投票 最多可选1项

设置

Are you ready ?

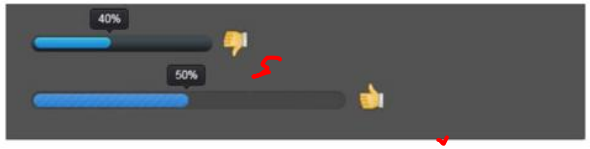
A Yes

B No

提交

Review - Interface design

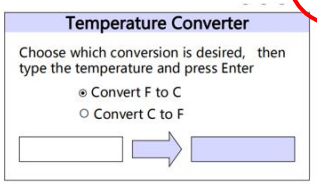
- Interface design golden rules
  1. Place the user in control
  2. Reduce the user's memory load
  3. Make the interface consistent



- Interface design principles
  - visibility / consistent/ mapping/ feedback

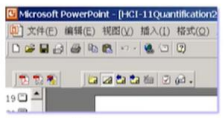
- Interface analysis and design step
  - People / Task / Enviroment / Content
  - interface objects and actions / event / state

- Design evaluation: KLM, Fitts



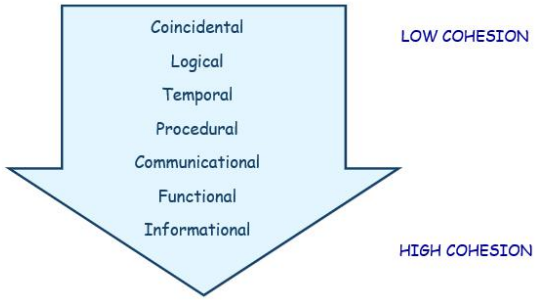
$$T = a + b\log_2(D/S+1)$$

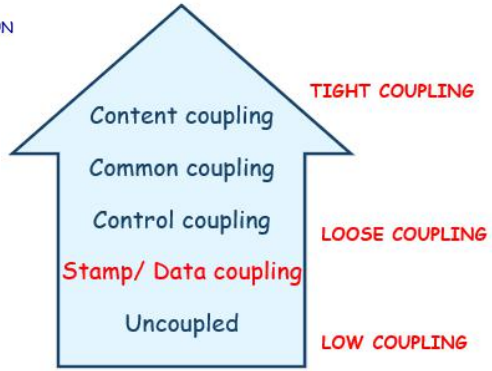
D: moving distance
S: the size of target



Review - Component-Level design

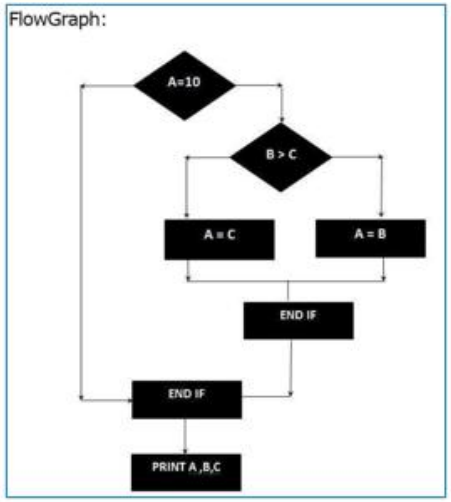
- High Cohesion & Low Coupling
  - qualitative analysis
  - quantitative analysis





- Complexity (McCabe)
  - quantitative analysis

$$
\begin{aligned}
V(G) &= E - N + 2 \\
&= 8 - 7 + 2*1 \\
&= 3
\end{aligned}
$$



Software Engineering

Part 2
Modeling

Chapter 15+
Writing the Programs

These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill 2014). Slides copyright 2014 by Roger Pressman.

Reproduced by Ning Li , 2022

Contents

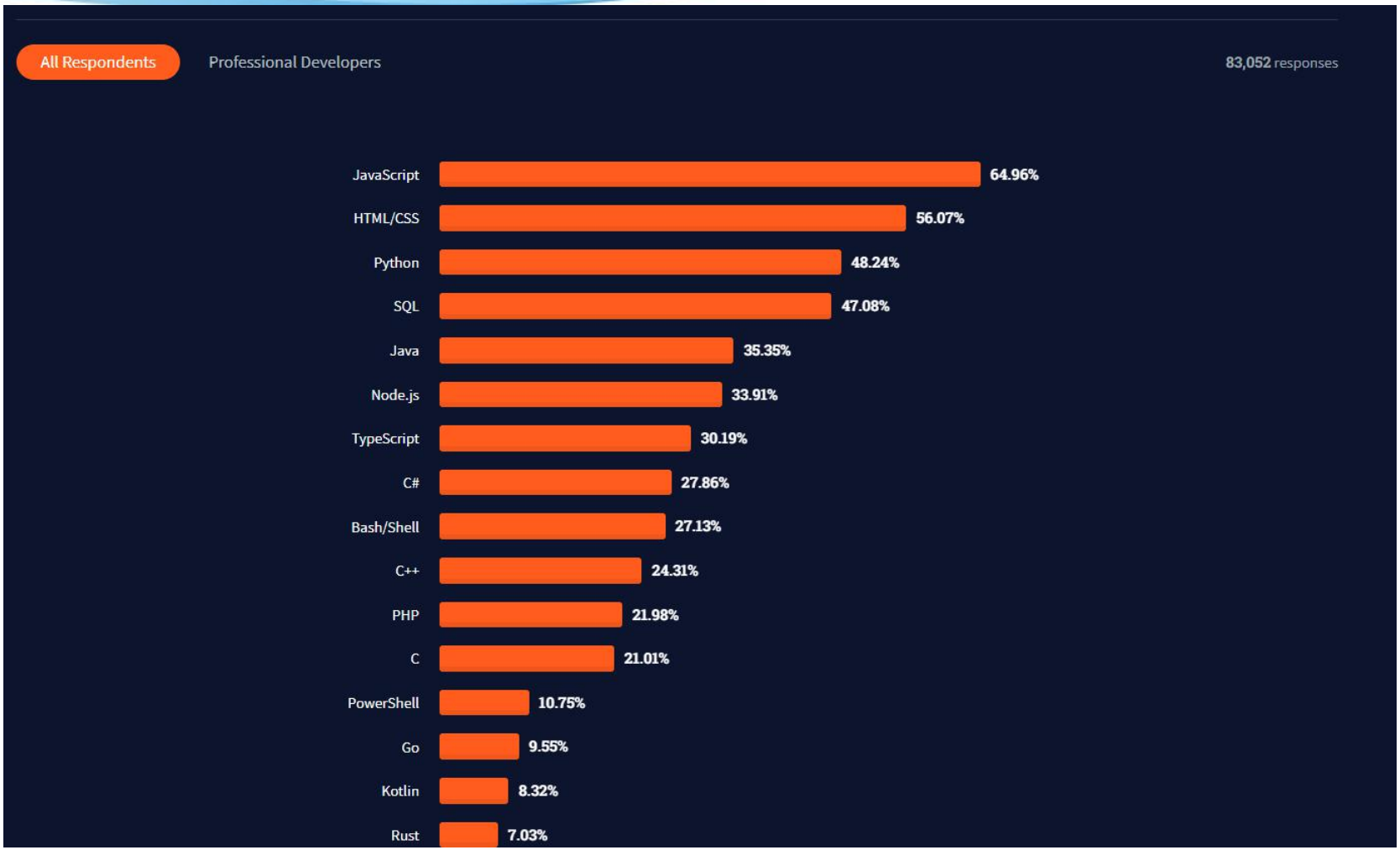
15+.1 Programming Guidelines

15+.2 Documentation

15+.3 Programming Styles & Rules

Quiz

nearly 83,000 developers took the survey.



https://insights.stackoverflow.com/survey/

Quiz

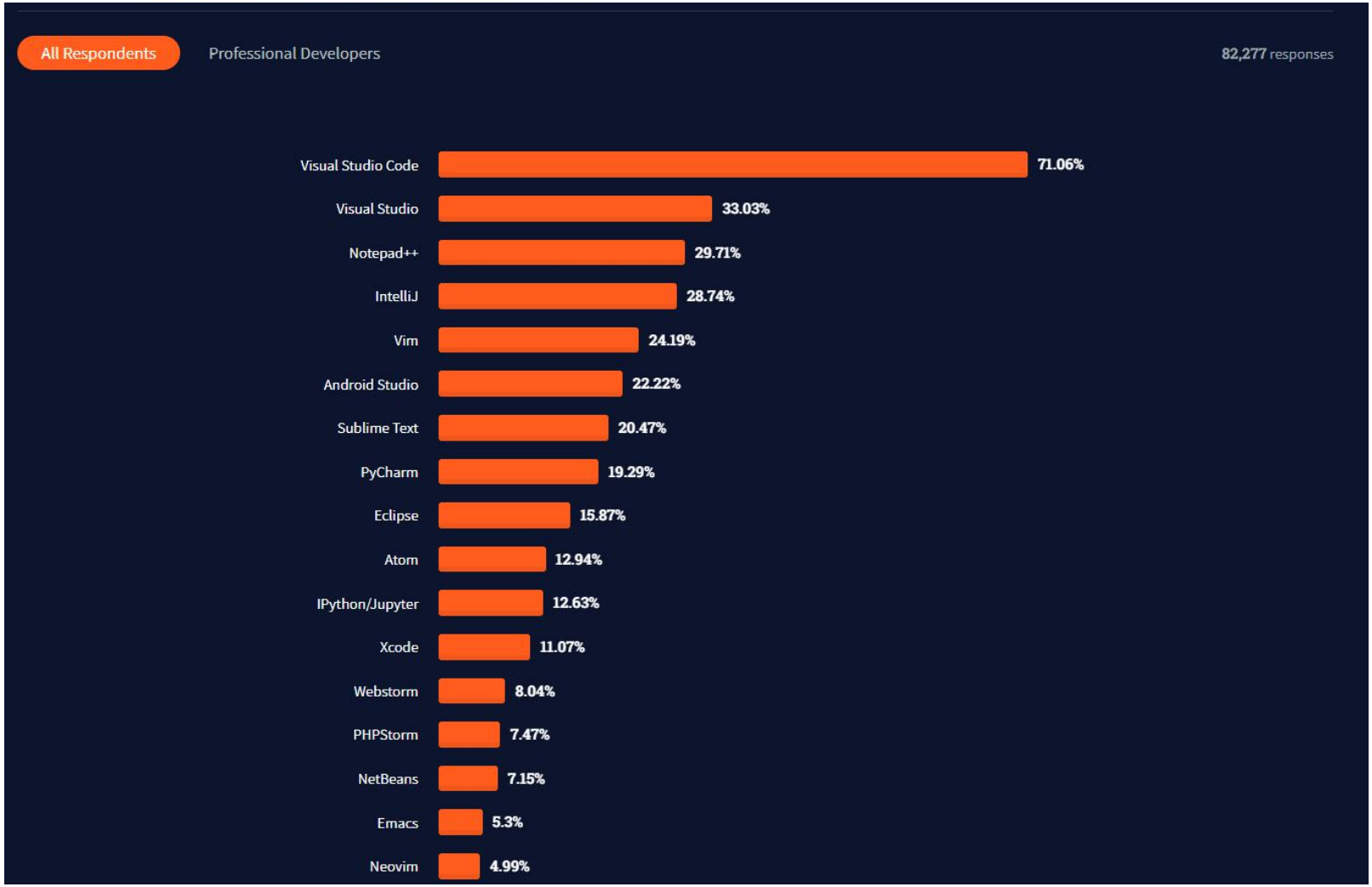
Coding as a Hobby:

Many developers work on code outside of work. Over ( ) of our respondents say that they code as a hobby.

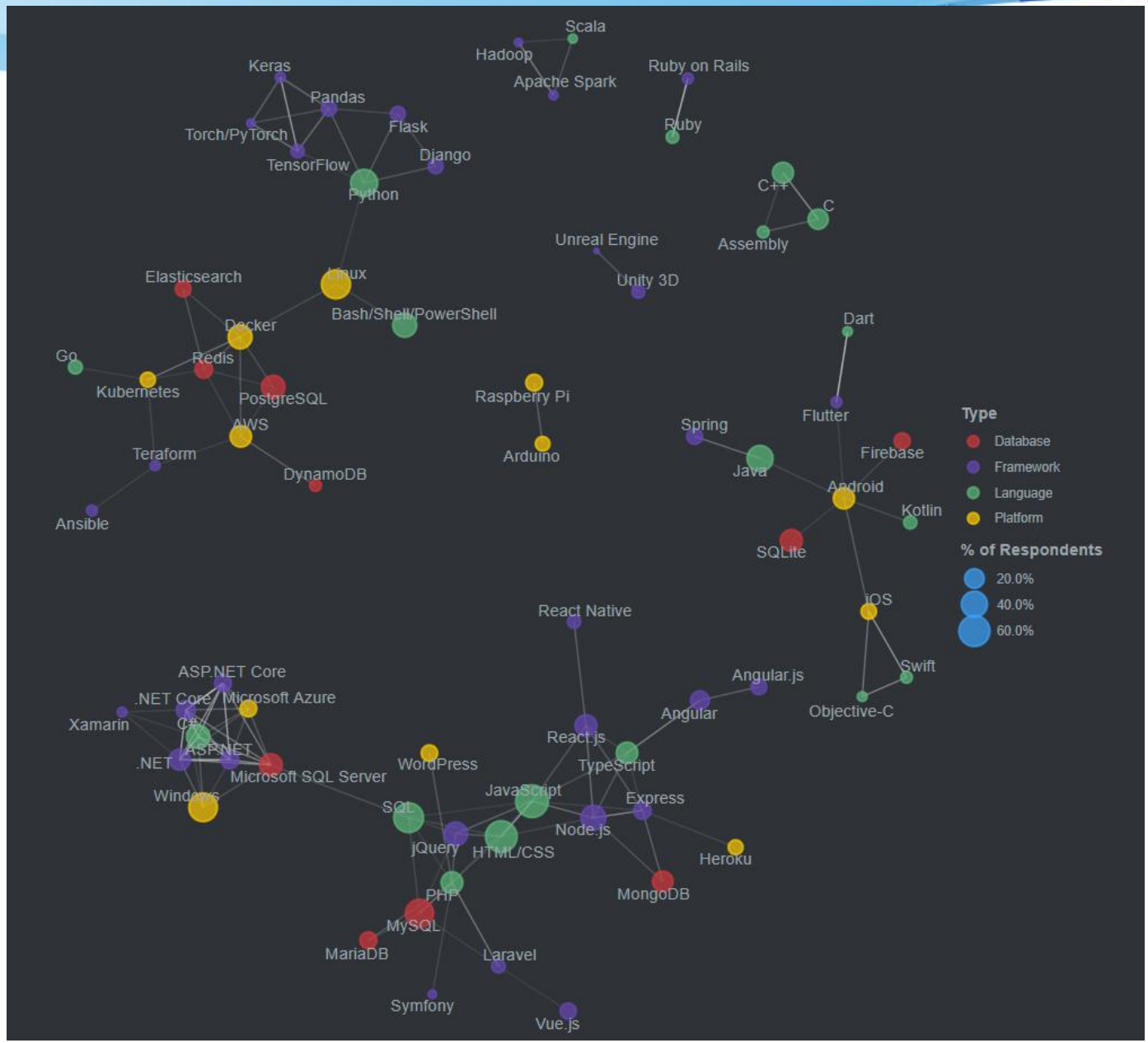
A.78% B. 60% C. 50% D.30%

https://insights.stackoverflow.com/survey/2020#overview

Developer Survey Results (2021) from stackoverflow



How Technologies Are Connected



15+.1 Programming Guidelines

Control Structures

- Make the code easy to read
- Build the program from modular blocks
- Use parameter names and comments to exhibit coupling among components
- Make the dependency among components visible

15+.1 Programming Guidelines

Example of Control Structures

- Control skips around among the program's statements

```
                benefit = minimum;
                if (age < 75) goto A;
                benefit = maximum;
                goto C;
                if (AGE < 65) goto B;
                if (AGE < 55) goto C;
A: if (AGE < 65) goto B;
                benefit = benefit * 1.5 + bonus;
                goto C;
B:              if (age < 55) goto C;
                benefit = benefit * 1.5;
C: next statement
```

15+.1 Programming Guidelines

Example of Control Structures

- Control skips around among the program's statements

```
            benefit = minimum;
            if (age < 75) goto A;
            benefit = maximum;
            goto C;
            if (AGE < 65) goto B;
            if (AGE < 55) goto C;
A:if (AGE < 65) goto B;
            benefit = benefit * 1.5 + bonus;
            goto C;
B:          if (age < 55) goto C;
            benefit = benefit * 1.5;
C:          next statement
```

- Rearrange the code

```
if (age < 55) benefit = minimum;
elseif (AGE < 65) benefit = minimum + bonus;
elseif (AGE < 75) benefit = minimum * 1.5 + bonus;
else benefit = maximum;
```

15+.1 Programming Guidelines

Algorithms

- Objective and concern: performance(speed)
- Efficiency may have hidden costs
  - cost to write the code faster
  - cost to test the code
  - cost to understand the code
  - cost to modify the code

15+.1 Programming Guidelines

- **Data Structures**

use data structure to organize the program

- – keeping the program simple
- – using a data structure to determine a program structure

15+.1 Programming Guidelines

Keep the Program Simple

Example: Determining Federal Income Tax

1. For the first $10,000 of income, the tax is 10%
2. For the next $10,000 of income above $10,000, the tax is 12 percent
3. For the next $10,000 of income above $20,000, the tax is 15 percent
4. For the next $10,000 of income above $30,000, the tax is 18 percent
5. For any income above $40,000, the tax is 20 percent

```
tax = 0.
if (taxable_income == 0) goto EXIT;
if (taxable_income > 10000) tax = tax + 1000;
else{
        tax = tax + .10*taxable_income;
        goto EXIT;
}
if (taxable_income > 20000) tax = tax + 1200;
else{
        tax = tax + .12*(taxable_income-10000):
        goto EXIT;
}
if (taxable_income > 30000) tax = tax + 1500;
else{
        tax = tax + .15*(taxable_income-20000);
        goto EXIT;
}
if (taxable_income < 40000){
        tax = tax + .18*(taxable_income-30000);
        goto EXIT;
}
else
        tax = tax + 1800. + .20*(taxable_income-40000);
EXIT;
```

15+.1 Programming Guidelines

Keep the Program Simple Example (continued)

- Define a tax table for each “bracket” of tax liability

Level	Bracket	Base	Percent
1	0	0	10
2	10,000	1000	12
3	20,000	2200	15
4	30,000	3700	18
5	40,000	5500	20

Data structure

- Simplified algorithm

```
for (int i=2; level=1; i <= 5; i++)
        if (taxable_icome > bracket[i])
                level = level + 1;
tax= base[level]+percent[level] * (taxable_income - bracket[level]);
```

15+.1 Programming Guidelines

General Guidelines to Preserve Quality

- Localize input and output
- Employ pseudocode
- Revise and rewrite, rather than patch
- Reuse
  - Producer reuse: create components designed to be reused in future applications
  - Consumer reuse: reuse components initially developed for other projects

15+.1 Programming Guidelines

Consumer Reuse

- Four key characteristics to check about components to reuse
  - does the component perform the function or provide the data needed?
  - is it less modification than building the component from scratch?
  - is the component well-documented?
  - is there a complete record of the component's test and revision history?

15+.1 Programming Guidelines

Producer Reuse

- Several issues to keep in mind
  - make the components general
  - separate dependencies (to isolate sections likely to change)
  - keep the component interface general and well-defined
  - include information about any faults found and fixed
  - use clear naming conventions
  - document data structures and algorithms
  - keep the communication and error-handling sections separate and easy to modify

15+.2 Documentation

- **Internal documentation**
  - header comment block
  - meaningful variable names and statement labels
  - other program comments
  - format to enhance understanding
  - document data (data dictionary)
- **External documentation**
  - describe the problem
  - describe the algorithm
  - describe the data

15+.2 Documentation

Information Included in Header Comment Block

- What is the component called
- Who wrote the component
- Where the component fits in the general system design
- When the component was written and revised
- Why the component exists
- How the component uses its data structures, algorithms, and control

15+.2 Documentation

Header Comment

An example of a class header

```
/**
 * MyClass <br>
 *
 * This class is merely for illustrative purposes. <br>
 *
 * Revision History:<br>
 * 1.1 - Added javadoc headers <br>
 * 1.0 - Original release<br>
 *
 * @author T.D.Bishop
 * @version 1.1, 19/04/2000
 */
public class MyClass {
    . . .
}
```

15+.2 Documentation

An example of a method header

```
/**
 * This method has no use, and it just illustrative. <br>
 * Although it does suggest adding the two parameters together ! <br>
 *
 * @param first The first number to be added
 * @param second The second number to be added
 * @return The sum of the two parameters
 * @throws BadException if something goes very wrong !
 */
public int sumNumbers(int first, int second) throws BadException{
    . . .
}
```

15+.3 Programming Styles & Rules

- Primary rule:
  - a program should be readable.

Other:

- comments
- formatting
- naming
- method
- class

15+.3 Programming Styles & Rules

- **If statement**

```
if(condition)
    return x;
return y;
```

→

```
if (condition)
{
        return x;
}
else
{
    return y;
}
```

15+.3 Programming Styles & Rules

➢Expressions

- try to avoid using intermediate variables
- brackets and priority for expression
- return value, especially the abnormal value should be checked carefully
- if order of statements is important, then add comment to illustrate

Expressions: simple, no showing off skills

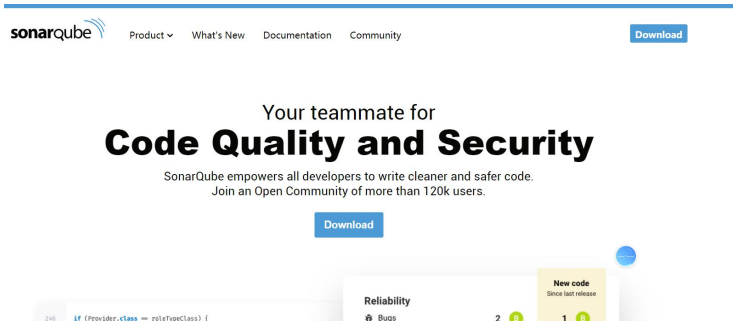
15+.3 Programming Styles & Rules

- **Language**
  - Python (Pep8)
  - Java
  - C#
  - …

https://www.python.org/dev/peps/pep-0008/

- **Company**
  - google
  - …

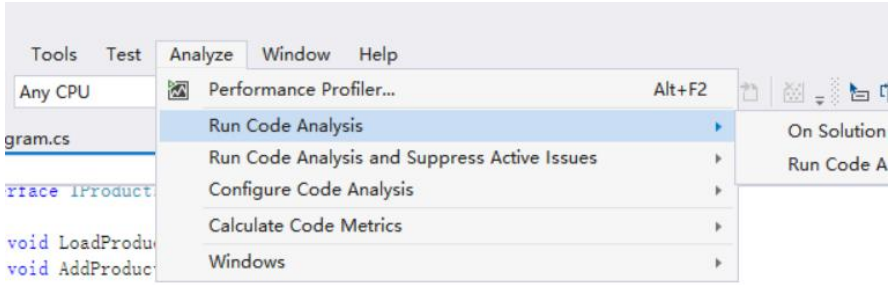
15+.3 Programming Styles & Rules



- **Java tools**

  Findbugs (bytecode)
  SonarQube
  Cobertura
  Jacoco

- **C++ /C/C#:** Static code analysis : Visual Studio



15+.3 Programming Styles & Rules

- **python tools:** Pylint, PEP 8

C:\Program Files\Python\Python37\Scripts> pylint.exe D:\2-SECode\PythonExample\ProfileTest.py -r y

```
Your code has been rated at 6.00/10 (previous run: 6.00/10, +0.00)

PS C:\Program Files\Python\Python37\Scripts> pylint.exe D:\2-SECode\PythonExample\ProfileTest.py -r y
************* Module ProfileTest
D:\2-SECode\PythonExample\ProfileTest.py:1:0: C0103: Module name "ProfileTest" doesn't conform to snake_case naming style (invalid-
name)
D:\2-SECode\PythonExample\ProfileTest.py:1:0: C0114: Missing module docstring (missing-module-docstring)
D:\2-SECode\PythonExample\ProfileTest.py:4:0: C0116: Missing function or method docstring (missing-function-docstring)
D:\2-SECode\PythonExample\ProfileTest.py:6:8: C0103: Variable name "t" doesn't conform to snake_case naming style (invalid-name)
D:\2-SECode\PythonExample\ProfileTest.py:10:0: C0116: Missing function or method docstring (missing-function-docstring)
D:\2-SECode\PythonExample\ProfileTest.py:19:0: C0116: Missing function or method docstring (missing-function-docstring)

Report
======
15 statements analysed.

Statistics by type
------------------

+---------+-------+-----------+-----------+------------+---------+
|type     |number |old number |difference |%documented |%badname |
+=========+=======+===========+===========+============+=========+
|module   |1      |1          |=          |0.00        |100.00   |
+---------+-------+-----------+-----------+------------+---------+
|class    |0      |NC         |NC         |0           |0        |
+---------+-------+-----------+-----------+------------+---------+
|method   |0      |NC         |NC         |0           |0        |
+---------+-------+-----------+-----------+------------+---------+
|function |3      |3          |=          |0.00        |0.00     |
+---------+-------+-----------+-----------+------------+---------+


Raw metrics
-----------

+----------+-------+------+---------+-----------+
|type      |number |%     |previous |difference |
+==========+=======+======+=========+===========+
|code      |17     |62.96 |17       |=          |
+----------+-------+------+---------+-----------+
|docstring |0      |0.00  |NC       |NC         |
+----------+-------+------+---------+-----------+
|comment   |2      |7.41  |2        |=          |
+----------+-------+------+---------+-----------+
|empty     |8      |29.63 |8        |=          |
+----------+-------+------+---------+-----------+
```

15+.3 Programming Styles & Rules

- **python tools:** McCabe complexity check

```python
# module.py
import random

def mtest2():
    num = random.random()
    if 0 <= num < 0.1:
        print("1")
    elif 0.1 <= num < 0.2:
        print("2")
    elif 0.2 <= num < 0.3:
        print("3")
    elif 0.3 <= num < 0.4:
        print("4")
    elif 0.4 <= num < 0.5:
        print("5")
    elif 0.5 <= num < 0.6:
        print("6")
    elif 0.6 <= num < 0.7:
        print("7")
    elif 0.7 <= num < 0.8:
        print("8")
    elif 0.8 <= num < 0.9:
        print("9")
    elif 0.9 <= num < 1:
        print("10")
```

mtest2() › elif 0.2 <= num < 0.3

```python
def mtest1():
    a = 1
    b = 2
    x = -1
    if (a == 1) & (b == 2):
        x = 1
    else:
        x = 2


if __name__ == "__main__":
    mtest1();
    mtest2();
```

Terminal: Local × python × + ⌄

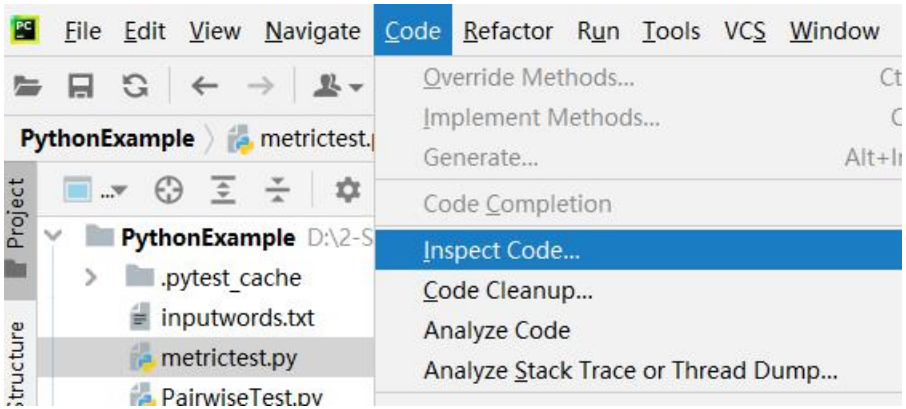
```
If 40 2
PS D:\2-SECode\PythonExample> python -m mccabe  D:\2-SECode\PythonExample\metrictest.py
4:0: 'mtest2' 11
28:0: 'mtest1' 2
```

Check: Mccabe>5

```
PS D:\2-SECode\PythonExample> python -m mccabe --min 5  D:\2-SECode\PythonExample\metrictest.py
4:0: 'mtest2' 11
PS D:\2-SECode\PythonExample>
```

15+.3 Programming Styles & Rules

- **python tools:** format check



15+.3 Programming Styles & Rules

- **python tools**

  Profile (python performance check)

```python
import profile


def profile_test1():
    for i in range(1000000):
        t = i * i
    return t


def profile_test2():
    ret = 1
    for i in range(10):
        ret = ret * (i + 1)
        # print("i=%d, ret=%d" % (i, ret))
    return ret


# factorial function
def profile_test():
    profile_test1()
    profile_test2()
```

```python


if __name__ == "__main__":
    profile.run("profile_test()")

```

15+.3 Programming Styles & Rules

- **python tools**

Profile (python performance check)

```
         7 function calls in 0.062 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.062    0.062 :0(exec)
        1    0.000    0.000    0.000    0.000 :0(setprofile)
        1    0.000    0.000    0.062    0.062 <string>:1(<module>)
        1    0.000    0.000    0.062    0.062 profile:0(profile_test())
        0    0.000             0.000          profile:0(profiler)
        1    0.000    0.000    0.000    0.000 pylinteg.py:11(profile_test2)
        1    0.000    0.000    0.062    0.062 pylinteg.py:20(profile_test)
        1    0.062    0.062    0.062    0.062 pylinteg.py:4(profile_test1)
```

ncalls: the number of calling
percall: totaltime/ncalls
percall: cumtime/ncalls

tottime: total time
cumtime: cumulated execution time

15+.3 Programming Styles & Rules

- **C++/C# tools Metric**

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) ConsoleAppTest

Debug Any CPU Start

Program.cs

ConsoleAppTest | ConsoleAppTest.ComponentOld.Status | None

```
22            switch (m_status)
23            {
24                case Status.None:
25                    Console.WriteLine("Error...");
26                    break;
27                case Status.Installed:
28                    Console.WriteLine("Hello...");
29                    break;
30                case Status.Uninstalled:
31                    Console.WriteLine("Sorry...");
32                    break;
33                default:
34                    break;
35            }
```

89 % No issues found Ln: 12 Ch: 10 SPC CRLF

Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'ConsoleAppTest' (1 of 1 p

- ConsoleAppTest
  - Properties
  - References
  - App.config
  - Program.cs

Solution... Team Exp... Resource...

Notifications

Properties

Code Metrics Results

Filter: None Min: Max:

Hierarchy	Maintainability Index	Cyclomatic Compl...	Depth of Inheritance	Class Coupling	Lines of Source code	Lines of Executable code
ConsoleAppTest (Debug)	95	16	2	5	95	16
ConsoleAppTest	95	16	2	5	95	16
Component	95	2	1	2	15	3
ComponentBlocked	100	1	2	2	4	1
ComponentInstalled	100	1	2	2	4	1
ComponentNone	100	1	2	2	4	1
ComponentOld	90	5	1	2	29	2
ComponentOld.Status	100	1	1	0	6	0
ComponentStatus	100	1	1	0	4	0
Program	75	4	1	2	25	8

Code Metrics Results Error List Output

Ready Add to Source Control

15+.3 Programming Styles & Rules

Let's watch!

sonarqube | Product | What's New | Documentation | Community

Community EDITION

The starting point for adopting code quality in your CI/CD

Download Community Edition

All the following features:

- Static code analysis for 15 languages
  Java, JavaScript, C#, TypeScript, Kotlin, Ruby, Go, Scala, Flex, Python, PHP, HTML, CSS, XML and VB.NET
- Detect Bugs & Vulnerabilities
- Review Security Hotspots
- Track Code Smells & fix your Technical Debt
- Code Quality Metrics & History
- CI/CD integration
- Extensible, with 60+ community plugins

Developer EDITION

Maximum Application Security
Maximum value across branches & PRs

Request a Free Trial License

Download Developer Edition

Community Edition plus:

- C, C++, Obj-C, Swift, ABAP, T-SQL, PL/SQL support
- Detection of Injection Flaws in Java, C#, PHP
- Analysis of feature and maintenance branches
- Pull Request decoration for:
  GitHub | Bitbucket | Azure DevOps

Discover Developer Edition →

https://www.sonarqube.org/downloads/

Summary

Code quality

- **Learn from excellent coding**
- **review**
- **learn with tools**
  - static code analysis (style, grammar)
  - measurement

THE END