Team id : PNT2022TMID46283
Date : 15 November 2022

What is Debugging

Definition: Debugging is the process of detecting and removing of existing and potential errors (also called as “bugs') in a software code that can cause it to behave unexpectedly or crash. To prevent incorrect operation of a software or system, debugging is used to find and resolve bugs or defects. When various subsystems or modules are tightly coupled, debugging becomes harder as any change in one module may cause more bugs to appear in another.

Sometimes it takes more time to debug a program than to code it.

Description: To debug a program, user has to start with a problem, isolate the source code of the problem, and then fix it. A user of a program must know how to fix the problem as knowledge about problem analysis is expected. When the bug is fixed.

then the software is ready to use Debugging tools (called debuggers) are used to identify coding errors at various development stages. They are used to reproduce the conditions in which error has occurred, then examine the program state at that time and locate the cause. Programmers can trace the program execution step-by-step by evaluating the value of variables and stop the execution wherever required to get the value of variables or reset the program variables. Some programming language packages provide a debugger for checking the code for errors while it is being written at run time.

Here's the debugging process:

1. Reproduce the problem.

2. Describe the bug. Try to get as much input from the user to get the exact reason.

3. Capture the program snapshot when the bug appears. Try to get all the variable values and states of the program at that time.

4. Analyse the snapshot based on the state and action. Based on that try to find the cause of the bug.

5. Fix the existing bug, but also check that any new bug does not occur.

How d**ebugging works in software**

Typically, the debugging process starts as soon as code is written and continues in successive stages as code is combined with other units of programming to form

a software product. In a large program that has thousands and thousands of lines of code, the debugging process can be made easier by using strategies such as unit tests, code reviews and pair programming

To identify bugs, it can be useful to look at the code's logging and use a stand alone debugger tool or the debug mode of an integrated development environment (IDE). It can be helpful at this point if

the developer is familiar with standard error messages. If developers aren't commenting adequately when writing code, however, even the cleanest code can be a challenge for someone to debug.

Importance of debugging Debugging is an important part of determining why an operating system, application or program is misbehaving. Even if developers use the same coding standard, it's more than likely that a new software program will still have bugs. In many cases, the process of debugging a new software program can take more time than it took to write the program. Invariably, the bugs in software components that get the most use are found and fixed first.

Debugging vs. testing Debugging and testing are complementary processes. The purpose of testing is to identify what happens when there is a mistake in a program's source code. The purpose of

debugging is to locate and fix the mistake.

The testing process does not help the developer figure out what the coding mistake is -- it simply reveals what effects the coding error has on the program. Once the mistake has been error identified, debugging helps the

Examples

Some examples of common coding errors include the following:

- Syntax error
- Runtime error
- Semantic error
- Logic error
- Disregarding adopted conventions in the coding standard
- Calling the wrong function

- Using the wrong variable name in the *w*rong place
- Failing to initialize a variable when absolutely required
- Skipping a check for an error return

Debugging strategies Source code analyzers, which include security, common code errors and complexity analyzers, can be helpful in debugging. A complexity analyzer can find modules that are so intricate as to be hard to understand and test. Other debugging strategies include the following:

Debugging tools A debugger is a software tool that can help the software development process by identifying coding errors at various

stages of the operating system or application development.

Some debuggers will analyze a test run to see what lines of code were not executed. Other

debugging tools provide simulators that allow the programmer to model how an app will display and behave on a given operating system or

computing device.

Many open source debugging tools and

scripting languages do not run in an IDE, so they require a more manual approach to debugging. For example, USB Debugging allows an Android device to communicate with a computer running the Android SDK.

In this situation, the developer might debug a program by dropping values to a log, creating extensive print statements to monitor code execution or implement

```
def multiply(a, b):
    answer = a* b
    return answer
```

```
x = input("Enter first number :

y = input("Enter second number : result =
multiply (x, y) print (result)
```

Output :

```
In [4]: runfile('C:/Users/Vanshi/Desktop/gfg/untitled6.py'
wdir='C:/Users/Vanshi/ Desktop/gfg')

Enter first number : 23

Enter second number 34 Traceback
(most recent call last):

  File
      kunnstokuntiedu-py, line, in modules result =
  multiply (x, y)

  File "C:\Users\Vanshi\Desktop\gfe\untitled6.py", line 2,
  in multiply
    answer = a b

TypeError: can't multiply sequence by non-int of
type 'str


In [5]; import
pdb

In [6]: pdb.pm >
c:\userskvanshi\desktop\f\untitled6.py(2)
multiply
  1 def multiply(a, b): ---->
2 answer a b
          return answer
```

```
ipdb>
```