5. Software Measurement
5.a Software Engineering - Metrics
5.a.1 Software Metrics

• “When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is meager and unsatisfactory”¹

• Software Complexity Metrics
  - Computational complexity (efficiency in use of machine resources)
  - Psychological complexity (factors that affect programmer performance in creating, comprehending and modifying software.)
  - Psychological complexity metric types include problem complexity, design complexity, program complexity.

---

¹ Lord Kelvin, Popular Lectures and Addresses, 1889.
Software Metrics

- **Process Metrics** - resource consumption for creation or maintenance; levels of personnel experience, models used (waterfall, for example), development time consumed,

- **Product Metrics** - measures of code size, code logic or data structure.
5.a.2  Size Metrics - Lines of Code

- Code size measures are easy to compute and are useful in considering productivity.
- No general agreement on how to count lines of code.
- Should lines of code mean source statements or lines in file?
- Should comments or blank lines be included?
  - Their presence does not affect the functionality.
  - Is there a cost associated with their inclusion?
- Should executable statements only be included?
  - Declarations support the compiler and not the functionality,
  - Interested only in size of program that supports a certain function?
- What happens when 2 or more statements occur on the same line?
- What happens when 1 statement is spread over multiple lines?
One Definition of SLOC

- A line of code is 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 specifically includes all lines containing program headers, declarations, executable and non-executable statements.
5.a.2.a Example Fortran

1. Subroutine Sort (X,N)
2. integer X(100), N, I, J, Save, Im1
3. c This routine sorts array X into ascending order.
4. c The array is an integer array of 100 elements.
5. if (N .LT. 2) go to 200
6. do 210 I=2,N
7. Im1=I-1
8. do 220 J=1, Im1
9. if (X(I).GE.X(J)) go to 220
10. Save=X(I)
11. X(I)=X(J)
12. X(J)=Save
13. 220 continue
14. 210 continue
15. 200 return
16. end

2. Derived from Figure 2.2 in Conte’s book Software Engineering Metrics and Models
Example Lines of Code

• How many lines of code are in the Fortran example?
• If we are only counting executable lines of code then 11,
• If we ignore comments then 14,
• If we are counting all Fortran Statements only then 14 (exclude comments)
• If we are counting lines in the file (compiler listing line numbers usually) then 16.
5.a.3 Token Counts

- Counting lines of code is not a consistent because some lines are more complex than others. One way to improve is to use metrics based on the number of Tokens.

- What is a Token? A Token is the basic syntactic unit that makes up a program. Tokens are the programming language equivalent of natural language words.

- A token is a value of a lexical category such as integer constant (254, 30), operator (+, -, *, /, >, =, :=), reserved word (if, for, class, then, else, while), identifier (myType, myInteger, getSymbol).

- Examples:
  myInt = myInt + 25 ; has 6 tokens
  public class Cal extends Object implement Calendar { ... (has 8 tokens)

- Halstead devised a family of metrics (called Software Science) based on token counts.
5.a.4 Halstead Measures Based on Token Counts

• Basic metrics:
  - $\eta_1 =$ number of unique operators,
  - $\eta_2 =$ number of unique operands,
  - $N_1 =$ total occurrences of operators,
  - $N_2 =$ total occurrences of operands.

• Operators - any token in a program that specifies an action is considered an operator.
  - Most delimiters are also categorized as operators.
  - Examples: +, -, *, /, ;, =, ==, (), {}, if, while, for, then, else

• Operands - any token that is not an operator is considered an operand.
  - Operands include: objects, class names, method names, type names, and constants.

Halstead Measures

- Halstead’s rules excluded counting tokens in declaration statements, comments and tokens in input and output statements.
- Using these conventions, the Halstead measures for the Fortran program example above is:
  - $\eta_1 = 14 = \text{number of unique operators},$
  - $\eta_2 = 13 = \text{number of unique operands},$
  - $N_1 = 51 = \text{total occurrences of operators},$
  - $N_2 = 42 = \text{total occurrences of operands}.$
5.a.4.a Using the Basic Metrics: Size

- **Program Size** (or length)
  The size of a program in terms of total number of tokens used is:
  
  \[ N = N_1 + N_2 \]

  - The size of the example Fortran program is: \( N = 93 \)

- The size can be converted to an estimate of the number of lines of code by
  the relationship:
  
  \[ SLOC = \frac{N}{c} \]

  where \( c \) is language dependent (Fortran value is 7)

- So, the Halstead based estimate of **SLOC** for SORT is 93/7, about 13
5.a.4.b Using the Basic Metrics: Vocabulary

- Vocabulary emphasizes the fact that programmers have a finite subset of all possible operators and operands that are used to construct a program.
- Program vocabulary is defined as:
  \[ \eta = \eta_1 + \eta_2 \]
- For the Fortran program, vocabulary is \( \eta = 27 \)
5.a.4.c Using the Basic Metrics: Volume

- Halstead also provided a measure of the volume of a program. This measure is reflected by the number of bits that would be necessary to store the program given a uniform binary encoding for vocabulary is used.
- For example, if we have 8 operation codes then it will take 3 bits to store them for purposes of commands. Each operation code (0 to 7 or binary 000 to 111) can be expressed uniquely in the 3 bits.
- The number of bits required is determined by the relation:
  \[ \text{Bits} = \log_2 8 = 3 \]
- So, the volume of a program is the size of the program times the number of bits required to encode its operators and operands(vocabulary):
- Volume of a program:
  \[ V = N \times \log_2 \eta \]
- For the Fortran example: \( V = 93 \times \log_2 27 = 93 \times 5 = 465 \)
5.a.4.d Halstead Potential Volume

- Many implementations can be created for an algorithm. This leads to many different but functionally equivalent programs.
- Among these, Halstead defined the one having minimal size to have the **potential volume** $V^*$.  
  \[ V^* = (2 + \eta_2^*) \cdot \log_2(2 + \eta_2^*) \]

where:

  - $2$ is the number of unique operators for a procedure call (procedure name and () )
  - $\eta_2^*$ represents the number of conceptually unique input and output parameters (operands).

- $V^*$ represents a program that performs the entire function by making a procedure call with appropriate input and output parameters.
- With the Fortran sort program: X is an array input and output; and N is an input of the size of the array (3 inputs and outputs).
  \[ V^* = (2 + 3) \cdot \log_2(2 + 3) = 11.6 \]
5.a.4.e Halstead Level

- The level $L$ of a program reflects the level of abstraction in which the program is expressed. The maximum value for Level is 1, which means the program is written with the minimum size.

$$L = \frac{V^*}{V}$$

- The abstraction level of the implementation of an algorithm is the ratio of the potential volume and the actual volume of the program.

- Using the Fortran Sort routine as an example,
  $$L = \frac{V^*}{V} = \frac{11.6}{465} = 0.02$$
5.a.4.f Halstead Effort

- The effort to create a program increases with the size of the program.
- Halstead defined a measure of the effort (E) required to implement a program. Halstead hypothesized that E represents the number of elementary mental discriminations necessary to create the program.
- **Effort E** is the Volume divided by an estimate of the level of the program.
- Halstead’s method for estimating the level is: \( \frac{2 \eta_2}{\eta_1 N^2} \)
- This leads to the following equation for effort E:
  \[
  E = \frac{\eta_1 N^2 N \log_2 \eta}{2 \eta_2}
  \]
- The Fortran Sort example:
  \[
  E = \frac{14 \times 42 \times 465}{2 \times 13} = 10516
  \]
5.a.4.g Halstead Time

- Psychologist John Stroud suggested that the human mind is capable of making a limited number of elementary discriminations per second - ranging between 5 and 20.

- Halstead experimented with his effort estimate and what he called the Stroud number to arrive at the following time estimate $T$:

  $$ T = \frac{E}{\beta} $$

  Halstead found the $\beta = 18$ worked best in estimating time.

- With the Fortran example:

  $$ T = \frac{10516}{18} = 584 \text{ seconds} = \text{about 10 minutes} $$

- Halstead would estimate about 10 minutes to write the Fortran Sort routine
Halstead’s Software Science - Reflections

• Several have questioned various aspects of Halstead’s software science (theory). Various studies have shown Halstead’s metrics to be inaccurate or incomplete.

• However, Halstead’s work was the first and most comprehensive theory devised to describe the software development process.

• **Halstead’s Software Science** made many aware (painfully) of the need and potential uses for metrics and estimators.

• Halstead’s basic metrics of size length and volume relate very well to Source Lines of Code.

• These basic measures have also been shown to relate well to error rates.
Function Count

- For large software products, a measure that is coarser than SLOC is useful.
- Methods, modules, classes, and functions have all been used as the basis for these coarser measures.
- Since criteria for decomposing a problem into modules vary widely, functions or methods have been a more successful basis for measure.
- A function in a program is a collection of executable statements and (or) declarations, parameters, along with local name definitions (variables, types, constants, exceptions). A function is an abstraction of part of the tasks that a program is to perform.
- The cognitive and syntactic overhead involved in constructing functions discourages functions that are either too small or too large.
- Some studies have shown that programmers use a similar number of functions to solve a given problem.
5.a.5  Data Structure Metrics

- Most software systems function as data processors: For example
- Application: DeskTop Publisher
  - **Input Data**: text string and formatting inputs
  - **Internal Data**: internal representation of text and formatting (paragraph formats, character formats, dictionary for spell checking)
  - **Output Data**: formatted document - hard-copy or electronic
Data Structure Metrics

- **Application:** SpreadSheet
  - **Input Data:** item names, item amounts, relationships among items,
  - **Internal Data:** cell computations and intermediate results of computations
  - **Output Data:** formatted spreadsheet of items and computation results

- Measuring the amount of data processed by software is important to understanding the psychological complexity of software.
5.a.5.a Amount of Data

- A compiler’s cross-reference listing can be used to determine the amount of data manipulated by a program.

- **VARS** is the count of all user-defined identifiers which are used to represent some value during compilation or execution of the program. VARS can be obtained from the cross-reference listing:

  Cross Reference of Fortran Program Sort

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Referenced in Program Line Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>1 2 9 10 11 12</td>
</tr>
<tr>
<td>N</td>
<td>1 2 5 6</td>
</tr>
<tr>
<td>J</td>
<td>2 8 9 11 12</td>
</tr>
<tr>
<td>SAVE</td>
<td>2 10 12</td>
</tr>
<tr>
<td>IM1</td>
<td>2 7 8</td>
</tr>
<tr>
<td>I</td>
<td>2 6 7 9 10 11</td>
</tr>
</tbody>
</table>

- For Sort, VARS = 6, which is the number of rows in the table above. Halstead’s \( \eta_2 = VARS + \text{unique constants + labels} \).

- **VARS, \( \eta_2 \) and N2** are the most popular data structure measures.
Measuring Data Usage

- **Live Variables**: One view is that complexity increases with the number of data items a programmer must keep track of at any point in program construction.

- For each statement in a program, we can consider the set of variables that are live during the execution of the statement.

- The set of Live Variables for a particular statement is not limited to the set of variables referenced by the statement.

- A variable is live over the region of program ranging from its first reference to its last reference, or from its declaration occurrence until its no longer accessible.

- Define the average number of live variables (LV) to be the sum of the count of live variables divided by the count of executable statements in a method or function.
Measuring Data Usage - Example Sort

• What are the live variables for Sort?

<table>
<thead>
<tr>
<th>Line</th>
<th>Live Variables</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>N, I</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>I, IM1</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>I, IM1, J</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>I, J, X</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>I, J, X, SAVE</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>I, J, X, SAVE</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>J, X, SAVE</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

• The average number of live variables is:

\[ LV = \frac{22}{11} = 2 \]
5.a.5.b Variable Spans

- **Span (SP)** tries to capture a notion of how often a variable is used in a program. The term span refers to a region of text within a program.

- If a program references a variable in $n$ statements, then there are $n-1$ spans for that variable.

- Each span has a size, which is a count of the number of statements from one reference of the variable to the next.

- The larger, and more spans, the more the programmer must remember when constructing the statements of a program.

- In Sort, the variable $I$ is referenced 6 times. $I$ has 5 spans in Sort, which are of length: 4, 1, 2, 1, and 1.
Modules Sharing Data

- The relationship between Sort and any other program that uses Sort is relatively straightforward. Two variables are passed (the array is modified and is returned as output). There are no non-local variable references.
  - In an Object-Oriented programming language, instance variables and class variables may be referenced internally or if they are protected, in extending classes. These would be considered as non-local variable references.

- **Segment-global usage** pairs are defined to show the basis of data communication.

- The **segment-global pair** \((P, R)\) indicates that the code segment \(P\) either reads or writes non-local data \(R\).

- A **Binding** is represented by a triple \((P,R,Q)\) which indicates that segment \(P\) sets global variable \(R\) whose value is subsequently read by segment \(Q\).
**Fan-In**

- Henry and Kafura extended the notion of *bindings* \((P, R, Q)\) so that \(R\) can be either non-local or parameter data.

- **Fan-in** of a module \(Q\) is the number of unique modules \(P\) so that one of the following hold:
  - There exists a variable \(R\) so that the triple \((P, R, Q)\) is a valid data binding triple,
  - There exists a module \(T\) and variables \(R, S\) so that the triples \((P, R, T)\) and \((T, S, Q)\) are valid data binding triples.

- The **fan-in** for a module is the number of modules that pass data to the module, either directly or indirectly.

- We can similarly define **fan-out** for a module as the number of modules to which data is passed either directly or indirectly.
5.a.6  Program Structure Metrics

- **Decision Count (DE)** - decision count is the number of if, for, while, case, and any other conditional or looping control structures in the program.

- One extension to Decision Count is to count each conjunct (and &&) or disjunct (or | |) in decision predicates as additional decisions.

- A better know metric based on the number of decisions in the flowgraph for the program is the **Cyclomatic Complexity Number**:

  \[ v(G) = e - n + 2*p \]

  where:
  - \( e \) is the number of edges,
  - \( n \) is the number of nodes, and
  - \( p \) is the number of connected components in the flowgraph.

- It turns out that the cyclomatic complexity number \( v(G) \) is equivalent to the number of predicates in the program + 1

  \[ v(G) = DE + 1 \]

- Computationally, its easier to calculate \( DE \) than \( v(G) \).
5.a.6.a Paths

- The minimum number of paths in a program’s flowgraph is useful information for testing, and is a common test selection criterion.
- A path can be defined as a unique sequence of statements from the first executable statement to a terminal statement (return or end).
- The paths through the Fortran Sort program (in statement numbers) are:
  - 1, 5, 15
  - 1, 5, 6, 7, 8, 9, 13, 14, 15
  - 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
- Note that paths treat iterations as branches: the loop body is either executed, or it is not. In the case of Sort, there are two unreachable paths (0 executions of each of the loop statements). The guard on N in statement 5 precludes these two paths.
Code Metric Exercise

- Using a component decomposition of a source-code project, identify some number of modules (more than 1 module, function, method, or class) of code that represents together a total of 75 to 150 source lines of code.
Code Metric Exercise

• For the modules selected, generate (in tabular form) the following metrics:
  - SLOC\(^4\)
  - Halstead Size
  - Halstead Vocabulary
  - Halstead Volume
  - VARS
  - Live Variables
  - SPAN (pick 5 variables that adequately demonstrate the concept)
  - Fan-In, Fan-out - pick 1 module that adequately demonstrates the concept
  - DE for 1 module with decision constructs that demonstrates the concept
  - Paths - for an example module that demonstrates the importance of paths to testing.

\(^4\) Base your calculation (and definition) of SLOC on the definition given in the course notes as one sample definition of SLOC.