

~ UNIT 3. Software Development Life Cycle ~

Lesson 14



Coding / Part 4

Programming and Bug Fixing.....	147
3 Main Types of Software Bugs.....	148
Syntax Bugs.....	148
User Interface (UI) Bugs.....	150
Logical Bugs.....	151
UI or Logical Bug? Simple Question or Not?.....	153
Version Control for Test Cases.....	154
Test Case Review Meetings.....	155
Lesson Recap.....	155
Homework.....	156
Quiz.....	157

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

**A computer lets you make more mistakes faster
than any invention in human history -
with the possible exceptions of handguns and tequila.**
-Mitch Ratliffe

Programming and Bug Fixing

Let's talk about the two main things done by the programmer:

1. **Programming.**
2. **Bug fixing.**

Here is a simplified version:

1. PROGRAMMING

A developer writes

```
child_value = parent_value + 3
```

2. BUG FIXING

According to the spec, the child_value must be equal to the parent_value **minus** 3:

```
child_value = parent_value - 3
```

So, a tester files a bug about "+" instead of "-" and the programmer does the bug fixing.

Most of the PROGRAMMING takes place during the *Coding* stage. But PROGRAMMING can also take place anytime when there is a need to write, modify, or delete code.

Most of BUG FIXING takes place during

- the *Coding* stage (developer finds his bugs) and
- the *Testing and bug fixes* stage (bugs are found by a tester).

But BUG FIXING can also take place anytime a new bug is found and deemed to be worth fixing.



Question: What does a developer do when the programming for the coming release (for example, 1.0) is finished?

Answer: he writes the code for the next release (2.0)!

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

So, when testers find bugs during the **Testing and bug fixes** stage, developers have to interrupt their work on release 2.0 and fix bugs for release 1.0.

The important thing to understand here is that **after the developer has done his work for the current Cycle, he moves to the Coding stage of the next Cycle.**

3 Main Types of Software Bugs

Now, let's learn about three main types of software bugs.

- **Syntax bugs**
- **User Interface (UI) bugs**
- **Logical bugs**

Syntax Bugs

The beauty of syntax bugs is that they are detected and even sort of explained by a compiler (if the programming language is C++) or an interpreter (if the programming language is Python) once you try to compile or execute the program.



Compilers and interpreters are special programs that translate code typed by humans ($a=2+3$) into the language understood by computers (0110000100111101001100100010101100110011).

The difference between a compiler and an interpreter is that:

- a **compiler** produces an executable file but doesn't execute our code. Thus, text files written in C++ must be compiled to become **executable** (launchable) programs.

- an **interpreter** doesn't produce an executable file but immediately executes our code. Thus, text files written in Python are already executable.

Let's look at the example of a syntax bug in C++ program. C++ uses compiler.

Here is first program of any student learning C++ (line numbers are not part of software code). This code is saved in the file **hello_world.cpp**.

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

```
1 // 'Hello World!' program
2
3 #include <iostream>
4
5 int main()
6 {
7     std::cout << "Hello World! << std::endl;
8     return 0;
9 }
```

Let's try to compile this program:

```
$ g++ hello_world.cpp -o hello_world
```

**hello_world.cpp:7:16: warning: missing terminating '"' character
[-Winvalid-pp-token]**

```
std::cout << "Hello World! << std::endl;
                ^
```

hello_world.cpp:7:16: error: expected expression

1 warning and 1 error generated.

As you can see, the C++ compiler refused to compile the program and generated an error message. The bug happened because we didn't close the double quotes after **World!**. How would we describe this bug in one sentence? **"hello_world.cpp: closing double quote is missing after World!"**

If we fix the problem, our hello_world.cpp will compile just fine, and we will get an executable file hello_world that prints "Hello, World!" when we launch it.

```
$ g++ hello_world.cpp -o hello_world
$ ./hello_world
Hello World!
```

SL

Please, file "Hello World!" bug into our bug tracking system. Fill up Summary and Description, and also specify *Prashant* in **Assigned to** field.

Let's look at the example of a syntax bug in Python program. Python uses interpreter.

...
This is promotional excerpt
from QA Mentor Course

"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

SL Go to Test Portal>More Stuff>Python Errors>register_with_error.py. As you can see, there is a syntax error in the line 414 of code:

```
body_html = get_firstpage(zip_code)
```

The programmer made a mistake by giving the wrong name of a function:

Expected: **get_first_page**
Actual: **get_firstpage**

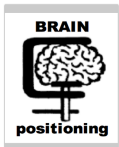
Here is a bug summary: "register_with_error.py: wrong name of the function get_first_page()".

SL Please, file this bug into our bug tracking system. Fill up Summary and Description, and also specify *Billy* in **Assigned to** field.

SL Now go to Test Portal>Application>Source Code>register.py.

If we look at line 414 of file register.py, we'll see that the function is called correctly and thus no error is generated by the interpreter:

```
body_html = get_first_page(zip_code)
```



The **only** difference between

- a **perfectly working** file register.py (Test Portal>Application>Source Code>register.py) and
- **not working at all** file register_with_error.py (Test Portal>Application>Source Code>register_with_error.py)

is a **single underscore character** ("_"). So, even the smallest modification to the existing code can break **the whole system**. That's why regression testing has immense importance.

User Interface (UI) Bugs

A UI bug is a bug in how the software *presents* the information.

UI bugs range from:

visual problems like

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

- a link on the Web page has a wrong color

to **interactive problems** like

- it's hard for a user to figure out how to use certain functions.

So remember that UI bugs cause **presentation** problems.

Logical Bugs

A logical bug is a bug in how the software *processes* information.

Logical bugs are the primary focus of the software testers because

- As a rule, it's much harder to find logical bugs than UI bugs;

AND

- As a rule, the consequences of releasing logical bugs are much more severe than the consequences of releasing UI bugs.

But keep in mind that even UI bugs can be really nasty: if users cannot figure out how to put a book into the shopping cart, then the perfectly working code of the application core doesn't really matter.

Here is a stage-by-stage illustration of a logical bug:

<Product design>

Spec #9877: "7.2. User must enter two integers from 1 to 12 and press enter. Program must print on the screen the arithmetical mean between the entered values."

<Coding>

Here is the code written in Python. That code, also called a **source code**, is in text file **get_average.py**.

```
1    #Get average
2
3    first_number = int(raw_input("Enter first number: ")) #get first integer
4
5    second_number = int(raw_input("Enter second number: ")) #get second
integer
6
```

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

```
7     average = first_number + second_number/2.0 #calculate average
8
9     print average
```

Programmer was supposed to test it, but he didn't.

<Testing and bug fixes>

Let's test it now:

```
$ python get_average.py
Enter first number: 9
Enter second number: 2
10.0
```

According to the spec, our expected result is 5.5 if we use 9 and 2 as inputs. But our actual result is 10, so we have a logical bug.

What will you do now?

1. You will file a bug against the responsible programmer. Bug summary: **"Spec9877: get_average.py: wrong calculation of arithmetical mean"**.
2. Then the programmer should look into the problem, fix it, and **reassign** the bug to you (soon we'll cover all the technicalities related to bug tracking).
3. Then you must retest the code and
 - if the bug is fixed, you close it;
 - if the bug is NOT fixed, you reassign it back to the developer.

Let's retest the code now:

```
$ python get_average.py

Enter first number: 9
Enter second number: 2
5.5
```

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

SL

Please, file this bug into our bug tracking system. Fill up Summary and Description, and also specify *Willy* in **Assigned to** field and *Checkout* in **Component** field.



We should file another bug against the PM who didn't specify the precise range of inputs. If the max valid input is 12, than the first sentence of item 7.2 must have this wording: "User must enter two integers from 1 to 12 **inclusively** and press enter."

Why do we need "**inclusively**"? Because without it:

- Some folks will be confident that "integers 1 to 12" means that the max value is 12.
- Some folks will be confident that "integers 1 to 12" means that the max value is 11.
- Some folks will be plainly confused.

So while writing specs, the PM must make sure that everyone in the spec audience will understand the spec the same way.

SL

Please, file this bug into our bug tracking system. Fill up Summary and Description, and also specify *Willy* in **Assigned to** field and *Spec* in **Found on** field.



The spec has another problem: it's not specified how the program should react to invalid input, for example, 0, 13, "A", "#", or Null input - it's when the user simply presses Enter without any data typed.

SL

Please, file this bug into our bug tracking system. Fill up Summary and Description, and also specify *Anitha* in **Assigned to** field and *Spec* in **Found on** field.

UI or Logical Bug? Simple Question or Not?

One of the students of this course once asked: **If an HTML link leads to the wrong page, is it a UI or a logical bug?**

Interesting question:

- on the one hand, it's about presentation;
- on the other hand, sometimes links are dynamically generated by our code - for example, the link "log out" is generated only if user is logged in.

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

So, our response was:

- “If the misleading link is **hard-coded** into the HTML code of the Web page, then it’s a **UI bug**, because there is no problem with processing here and the presentation is wrong.”

*BTW, **hard-coded** means that data is static and hence not generated dynamically. Programmer just typed URL when writing his code.*

- “If the misleading link is **dynamically** generated by our code, then it’s both a **logical** and a **UI bug**, because there is a definite issue with processing and there is definite issue with presentation.”

Does this make sense? **What do you think?**

...

I want to mention two more things about the **Coding** stage:

Version Control for Test Cases

During the **Coding** stage, programmers write code for the coming release and testers write test cases to test that code. We recommend that files with test suites, just like files with specs, should be

- stored in CVS (or other version control system);
- available to anyone inside the company.

The main advantages of storing files with test suites in CVS (or other version control system) are:

- it eliminates the risk of an accidental deletion of a file;
- it allows access to old editions of the file and its history;
- the file is stored on the shared server, and everyone who needs to (and who has a right to do so) can take that file to
 - > execute, modify, or delete existing test cases;
 - > append new test cases.

You can also use **test case management systems** for all mentioned tasks.

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

Test Case Review Meetings

It's a good idea to have rules about test case review meetings. Test case review meetings take place before the **Testing and bug fixes** stage and right after the tester has finished writing test cases. What happens is that the following people:

- the PM who wrote the spec,
- the programmer who is responsible for coding that spec and
- the tester who has finished test cases to test that code

get together, and the tester makes a mini presentation about the way he is going to do testing.

The tester goes from one test case to another and elaborates about test case idea, expected results and other relevant things.

The great value of this kind of meeting is that in many cases PMs and programmers

- give testers new IDEAs for testing
- find omissions and/or errors in test cases.



Right after the test case review meeting, send out an email to every participant and to those who were invited but couldn't come to the meeting. In this email, list all improvements that you've talked about during the meeting. This email is very useful because

- you'll refresh the suggested ideas in your brain;
- you'll show others that you understood them correctly.

Lesson Recap

1. Two main activities done by the programmer are **Programming** and **Bug fixing**.
2. When programmer is done with coding for release 1.0, he starts coding for release 2.0.
3. Three main types of bugs are **Syntax bugs**, **UI bugs** and **Logical bugs**.
4. Syntax bugs are caught by the **compiler** or **interpreter**.
5. The compiler creates **executable file**.

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

6. The interpreter executes the program right away.
7. UI bugs are about presentation. We usually see them in a form of visual problems. For example, wrong font color, or interactive problems, like confusing layout that make customers wonder how to use our application.
8. Logical bugs are bugs in logical processing.
9. Logical bugs represent the main focus of software testers.
10. **Even one character - added or removed - can break a working software system of any size.**
11. Test cases should be either stored in CVS or inside a test case management system.
12. During the Coding stage, testers write test cases. Once test cases are written, test case review meetings should be called, and the PM and developers should review all test cases and provide their feedback.

Homework

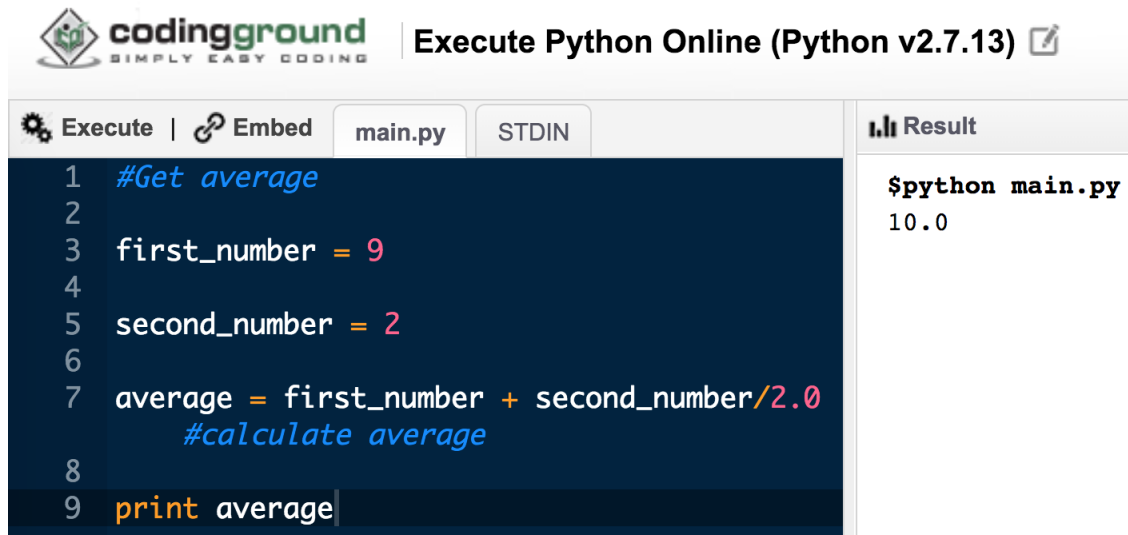
STEPS:

1. Go to https://www.tutorialspoint.com/execute_python_online.php
2. Delete all source code under the "main.py" tab.
3. Type the following source code under the "main.py" tab:

```
#Get average  
first_number = 9  
second_number = 2  
average = first_number + second_number/2.0 #calculate average  
print average
```

4. Click the **Execute** button. See picture:

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.



The screenshot shows the CodingGround Python Online IDE interface. The title bar says "Execute Python Online (Python v2.7.13)". Below the title bar are tabs for "Execute", "Embed", "main.py", and "STDIN". The "main.py" tab is active, showing the following code:

```
1 #Get average
2
3 first_number = 9
4
5 second_number = 2
6
7 average = first_number + second_number/2.0
8     #calculate average
9
10 print average
```

The "Result" tab on the right shows the output of the script:

```
$python main.py
10.0
```

5. As you can see, the result is 10.0

6. Fix the bug right in the code pane. Don't worry: it's pure arithmetic! No programming experience is needed!

7. Test your fix and click "Execute" button.

Expected result: 5.5

Quiz

Question 1: Two main activities of a developer are programming and bug fixing.

- ☐ True
- ☐ False

--

Question 2: After a developer is done with Coding for release 1.0, he

- ☐ switches to Coding for release 2.0.
- ☐ does bug fixing for release 1.0.
- ☐ All of the above.

--

Question 3: Syntax bugs are caught by a compiler or interpreter.

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

- ☐ True
- ☐ False

--

Question 4: A compiler creates an executable file out of the source file.

- ☐ True
- ☐ False

--

Question 5: An interpreter executes the source file right away.

- ☐ True
- ☐ False

--

Question 6: The color of the HTML link "Contact us" should be red, but it's green. This is a

- ☐ syntax bug
- ☐ UI bug
- ☐ logical bug

--

Question 7: A user cannot login after he completed his registration and confirmed his email. This is a

- ☐ syntax bug
- ☐ UI bug
- ☐ logical bug

--

Question 8: When a programmer launches his Python script, the interpreter issues an error message. This is a

- ☐ syntax bug

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.

- ☐ UI bug
- ☐ logical bug

This is promotional excerpt
from QA Mentor Course
"How to Become a QA Tester in 30 Days"
Do not distribute.
For private use only.