**Software Engineering:**

**Exploring Software Metrics with Python**

Samantha M. Hipple

School of Technology and Computing, City University of Seattle

CS 504: Software Engineering

Kendra Schraml

June 5, 2022

**Software Engineering:**

**Exploring Software Metrics with Python**

Basic info about software metrics – brief history, general types, importance.

This report explores software metric testing using three Python modules **pylama**, **pylint**, and **radon**. Pylama is a code audit tool that wraps nine software metric tools in one: **pycodestyle**, **pydocstyle**, **pyflakes**, **mccabe**, **pylint**, **radon**, **eradicate**, **mypy**, and **vulture**. We began our metrics testing by running all of the metrics tools available in Pylama and saving the results to their respective report files.

The **vulture** module is designed to find unused and untested code, while the **eradicate** module is designed to remove all dead code (commented-out code), in the files they parse. The **mypy** module is a static type-checker that requires your code to be written using static-typing, which does not exist at all in our example program. These three modules returned no results when ran against our Tic Tac Toe source code via Pylama.

Next, we used the actual **radon** module to provide a more in-depth look at the our source code’s raw metrics, cyclomatic complexity (CC), Halstead metrics, and maintainability index (MI).

Text

Description automatically generated

Text

Description automatically generated Text

Description automatically generated

Lastly

Something about the three modules that make up our program main.py, ttt\_menus.py and tic\_tac\_toe.py.

As a way of demonstrating the importance of metric testing, we will be using the metrics we discover to improve our code as we progress through the report.

Software metrics are used to measure software characteristics that are quantifiable. [source](https://stackify.com/track-software-metrics/)

There are four functions related to software metrics: (1) planning, (2) organizing, (3) controlling, and (4) improving. [source](https://www.geeksforgeeks.org/software-measurement-and-metrics/)

Software metrics are used to measure performance, plan working items, measure productivity, estimating costs, and more. The goal of tracking these metrics is to determine the quality of the software product, then apply what is learned to improve the system and predict its end-state quality. Software development managers often use software metrics to identify areas of improvement, manage workloads, and reduce costs while development team members can use them to better communicate that status of their projects, to find and address issues, and better manage their own workflow. [source](https://stackify.com/track-software-metrics/)

The first attempts at applying this idea of quantifying software characteristics to track software quality and development progress appeared in the 1970s. These initial metrics focused on coding design and attribute more so than a program’s overall architecture. TEXTBOOK

PyFlakes

**Radon**

The first software metrics module we are going to explore is called **radon**. Radon provides analysis of four different metric types: Halstead metrics, Cyclomatic Complexity (CC), raw metrics, and the maintainability index ([source](https://radon.readthedocs.io/en/latest/index.html)).

This section will break down each of these metrics with a brief overview that includes the analysis results from our source code and an explanation of how it can be used to improve our program, and maybe the results after improvements.

**Halstead Metrics**

The Halstead Complexity metric, for example, uses four metric units when analyzing a program’s source code: (1) number of *distinct* operators, (2) number of *distinct* operands, (3) sum of all *operators*, (4) and the sum of all *operands*. Halstead then applied these values to define two more measurements: (1) unique program vocabulary (sum of unique operands and operators), and (2) program length (sum of all operands and operators). From there, Halstead also uses the values to define four more metrics: (1) volume, (2) potential volume, (3) program implementation level, and (4) effort. A solid start to the field of metrics, Halstead leaves a lot to be desired in the analysis of program complexity as its only true measurement is that of the program’s source code lexical complexity, not the structure, logic, or design (TEXTBOOK).

**Raw Metrics**

The following are the definitions employed by Radon:

* **LOC**: The total number of lines of code. It does not necessarily correspond to the number of lines in the file.
* **LLOC**: The number of logical lines of code. Every logical line of code contains exactly one statement.
* **SLOC**: The number of source lines of code - not necessarily corresponding to the **LLOC**.
* Comments: The number of comment lines. Multi-line strings are not counted as comment since, to the Python interpreter, they are just strings.
* Multi: The number of lines which represent multi-line strings.
* Blanks: The number of blank lines (or whitespace-only ones).

The equation SLOC + Multi + Single comments + Blank = LOC should always hold. Additionally, comment stats are calculated:

* C % L: the ratio between number of comment lines and **LOC**, expressed as a percentage;
* C % S: the ratio between number of comment lines and **SLOC**, expressed as a percentage;
* C + M % L: the ratio between number of comment and multiline strings lines and **LOC**, expressed as a percentage.

Text

Description automatically generated

Text

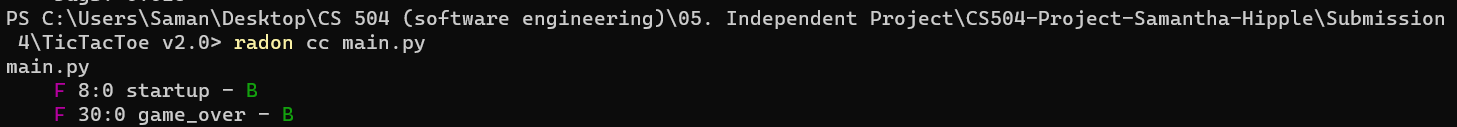
Description automatically generated

Text

Description automatically generated

**McCabe’s Cyclomatic Complexity**

McCabe directly related the complexity of a program’s control flow to its overall quality. That is, the number of branches in the program’s design or source code and is based on graph theory.



Text

Description automatically generated

Text

Description automatically generated

**Raw Metrics**

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

**Maintainability Index**

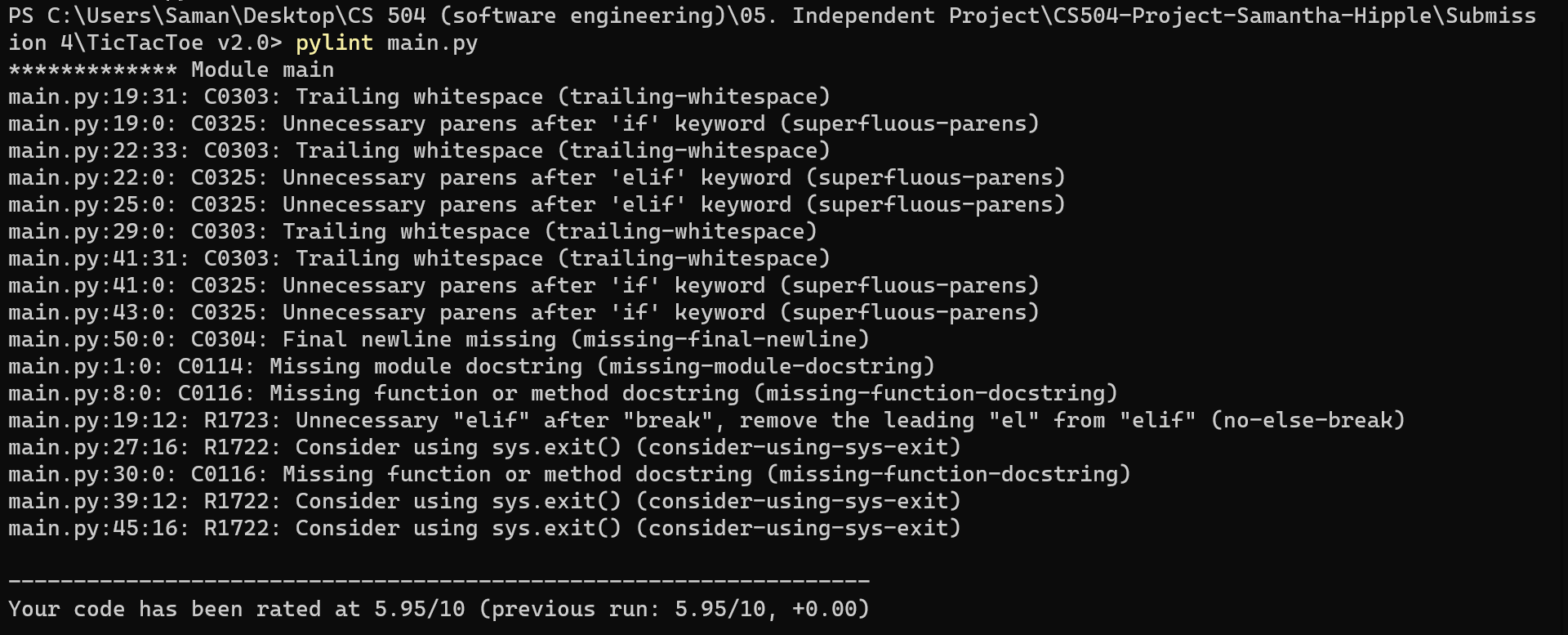
**Text

Description automatically generated**

**Pylint**

The second metrics module for Python we delved into is called **pylint**.

“Most of the messages here will be self-explanatory, but the first letter in each line will map to Convention, Refactor, Warning, or Error — and with regards to coding style, Pylint follows the [PEP8 style guide](http://www.python.org/dev/peps/pep-0008/).” [source](https://blog.codacy.com/which-python-static-analysis-tools-should-i-use/)



Graphical user interface, text

Description automatically generated

**Appendix A**

Initial Radon McCabe Cyclomatic Complexity Results

**Figure A1.** Initial CC results for *main.py*

Text

Description automatically generated**Figure A2.** Initial CC results for *ttt\_menus.py*

Text

Description automatically generated**Figure A3.** Initial CC results for *tic\_tac\_toe.py*

Text

Description automatically generated**Figure A4.** Initial CC results for *test.py*

**Appendix B**

Text, letter

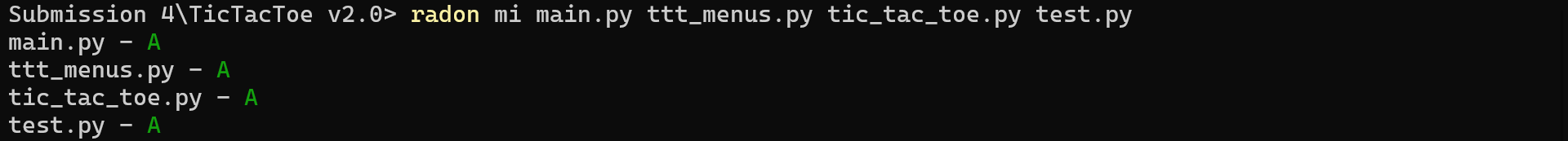
Description automatically generated**Appendix B.** Initial Pylama McCabe CC Report

A screenshot of a computer

Description automatically generated with low confidence**Figure 1.** Initial Radon Raw Metrics

**Text

Description automatically generatedFigure 2.** Initial Radon Halstead Metrics

**Figure 3.** Maintainability Index

**Pylint**

**A screenshot of a computer

Description automatically generated with low confidenceFigure #.** Initial Pylint report for main

**Text

Description automatically generatedFigure #.** Initial Pylint results for main

**Text, letter

Description automatically generatedFigure #.** Pylama’s initial Pylint results for main

**Text

Description automatically generatedFigure #.** Initial Pylint results for ttt\_menus

Text

Description automatically generated with low confidence**Figure #.** Pylama’s initial Pylint results for ttt\_menus

**Graphical user interface

Description automatically generated with medium confidenceFigure #.** Initial Pylint report for ttt\_menus

A screenshot of a computer

Description automatically generated with low confidence**Figure #.** Initial Pylint report for tic\_tac\_toe

**Appendix A**

Initial Pylint results for tic\_tac\_toe (sorted)

tic\_tac\_toe.py:163:4: C0103: Method name "game\_mode\_2\_PvE" doesn't conform to snake\_case naming style (invalid-name)

tic\_tac\_toe.py:243:4: C0103: Method name "game\_mode\_1\_PvP" doesn't conform to snake\_case naming style (invalid-name)

tic\_tac\_toe.py:1:0: C0114: Missing module docstring (missing-module-docstring)

tic\_tac\_toe.py:15:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:21:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:33:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:37:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:47:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:52:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:56:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:62:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:69:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:79:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:83:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:101:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:112:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:120:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:149:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:152:4: C0116: Missing function or method docstring (missing-function-docstring)

tic\_tac\_toe.py:12:0: C0301: Line too long (113/100) (line-too-long)

tic\_tac\_toe.py:57:0: C0301: Line too long (101/100) (line-too-long)

tic\_tac\_toe.py:73:0: C0301: Line too long (102/100) (line-too-long)

tic\_tac\_toe.py:75:0: C0301: Line too long (102/100) (line-too-long)

tic\_tac\_toe.py:115:0: C0301: Line too long (111/100) (line-too-long)

tic\_tac\_toe.py:30:54: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:41:33: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:43:21: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:48:95: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:49:78: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:72:65: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:85:82: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:89:65: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:96:85: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:107:82: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:113:59: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:114:65: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:119:40: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:150:44: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:151:54: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:160:25: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:168:57: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:171:31: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:173:52: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:180:64: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:196:23: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:206:63: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:211:21: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:217:65: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:220:25: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:226:61: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:228:25: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:230:36: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:241:0: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:248:57: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:251:31: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:253:52: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:260:64: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:273:48: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:274:63: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:277:36: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:280:23: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:285:65: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:288:25: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:294:61: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:296:25: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:298:36: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:304:30: C0303: Trailing whitespace (trailing-whitespace)

tic\_tac\_toe.py:115:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)

tic\_tac\_toe.py:127:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)

tic\_tac\_toe.py:128:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)

tic\_tac\_toe.py:134:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)

tic\_tac\_toe.py:135:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)

tic\_tac\_toe.py:140:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)

tic\_tac\_toe.py:141:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)

tic\_tac\_toe.py:146:0: C0325: Unnecessary parens after 'if' keyword (superfluous-parens)

tic\_tac\_toe.py:147:0: C0325: Unnecessary parens after 'return' keyword (superfluous-parens)

tic\_tac\_toe.py:2:0: C0411: standard import "import random" should be placed before "import numpy as np" (wrong-import-order)

tic\_tac\_toe.py:3:0: C0411: standard import "from time import sleep" should be placed before "import numpy as np" (wrong-import-order)

tic\_tac\_toe.py:120:4: R0912: Too many branches (14/12) (too-many-branches)

tic\_tac\_toe.py:41:8: R1705: Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (no-else-return)

tic\_tac\_toe.py:155:8: R1705: Unnecessary "elif" after "return", remove the leading "el" from "elif" (no-else-return)

tic\_tac\_toe.py:120:4: R1710: Either all return statements in a function should return an expression, or none of them should. (inconsistent-return-statements)

tic\_tac\_toe.py:152:4: R1710: Either all return statements in a function should return an expression, or none of them should. (inconsistent-return-statements)

tic\_tac\_toe.py:39:15: R1714: Consider merging these comparisons with "in" to "marker not in ('X', 'O')" (consider-using-in)

tic\_tac\_toe.py:90:16: R1722: Consider using sys.exit() (consider-using-sys-exit)

tic\_tac\_toe.py:91:12: W0702: No exception type(s) specified (bare-except)

tic\_tac\_toe.py:169:29: W1309: Using an f-string that does not have any interpolated variables (f-string-without-interpolation)

tic\_tac\_toe.py:249:29: W1309: Using an f-string that does not have any interpolated variables (f-string-without-interpolation)

tic\_tac\_toe.py:95:21: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)

tic\_tac\_toe.py:107:13: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)

tic\_tac\_toe.py:168:17: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)

tic\_tac\_toe.py:217:25: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)

tic\_tac\_toe.py:225:25: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)

tic\_tac\_toe.py:248:17: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)

tic\_tac\_toe.py:285:25: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)

tic\_tac\_toe.py:293:25: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)