

- DC is a software metric used to understand the Architecture Design – not just for a specific module, but also between modules. This focuses on the Class (a.k.a. Module), Methods (a.k.a. Functions) and Attributes
  - A class is a set of objects that have common structure and behavior. A class consists of a collection of states (a.k.a. attributes or properties) and behaviors (a.k.a. methods)
  - A method is an operation, which can update the value of the certain attributes of an object
  - An attribute is an observable property of the objects of a class
- The overall Architecture Design considers the additional relationships:
  - Association is a relationship between classes which is used to show that instances of classes could be either linked to each other or combined logically or physically through a semantic relationship
  - Inheritance is a form of Association and a feature of object-oriented programming that allows code reusability when a class includes property of another class
- UML Definition:
  - Sequence diagram
  - State diagram
  - Use case diagram
  - Diagram of activities
  - Package diagram
  - Class diagrams and composite structure

OFP Equation to DC & CC Table

- The math behind this resulted in an Objective Function Point equation dependent on CC and DC:
  - **OFP = (0.125 * DC^2 – 0.125 * DC + 0.25) * CC^2 + (-0.475 * DC^2 + 0.875 * DC – 0.35) * CC + (0.875 * DC^2 – 1.375 * DC + 3.25)**
    - where **DC** = 1 for LOW; 2 for AVG; 3 for HIGH
- We now can simplify to a table that provides the OFPs in a simple form:

OFP	CC_bin=1	CC_bin=2	CC_bin=3	CC_bin=4
DC=0	1	2	3	4
DC=1	3	4	5	7
DC=2	4	5	7	10
DC=3	6	7	10	15

- Could this be used for Model Based System Engineering (MBSE)?
  - MBSE uses SysML where one can use Requirement diagrams to efficiently capture functional, performance, and interface requirements
  - SysML Definition:
    - Definitions for Block Definition and Internal Block Diagrams
    - Changes in the activity diagram
    - Requirements diagram
    - Parametric diagram
    - Allocations (traceability)
  - **Design Complexity in MBSE would be very useful!!!**

OFPs to Objective Module Points (OMPs)

- During the development of the OFPs, it became clear that what the automated OFP tool (UCC-G version) was looking at lower level details that FPA typically does not get a chance to observe
- FP Theory focuses on requirement nouns and verbs which correlate to Modules during development
- OMPs are now counted in the UCC-G tool
  - The equation for OMP is the average of all OFPs that fall under that Module
  - Example:
    - » If a Module has 5 Methods
    - » Each Method will have an OFP associated to it
    - » If each Method has 3 OFPs, that would total 15 OFPs for this Module
  - **To correlate it to traditional FP theory, we take the average of all 5 OFPs**
    - » **3 OMPs in this case**
- We applied the "Diff" capability to provide Effective OFPs (eOFPs) and Effective OMPs (eOMPs). This is driven by the percentage of SLOC that has changed between baselines

Module Name	Function Name	Code	LSLOC	New	Deleted	Modified	Unmodified	New OFP	EOFP	File A	File B
AccessGen	getAccess	N	30	1	1	1	28	3	0.3	U:\SimOrigCode\AccessLib.cs	U:\SimDev.2.6.0.5\AccessLib.cs

The average of the OFPs determine the OMPs which correspond to traditional Function Points

Testing OMPs

- To test this theory, a Certified Function Point Specialist (CFPS) performed an FPA against a set of requirements written to represent the enhancement work for 3 separate software baselines for an ongoing development program
  - Because some information was not available, the CFPS used best judgment to estimate complexity ranges for each module.
  - We then applied rework assumptions (percentages of redesign, recoding, and retesting) consistent with Average Modification enhancements to estimate effective function points (eFP).
- The results demonstrate that the eOMP automated count was consistently within the eFP range, trending toward the lower end
  - Calculated eOMP productivity metrics also look reasonable compared to function point productivity benchmarks.
  - We understand this is a small data set and that additional analysis needs to be conducted, but we believe this approach holds promise.

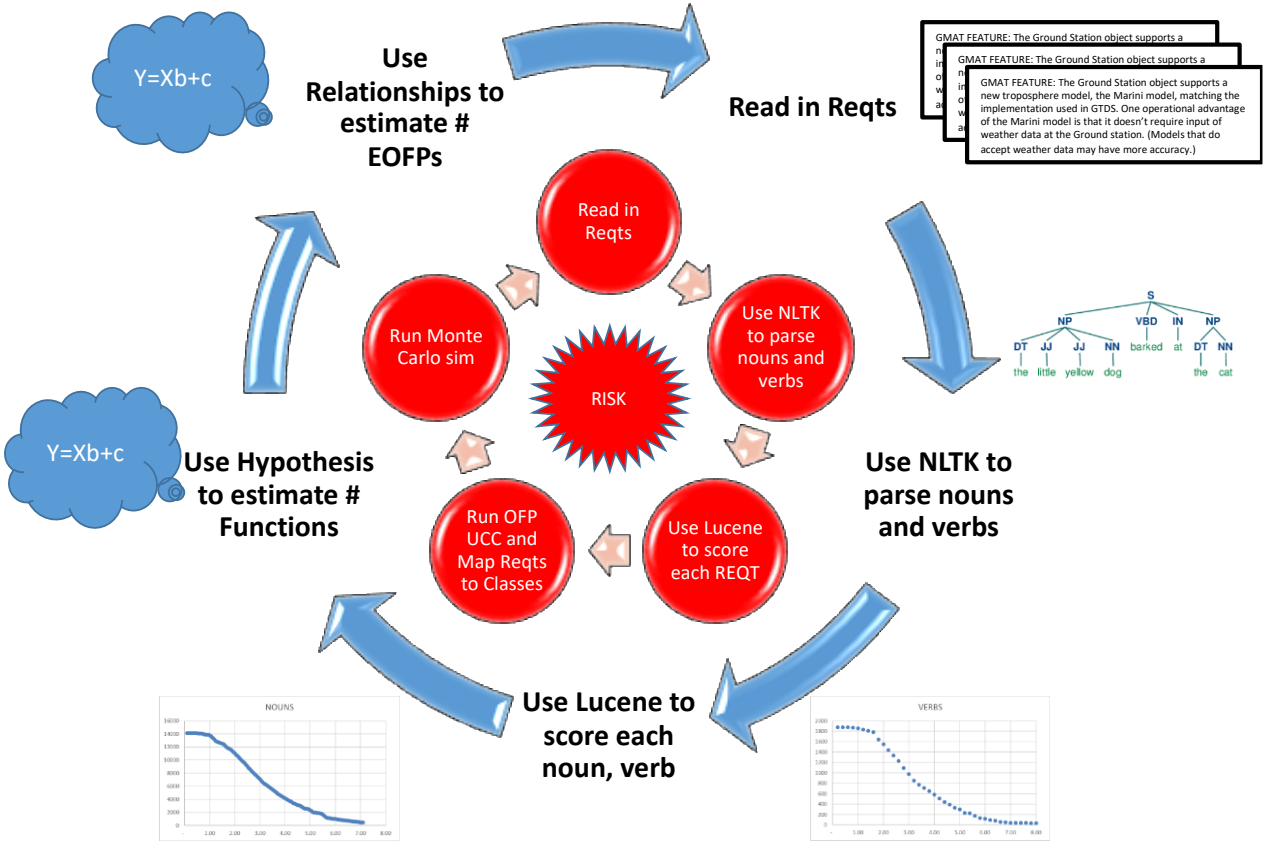
Baseline	eOMP	eff Function Point (LOW)	eff Function Point (LIKELY)	eff Function Point (HIGH)
Sim_2.5 to Sim_2.6.0.5	27.84	24	55	100
Sim_2.6.0.5 to Sim_2.6.0.11	14	12	24	43
Sim_2.6.0.11 to Sim_2.6.0.14	5.09	2	8	15
TOTAL	**46.93**	**38**	**87**	**158**

Date	Baseline	eOFP	eOMP	Hours	Hours/eOFP	Hours/eOMP
6/17/20	Sim_2.5 to Sim_2.6.0.5	199.91	27.84	576	2.9	20.7
7/5/20	Sim_2.6.0.5 to Sim_2.6.0.11	55.73	14	117.5	2.1	8.4
7/27/20	Sim_2.6.0.11 to Sim_2.6.0.14	15.27	5.09	81	5.3	15.9
	TOTAL	**270.91**	**46.93**	**774.5**	**2.9**	**16.5**

Results are within range of published productivities of 12 to 35 Hours per Function Point

Next Steps

- Continue data collections using the UCC-G Objective Function Point
- Continue to investigate using tools such as NLTK and Lucene to help break down requirements
- Goal is to be able to automatically estimate effort based on requirement documents



POCs

- Govt POC: Michal Bohn MICHALB6@dni.gov
- Presenter: Paul Cymerman pcymerman@quaternion-consulting.com
- Questions on UCC-G: uccg@centauricorp.com
  - Version 1.4 coming out Oct/Nov 2020
  - Will include OFP

BACKUP

BACKUP

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