Construction of a Generic Program Representation for Automated Metric Computation

ZACHARY MCNELLIS

APRIL 2016

motivation

- Little progress in programming instruction
  - Several online tools to foster learning
  - None evaluate quality
- No single tool to compute software metrics

contribution

1. Generic representation for metric computation
   - Language neutral
   - Interoperable
   - Standard metric interpretations
   - Use existing tools
2. Implementation
   - Python/C++
   - Demonstrate & validate results
3. Website prototype

overview

- **Software quality**
- Software metrics
- GAST Framework
- Results

software quality

- What is it?
  - “the standard of something as measured against other things of a similar kind”
  - Functional Requirements
  - Non-functional Requirements
- How do we measure it?

overview

- Software quality
- **Software metrics**
- GAST Framework
- Results

software metrics

- Quantitative guide to performance of software
- Selection of Metrics
  - No universal metric
- Process, Product, Resource
  - Software Code Metrics (product)
- By paradigm
- Prioritized quality attributes

software metrics

1. SLOC
2. McCabe
3. Halstead

SLOC

- Advantages
  - Simple
- Disadvantages
  - Not robust
  - Different languages?
  - Infamous
- /* What is a meaningful line of code? */
  - Many variations

McCabe Cyclomatic Complexity

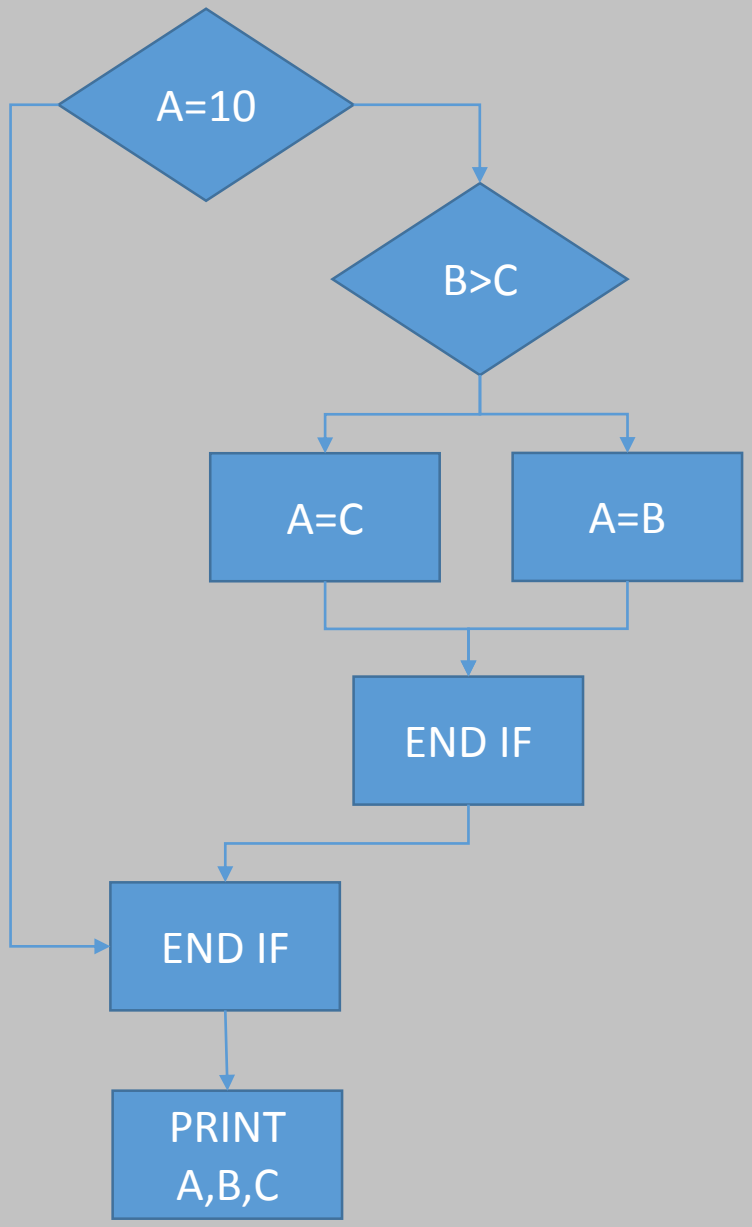
- Thomas McCabe 1976
- Code complexity impacts:
  1. Understanding
  2. Maintaining
  3. Explaining
  4. Updating
  5. Design
  6. Availability

McCabe Cyclomatic Complexity

- Control Flow Graph (CFG)
  - Directed graph
  - Edges = possible flows of control
  - Nodes = basic blocks of program
- $V(G) = E - N + 2P$
  - E: # Edges
  - N: # Nodes
  - P: # Disconnected Pieces
- Control statements?
  - If, While, For …

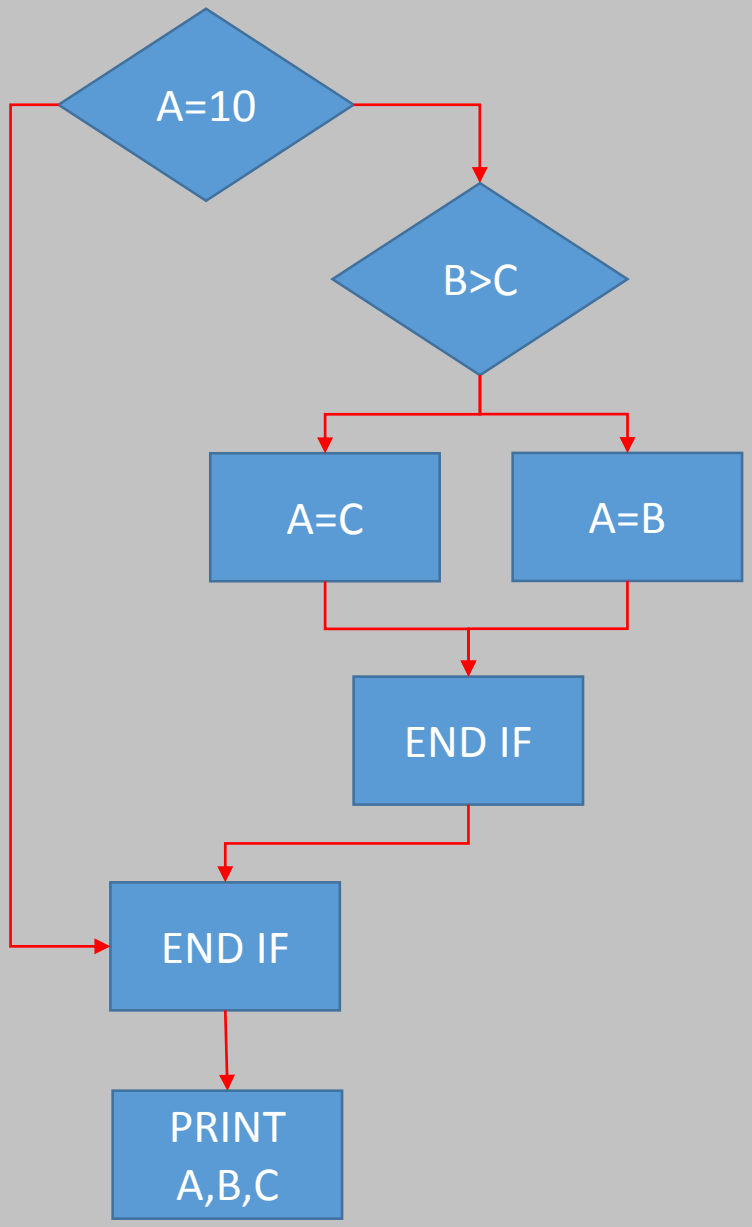
```
IF A = 10 THEN
   IF B > C THEN
      A = B
   ELSE
      A = C
   ENDIF
ENDIF
Print A
Print B
Print C
```

E = ??
N = ??
P = ??



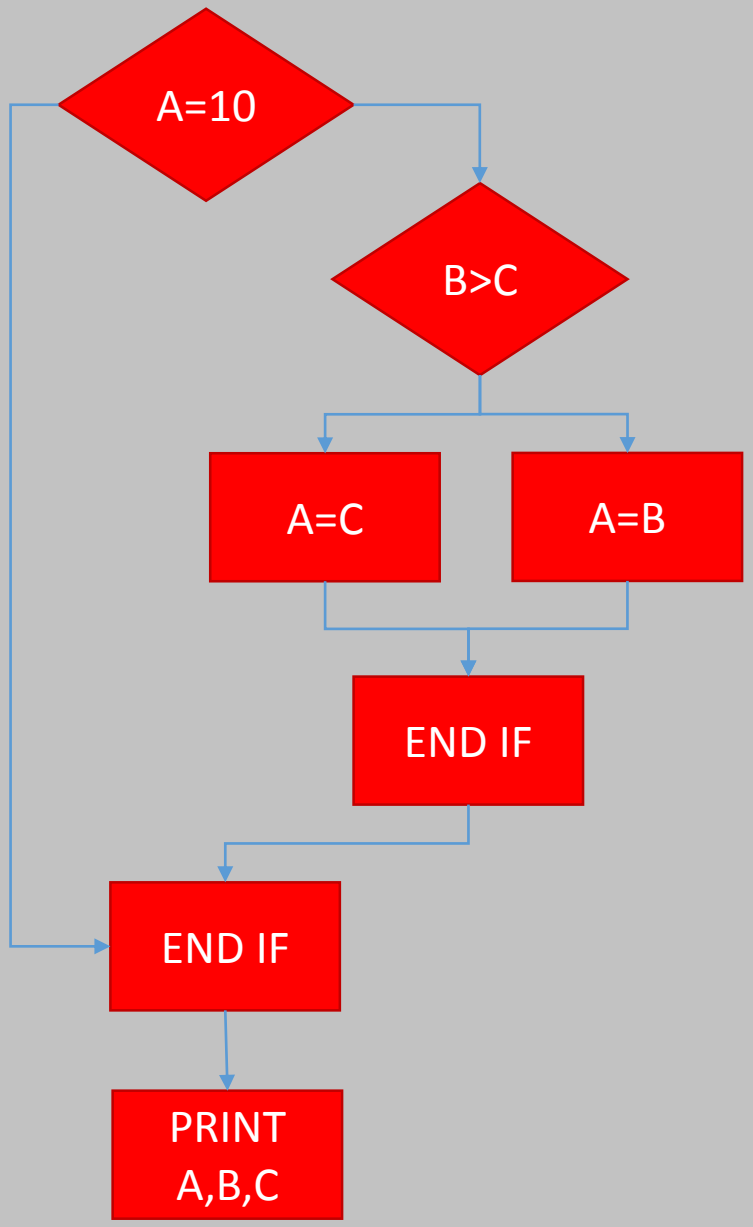
```
IF A = 10 THEN
   IF B > C THEN
      A = B
   ELSE
      A = C
   ENDIF
ENDIF
Print A
Print B
Print C
```

E = 8
N = ??
P = ??



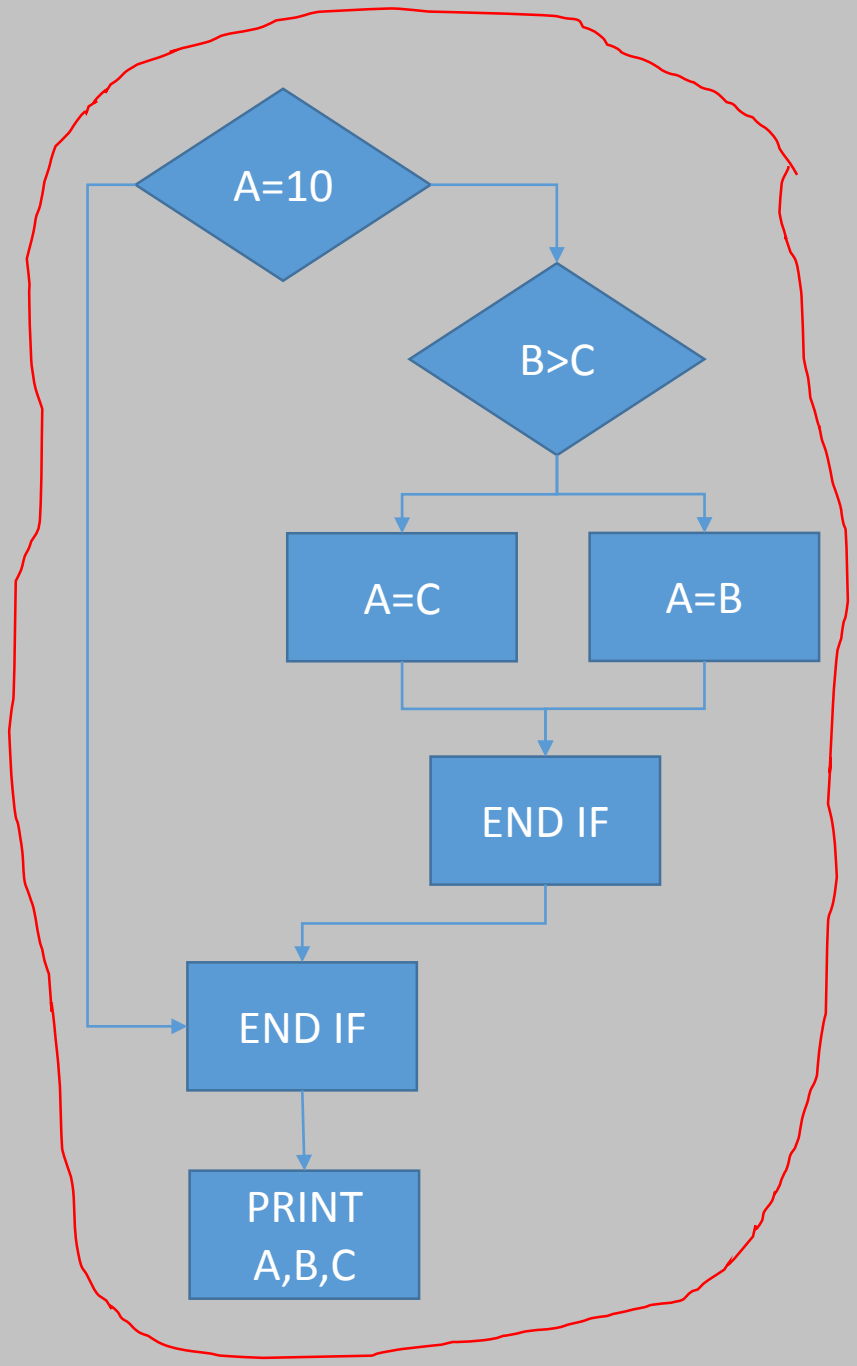
```
IF A = 10 THEN
   IF B > C THEN
      A = B
   ELSE
      A = C
   ENDIF
ENDIF
Print A
Print B
Print C
```

E = 8
N = 7
P = ??



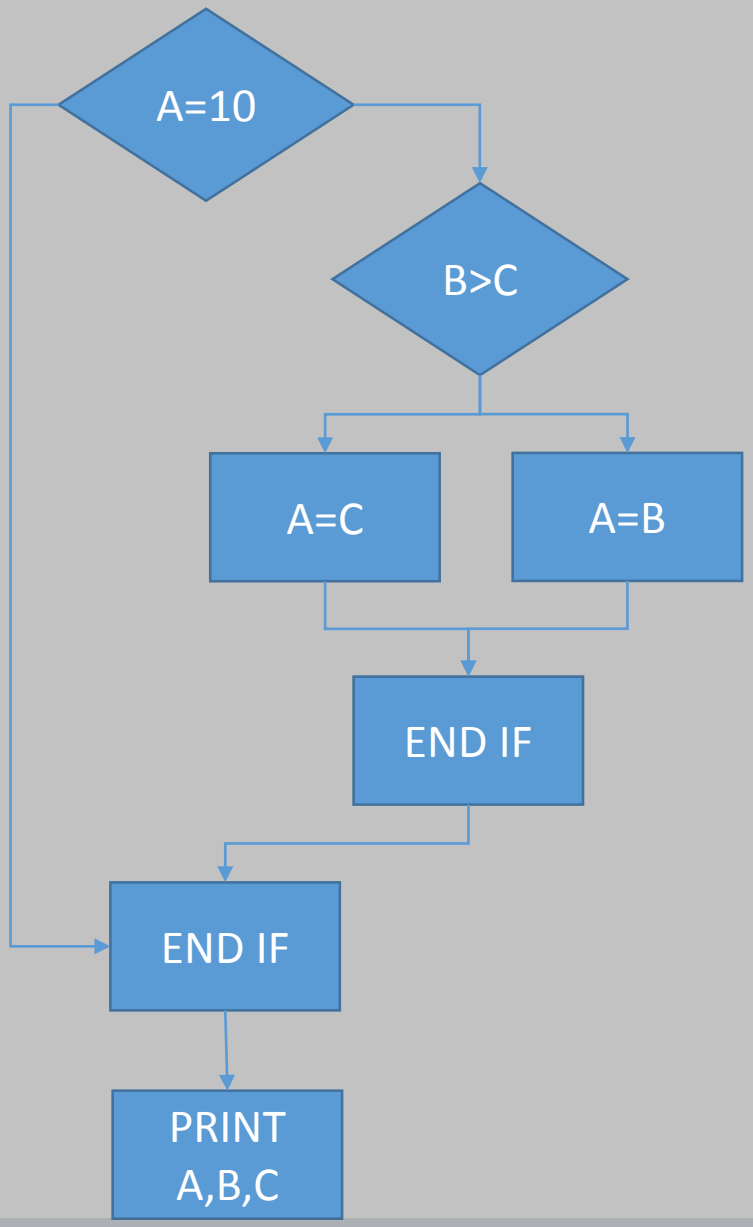
```
IF A = 10 THEN
   IF B > C THEN
      A = B
   ELSE
      A = C
   ENDIF
ENDIF
Print A
Print B
Print C
```

E = 8
N = 7
P = 1



```
IF A = 10 THEN
   IF B > C THEN
      A = B
   ELSE
      A = C
   ENDIF
ENDIF
Print A
Print B
Print C
```

E = 8
N = 7
P = 1



Cyclomatic Complexity: 8 – 7 + 2 = 3

McCabe Cyclomatic Complexity

- Advantages
  - Relative Complexity
  - Maintenance
  - Many existing tools
- Disadvantages
  - Ambiguous
  - May not match human interpretation

McCabe Cyclomatic Complexity

v1

```
def foo(a, b):
   distance = 0;
   if (a[0] != b[0]):
      distance += 1;
   if (a[1] != b[1]):
      distance += 1;
   if (a[2] != b[2]):
      distance += 1;
   return distance;
```

v2

```
def bar(a, b):
   count = 2;
   distance = 0;
   while (count>-1):
      distance = 0;
      for i in range(count, 3):
         if (int(a[i]) ^ int(b[i]) != 0):
            distance = distance + 1;
      count = count - 1;
   return distance;
```

Halstead Software Science

- Maurice Howard Halstead 1977
- Suite of metrics
- Let:
  - η_1=distinct operators
  - η_2=distinct operands
  - N_1=total operators
  - N_2=total operands

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $\mathrm{N} = N_1 + N_2$
- Volume: $\mathrm{V} = \mathrm{N} \times log_2 \eta$
- **Difficulty:** $\boldsymbol{D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}}$
- Effort: $E = D \times V$

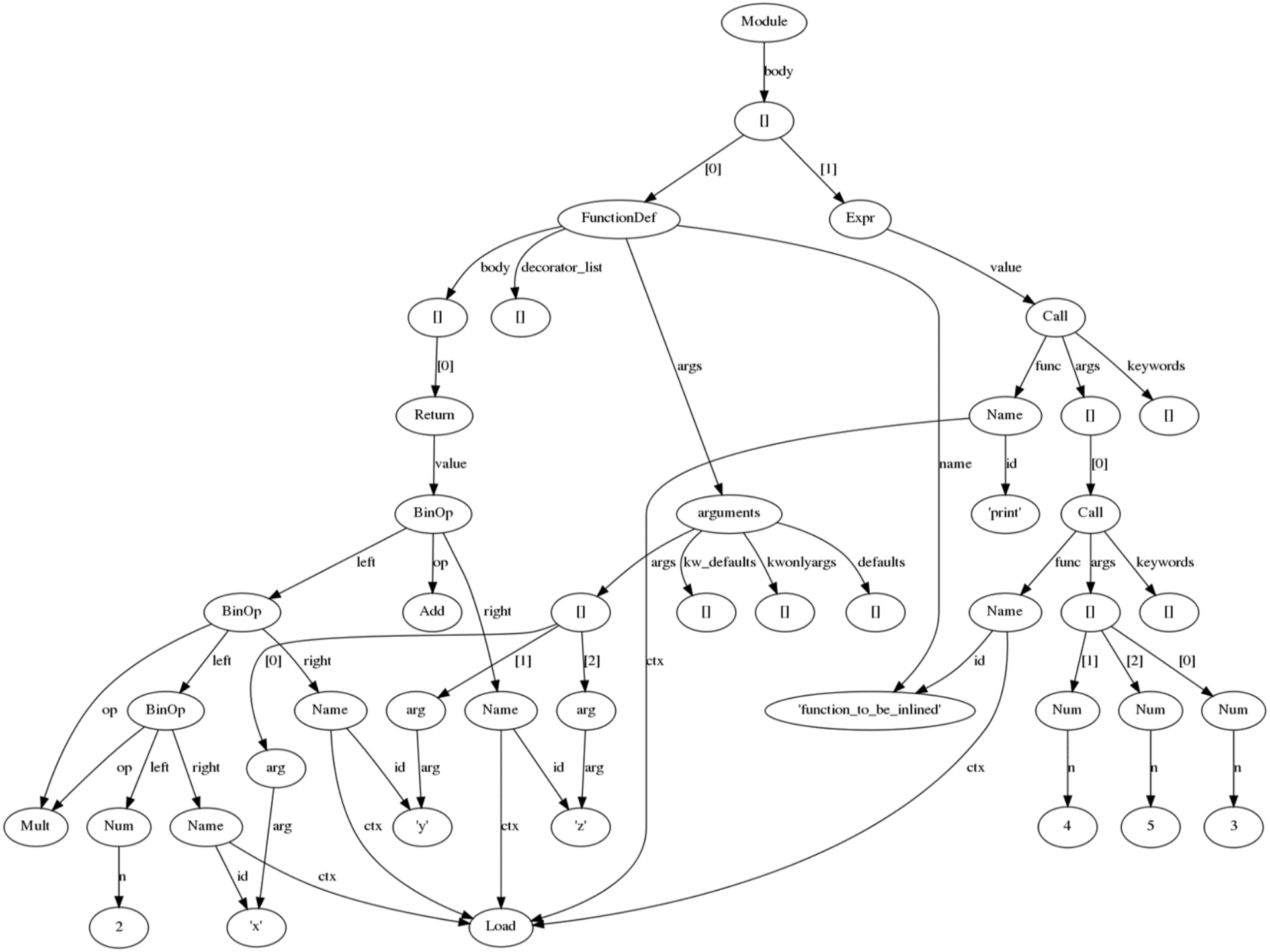
Halstead Software Science

- Advantages
  - Different properties
  - Input for other metrics
- Disadvantages
  - Difficult to compute
  - Distinguish operand/operator?

How do we actually compute these software metrics?

abstract syntax tree

- Tokenization
  - Need more abstract representation
- Solution: Use an Abstract Syntax Tree (AST)
  - Abstract hierarchical data structure
  - Overall structure without concrete details
- Many open-source tools


Module
body
[]
[0]
[1]
FunctionDef
Expr
body
decorator_list
args
value
[]
[]
Call
[0]
func
args
keywords
Return
Name
[]
[]
value
name
id
[0]
BinOp
arguments
'print'
Call
left
op
args
kw_defaults
kwonlyargs
defaults
func
args
keywords
BinOp
Add
right
[]
[]
[]
[]
Name
[]
[]
left
[0]
right
[1]
[2]
ctx
id
[1]
[2]
[0]
op
BinOp
Name
arg
Name
arg
'function_to_be_inlined'
Num
Num
Num
op
left
right
arg
id
arg
id
arg
ctx
n
n
n
Mult
Num
Name
arg
ctx
'y'
ctx
'z'
4
5
3
n
id
ctx
2
'x'
Load

overview

- Software quality
- Software metrics
- **GAST Framework**
- Results

STEP 1: AST Construction

- **Input:** source code
- **Output:** AST in XML
- Leverage parser front-end APIs
  - Clang
  - Python *ast* module

STEP 2: Generic AST Construction

- **Input:** AST in XML
- **Output:** GAST in XML
- GAST
  - **G**eneric **A**bstract **S**yntax **T**ree
- Design
  - Java, Objective C, Objective C++, C++, Python
  - Extensible
  - Interoperable
  - Standard metric interpretations

example

```
1  #include <iostream>
2
3  int main() {
4    int a = 10;
5    int b = a / 2;
6
7    if (a > b) {
8      std::cout<< a <<std::endl;
9    }
10   else {
11     std::cout<< b <<std::endl;
12   }
13
14   return 0;
15 }
```

example

```
<?xml version="1.0"?>
<gast filename="example.cpp">
  <TRANSLATION_UNIT>
    <FUNCTION_DECL lineno="3" col_offset="5">
      <COMPOUND_STMT lineno="3" col_offset="12">
        <DECL_STMT lineno="4" col_offset="3">
          <VAR_DECL lineno="4" col_offset="7">
            <INTEGER_LITERAL lineno="4" col_offset="11"/>
          </VAR_DECL>
        </DECL_STMT>
        <DECL_STMT lineno="5" col_offset="3">
          <VAR_DECL lineno="5" col_offset="7">
            <BINARY_OPERATOR lineno="5" col_offset="11">
              <UNEXPOSED_EXPR lineno="5" col_offset="11">
                <DECL_REF_EXPR lineno="5" col_offset="11"/>
              </UNEXPOSED_EXPR>
              <INTEGER_LITERAL lineno="5" col_offset="15"/>
            </BINARY_OPERATOR>
          </VAR_DECL>
        </DECL_STMT>
        <IF_STMT lineno="7" col_offset="3">
```

example

```
<?xml version="1.0"?>
<gast filename="example.cpp">
  <TranslationUnit>
    <FunctionDefinition lineno="3" col_offset="5">
      <Unknown lineno="3" col_offset="12">
        <Unknown lineno="4" col_offset="3">
          <Unknown lineno="4" col_offset="7">
            <Num lineno="4" col_offset="11"/>
          </Unknown>
        </Unknown>
        <Unknown lineno="5" col_offset="3">
          <Unknown lineno="5" col_offset="7">
            <BinaryOperator lineno="5" col_offset="11">
              <Unknown lineno="5" col_offset="11">
                <Unknown lineno="5" col_offset="11"/>
              </Unknown>
              <Num lineno="5" col_offset="15"/>
              <BinaryOperatorMeta>
                <rand>a</rand>
                <tor>/</tor>
                <rand>2</rand>
              </BinaryOperatorMeta>
            </BinaryOperator>
          </Unknown>
        </Unknown>
        <If lineno="7" col_offset="3">
```

STEP 3: Validation

- **Input:** GAST in XML
- **Output:** Valid GAST
- Explicitly defines GAST structure
- XML Schema Definition (XSD)

```
<!-- simple element definitions -->
<xs:element name="TranslationUnit" type="CursorKind"/>

<!-- attribute definitions -->
<xs:attribute name="filename" type="xs:string"/>
<xs:attribute name="lineno" type="xs:positiveInteger"/>
<xs:attribute name="col_offset" type="xs:nonNegativeInteger"/>
<xs:attribute name="mccabe" type="xs:nonNegativeInteger"/>

<!-- GAST Cursor Kinds -->
<xs:complexType name="CursorKind">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="BinaryOperator" maxOccurs="unbounded"/>
    <xs:element ref="UnaryOperator" maxOccurs="unbounded"/>
    <xs:element ref="TernaryOperator" maxOccurs="unbounded"/>
    <xs:element name="FunctionDefinition" type="CursorKind"
maxOccurs="unbounded"/>
```

…

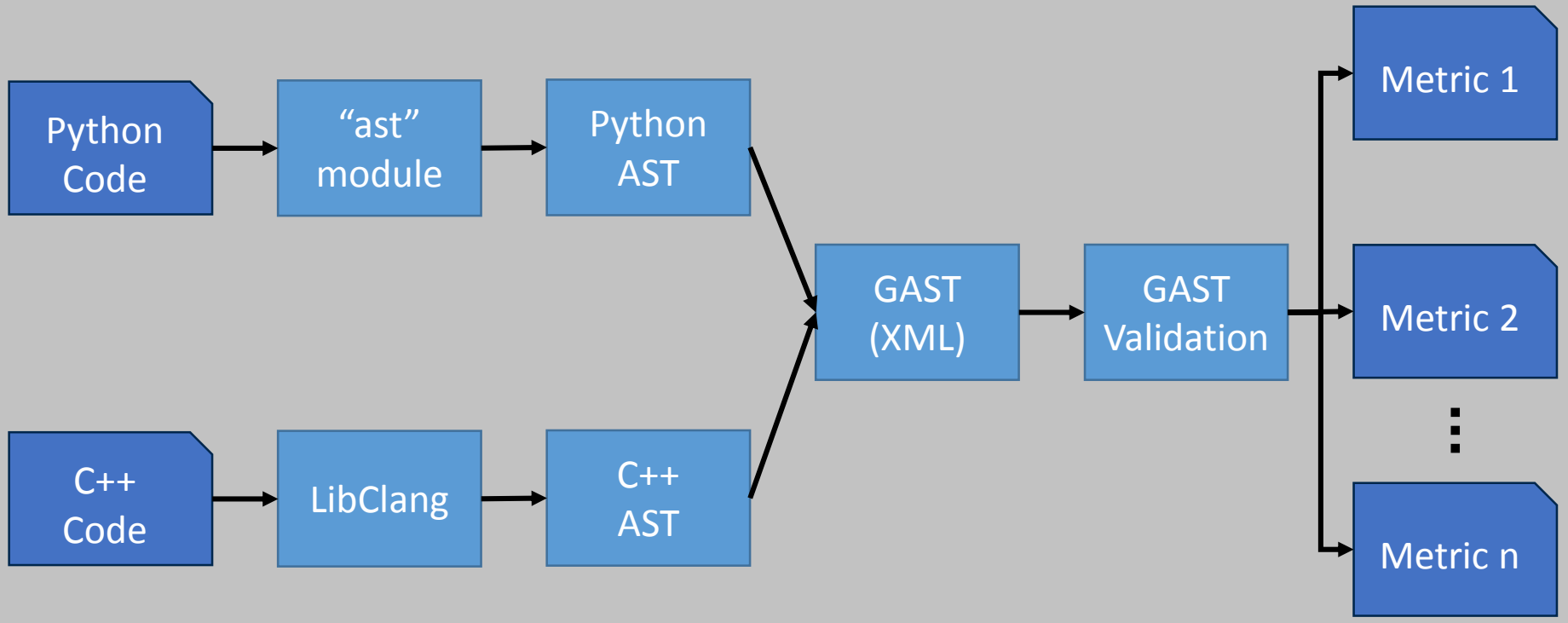
STEP 3: Validation

- BinaryOperator
  - BinaryOperatorMeta
- UnaryOperator
  - UnaryOperatorMeta
- FunctionDefinition
- ClassDefinition
- If, For, While, Switch, Return, Case, Break
- Num, Char, Str

STEP 4: Code Evaluator

- **Input:** Valid GAST
- **Output:** Metric scores
- Code quality using metrics on GAST
  - McCabe
  - Halstead

system overview



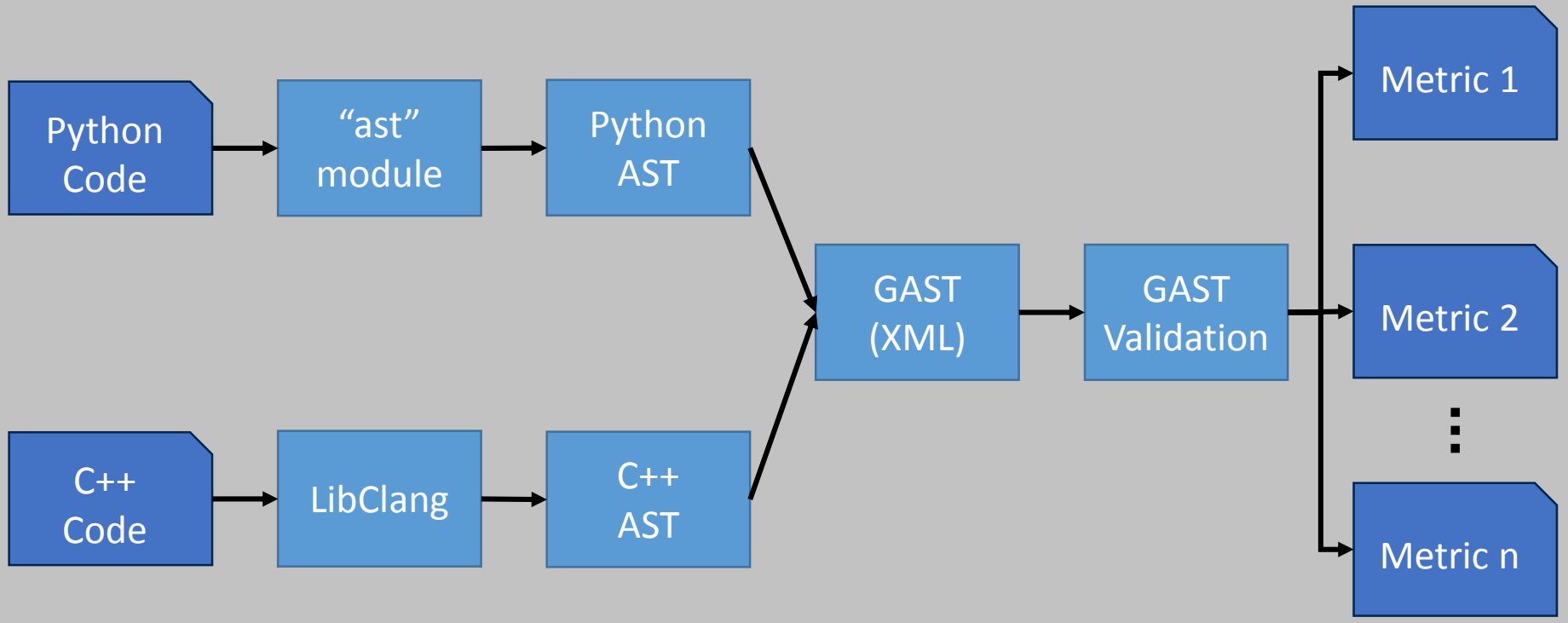
overview

- Software quality
- Software metrics
- GAST Framework
- **Results**

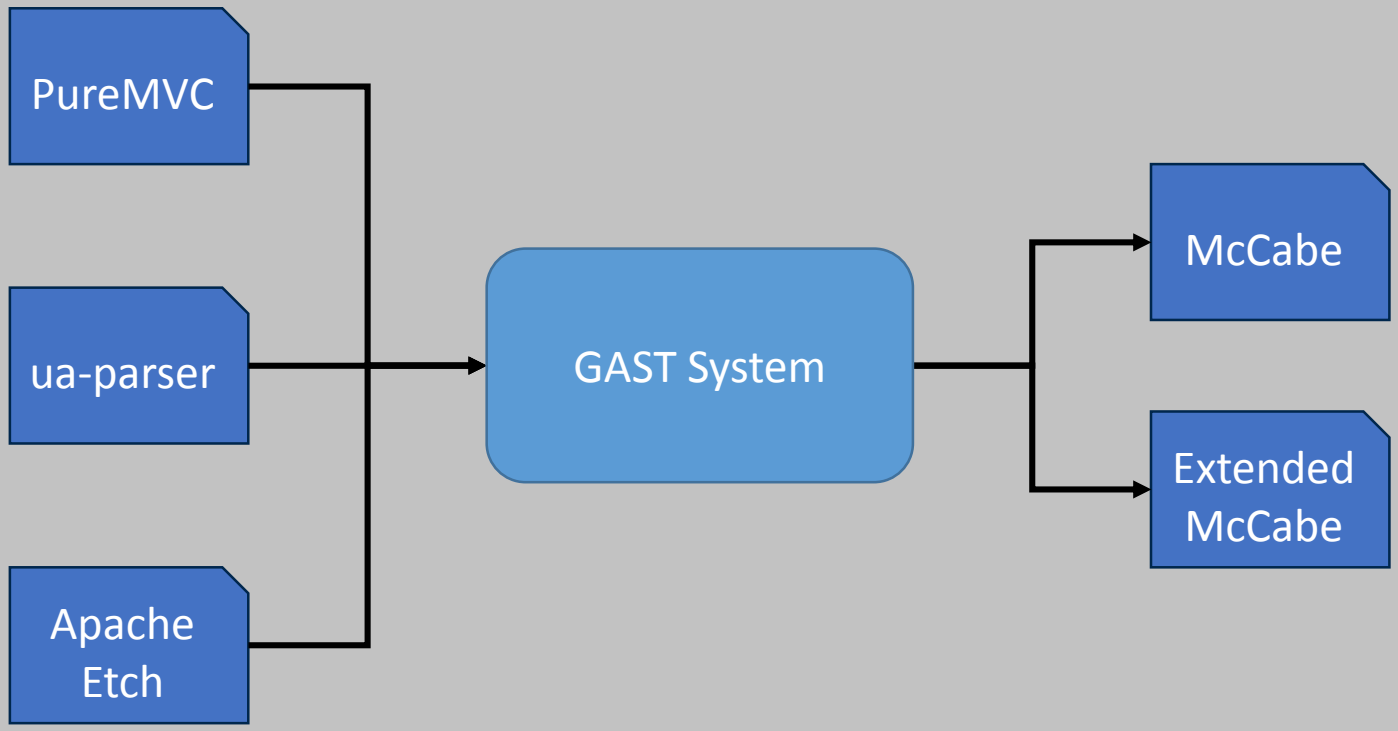
results

- 3 open source projects
  - PureMVC
  - ua-parser
  - Apache Etch
  - Each in C++/Python
- Calculate:
  - McCabe
  - Ext McCabe
- Verify with Tool

results


Python Code
"ast" module
Python AST
C++ Code
LibClang
C++ AST
GAST (XML)
GAST Validation
Metric 1
Metric 2
Metric n

results


PureMVC
ua-parser
Apache Etch
GAST System
McCabe
Extended McCabe

results

McCabe Metrics for PureMVC, ua-parser, and Apache Etch

Project	Language	SLOC	McCabe	Ext McCabe	Tool
ua-parser	C++	681	94	102	102
ua-parser	Python	542	91	106	106
PureMVC	C++	1991	154	162	162
PureMVC	Python	413	34	35	35
Apache Etch	C++	21125	1106	1302	1302
Apache Etch	Python	5023	377	422	422

results

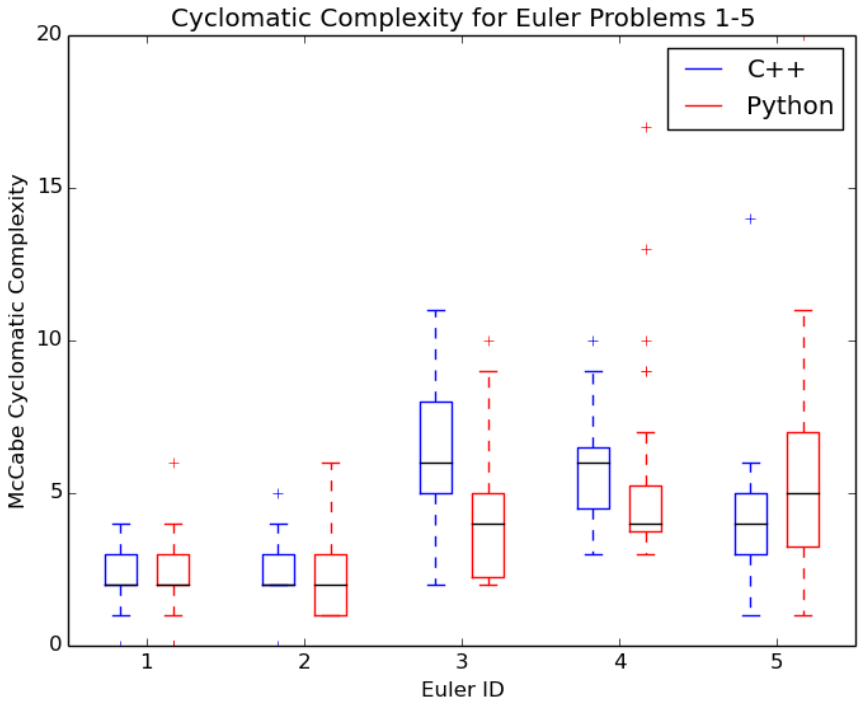
- Evaluate solutions from Project Euler
  - Problems 1-5
  - All solutions in C++/Python
- Calculate:
  - # solutions
  - Halstead difficulty
  - McCabe
  - Ext McCabe

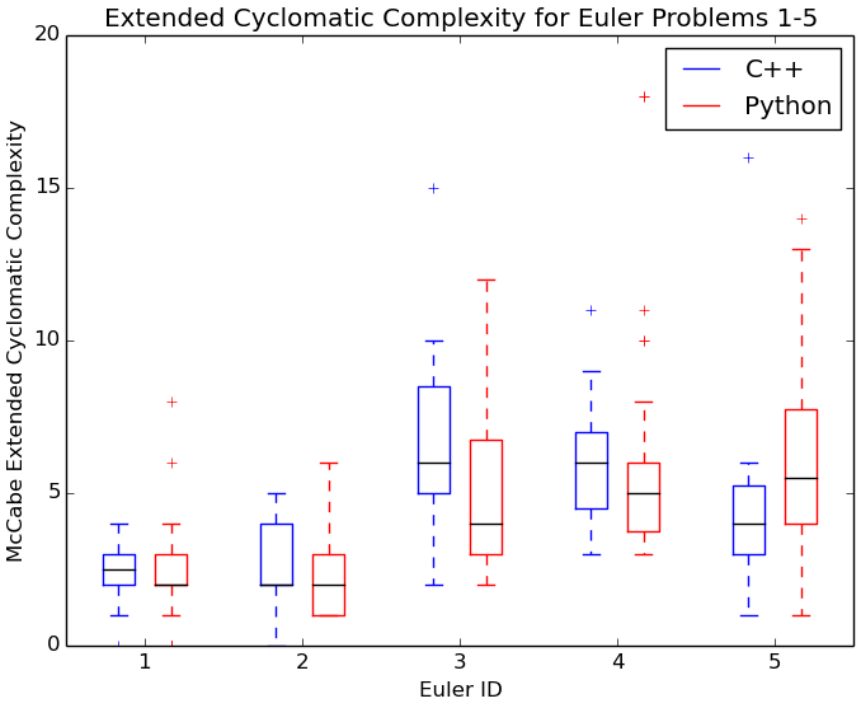
results

Descriptions of First Five Exercises for Project Euler

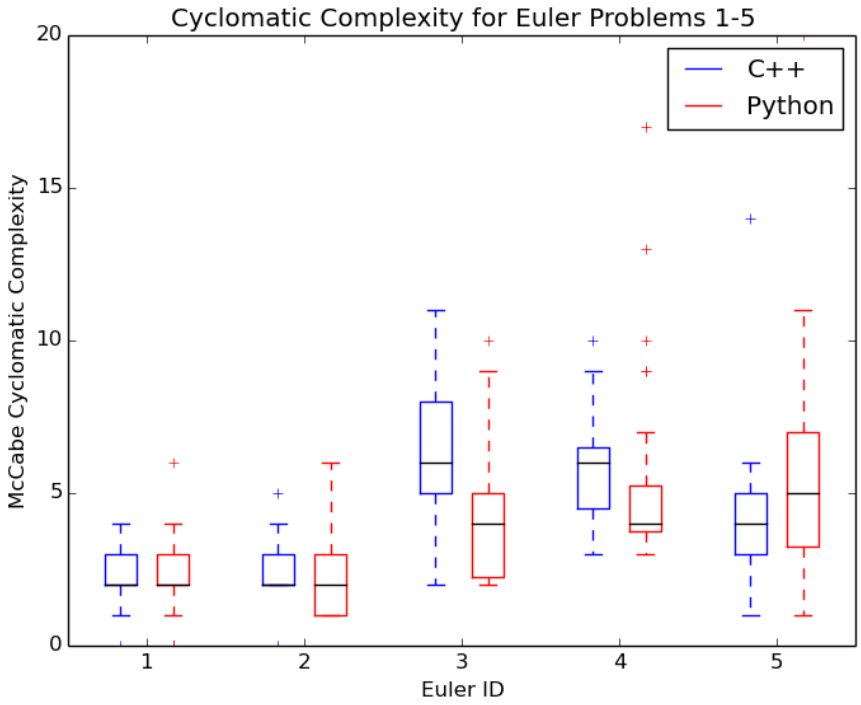
Tag	Full Description
1. Multiples of 3 and 5	If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 1000.
2. Even Fibonacci	Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.
3. Largest Prime Factor	The prime factors of 13195 are 5, 7, 13 and 29. What is the largest prime factor of the number 600851475143 ?
4. Largest Palindrome	A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009 = 91 × 99. Find the largest palindrome made from the product of two 3-digit numbers.
5. Smallest Multiple	2520 is the smallest number that can be divided by each of the numbers from 1 to 10 without any remainder. What is the smallest positive number that is evenly divisible by all of the numbers from 1 to 20?

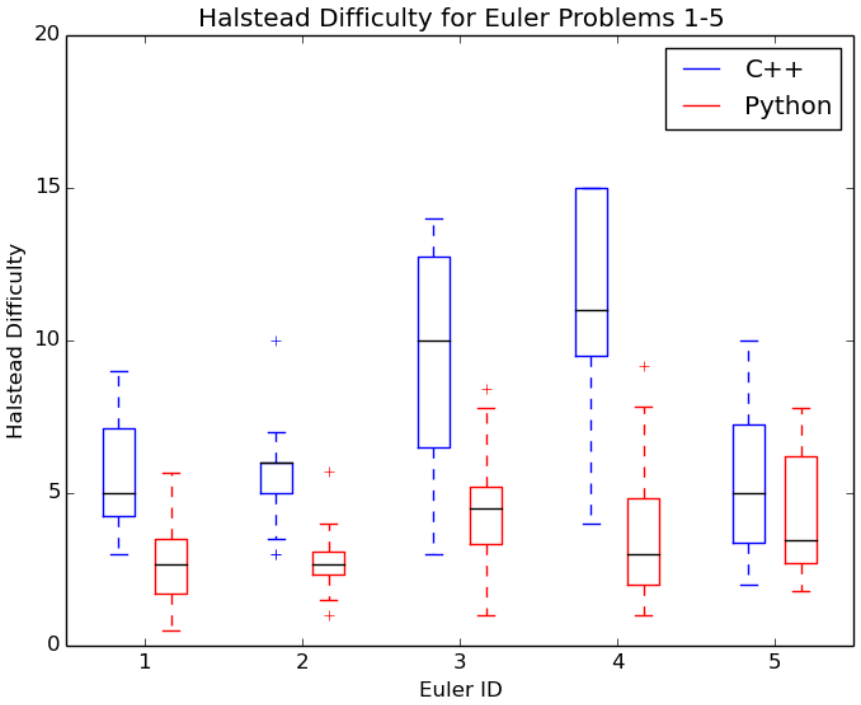
results


Cyclomatic Complexity for Euler Problems 1-5
McCabe Cyclomatic Complexity
Euler ID
C++
Python
0
5
10
15
20
1
2
3
4
5


Extended Cyclomatic Complexity for Euler Problems 1-5
McCabe Extended Cyclomatic Complexity
Euler ID
C++
Python
0
5
10
15
20
1
2
3
4
5

results


Cyclomatic Complexity for Euler Problems 1-5
McCabe Cyclomatic Complexity
Euler ID
C++
Python
0
5
10
15
20
1
2
3
4
5


Halstead Difficulty for Euler Problems 1-5
Halstead Difficulty
Euler ID
C++
Python
0
5
10
15
20
1
2
3
4
5

results

Tag	Solns	Halstead			McCabe			Ext McCabe		
		High	Low	Avg	High	Low	Avg	High	Low	Avg
1. Multiples of 3 and 5	14	9.00	3.00	5.57	4.00	0.00	2.21	4.00	0.00	2.36
2. Even Fibonacci	17	10.00	3.00	5.53	5.00	0.00	2.59	5.00	0.00	2.76
3. Largest Prime Factor	11	14.00	3.00	9.59	11.00	2.00	6.36	15.00	2.00	7.09
4. Largest Palindrome	19	30.00	4.00	13.03	10.00	3.00	5.68	11.00	3.00	5.89
5. Smallest Multiple	16	10.00	2.00	5.44	14.00	1.00	4.38	16.00	1.00	4.69

Table 4.3: Posted Solutions Written in C++

Tag	Solns	Halstead			McCabe			Ext McCabe		
		High	Low	Avg	High	Low	Avg	High	Low	Avg
1. Multiples of 3 and 5	33	5.67	0.50	2.65	6.00	0.00	2.00	8.00	0.00	2.12
2. Even Fibonacci	34	5.71	1.00	2.72	6.00	1.00	2.26	6.00	1.00	2.41
3. Largest Prime Factor	46	8.40	1.00	4.44	10.00	2.00	4.30	12.00	2.00	5.00
4. Largest Palindrome	40	9.15	1.00	3.49	17.00	3.00	5.23	18.00	3.00	5.65
5. Smallest Multiple	26	7.80	1.80	4.26	20.00	1.00	5.92	16.00	1.00	6.85

Table 4.4: Posted Solutions Written in Python

STEP 1

Choose Input Language

Python

Select An Exercise

Multiples of 3 and 5

Description

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

STEP 2

Enter Code Here

```
1  def multiples():
2      count = 0
3
4      for i in range(1000):
5          if (i%3 == 0 or i%5 == 0):
6              count += i
7
8      return count
9
10 print multiples()
```

Home

Run | Evaluate

```
$ python myprogram
$ 233168
$
```

CodeRx

Change Password | Logout

STEP 1

Choose Input Lang

Python

Select An Exercise

Multiples of 3 and 5

Description

If we list all the natural numbers
are multiples of 3 or 5, we get 3
sum of these multiples is 23.

Find the sum of all the multiple
1000.

Congratulations!

You did it! You got the correct answer.

Here's how you did:

```
Cyclomatic Complexity => 2
Halstead Difficulty => 0.5
```

Your Best Score:

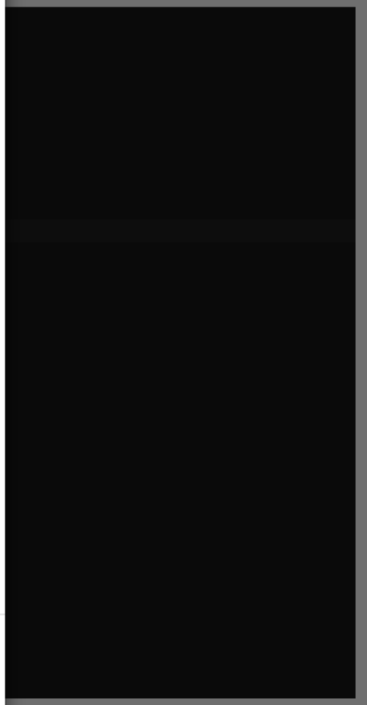
```
Cyclomatic Complexity => 2
Halstead Difficulty => 0.5
```

How you compare with others:

```
Average Cyclomatic Complexity => 2
Minimum Cyclomatic Complexity => 0
Maximum Cyclomatic Complexity => 6

Average Halstead => 2.65
Minimum Halstead => 0.5
Maximum Halstead => 5.67
```

OK



Home

Run | Evaluate

```
$ python myprogram
$ 233168
$
```

conclusion

- How to automate code quality evaluation?
- GAST framework:
  - Language neutral, extensible
  - Interoperable
  - Standard metric interpretations
  - Offload work to parser tools

Thank you!

```
PROCEDURE buildCpp(gast)
  OBTAIN index using clang.cindex.Index.create
  OBTAIN translation_unit using clang.parse

  CALL gast.ast_visit with translation_unit.cursor

PROCEDURE buildPython(gast)
  OBTAIN translation_unit using parse(inFileName)

  CALL gast.ast_visit with translation_unit

PROCEDURE main()
  READ inFile
  GET fileType

  SET root to "gast"
  SET root.attribute.filename to inFileName

  IF fileType = C++ THEN
    INIT gast as Cpp_Ast
    CALL buildCpp with gast
  ELSE IF fileType = Python THEN
    INIT gast as Python_Ast
    CALL buildPython with gast
  ENDIF

  WRITE root to outFile
```

```
CLASS gast
  @abstractmethod
  PROCEDURE str_node(node, tag)
    // 1. process AST node, storing node type info
    // as tag names in the XML file
    // 2. store auxillary information in AST for computing
    // metrics such as binary and unary operator meta
    // information (operator and operands)
    pass

  @abstractmethod
  PROCEDURE ast_visit(node, pnode=None, level=0)
    // 1. recursively traverse AST nodes, keeping
    // track of depth in tree
    pass

  @abstractmethod
  PROCEDURE translate(node, name)
    // 1. translate AST node labels according to
    // GAST definition
    pass

  @abstractmethod
  PROCEDURE getRoot()
    // return root tag
    pass
```