written by Dragos Pitica on 26 October 2016 in devops, ideas

chase the bugs

Debugging is an important part of determining why an application is misbehaving. This activity is mainly carried out by developers and it involves intensive testing execution, chasing errors and code correction. Once an error has been identified, it is necessary to actually find its cause in the code. At this point, you have a few options that would help you achieve this.

origins

First things first. Nowadays, **debugging** is already a common word, however we need to thank Grace Hooper that coined the terms **bug** and **debug** after an incident involving Harvard University's Mark II calculator. During the examination of the computer, the technical team found a moth inside. Grace Hopper said it was the " first actual case of bug being found" and that's the first time anyone used the word bug to describe a computer glitch. What a story!""

the easy one

Let's get back to some proper action. A simple method to identify the faulty bit of code is to use print statements for most steps of the program execution. This method is not very efficient as it will not provide any specific information about the problem. Also, it requires program recompilation every time a print statement is added to the code. However, this is suitable for a first step in solving the problem.

logs

Normally, there should be code implemented within the application that writes various types of events to an application log file. Since the content of the logs can be determined by the developer, it can contain informational events such as warnings and errors. Usually, an application log file is used to store information about user and system actions. Therefore, checking the logs during or after a program execution could also be a step forward in finding the main issue.

tests

An important aspect of an efficient debugging session is to use as few resources as possible. In order to achieve this, you can use the debugging component of your IDE which proves to be very useful. You need to reproduce the issue using unit tests (e.g JUnit), which allow the developer to run a specific function with specific inputs. Usually, the primary aim of tests is not to demonstrate the correctness of the functionality, but to locate the source of errors.

remote debugging

If you tried all options listed above and you're still struggling with finding the damn bug, you can use a remote debugger. Some important aspects of it were covered in a previous blog post. Anyways, this technique is pretty simple and straightforward. The application attaches a socket to itself and then listens to the data coming through a specific port. The debugger would bind itself to that port number and then send live data to that socket. At this point, you can set the breakpoints, run the application and see how the data is actually processed.

In conclusion, I'd like to share a very funny and correct statement which pretty much describes the concept of debugging.

> Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it. - Brian Kernighan

If you enjoyed the read, drop us a comment below or share the article, follow us on Twitter or subscribe to our #MetaBeers newsletter. Before you go, grab a PDF of the article, and let us know if it's time we worked together.