Miranda Edwards
mirae
1340721

Software Development Difficulties

Difficulty: Debugging in general can be difficult without the use of a debugger. Debuggers themselves are useful although using them can be tedious: the repetitive act of pausing the code execution before the problematic point, then stepping in or through each line of code slows the programmer down. The general fallback here for programmers is to debug using print statements to verify the state of execution at certain points of time and running the program normally. Adding print statements of every interesting variable is slow and requires manual inspection of the printed values, which can be prone to human error. Additionally, this litters the executed code and makes it less readable. Once the bug has been fixed, the programmer must go back through their code removing all print statements to check in their code. This process is repeated the next time a bug is found.

Current tooling that addresses this difficulty are debuggers in general, which fall short in the scenarios described above. Using logging statements instead of print statements can allow some of the debugging code to be useful in the future and left in production code, but still risks littering the logs.

I propose adding assertions to the execution of the debugger, linked to the code being debugged. Specifically, the ability to say, "Run until one of my assertions are not true, then pause execution" allows a programmer to investigate the program state at the time of execution. In other words, this is essentially a breakpoint dependent on some piece of logic at a specific location, not just a given line number. The difference between this and current tooling or methods is that these assertions lay outside of the actual code being debugged. Instead of needing to manually inspect the state of variables, the debugger validates them, reducing the allowance for human error.

Similar solutions exist in the form of 'watchpoints', for example in Visual Studio or gdb, which pause execution anywhere in code when a variable changes value. My proposition would be different in that the assertions are more in the form of a precondition, tied to a specific line of code.

Difficulty: Race conditions in multi-threaded code can be difficult to debug. Debugging a race condition requires knowledge of the ordering of code execution that caused the bug, and an ability to recreate that ordering. It's also difficult to mark a race condition as 'solved' because there's no guarantee of the ordering of threads running in the order that cause the original bug.

Tools that help find and fix race conditions exist for individual languages. Java PathFinder searches for deadlocks by executing all possible statement sequences and scheduling combinations for objects shared between threads. The Go Race Detector

Summary of Comments on software_development_difficulties

Page: 1

Number: 1 Author: mernst Subject: Highlight Date: 1/9/2018 7:56:18 PM

Here are some to think about. Debuggers are awesome tools and are very useful. However, many programmers employ logging or "printf debugging". Are those programmers cleverly employing a complementary technique? Or are they unprincipled hackers who should be ashamed of not knowing how to use the debugger? (I get the impression you are sympathetic towards the people who use logging, but is likely to be illuminating to think hard about why both of these styles of work exist.)

Number: 2 Author: mernst Subject: Highlight Date: 1/9/2018 7:57:15 PM

Do you just mean conditional print statements? Or are you thinking of there being more functionality than that in logging statements?

Number: 3 Author: mernst Subject: Highlight Date: 1/9/2018 7:57:45 PM

Do you view the assertions is being linked to specific program points (lines of code)? Or are they global assertions? (Oh, later I see you mean the former.)

For the former, most a bug or support conditional breakpoints. Execution stops at the breakpoint only if the associated condition is true.

For the latter, I believe that many debugger support memory or variable watch points (I'm not sure that is the standard term). This can be used as a kind of global condition to stop execution. If my memory is mistaken, then this would definitely be useful exhalation point

Number: 4 Author: mernst Subject: Highlight Date: 1/9/2018 7:59:54 PM

What are the pros and cons of this? You mentioned that it might be useful to leave some logging statements in. Likewise, it might be useful to associate the assertions persistently, so they can be used during a future debugging episode. Furthermore, it might be nice to write assertions in the source code because that would indicate to a reader of the code what facts are true. Triggering the assertion would drop the user into the debugger. I'm not aware of whether conditional breakpoints can be saved persistently in current tools.

Number: 5 Author: mernst Subject: Highlight Date: 1/9/2018 8:01:27 PM

Is it possible to enter the debugger when a global condition becomes false, or am I just imagining that functionality? (I admit to being more partial to logging into using a debugger.)

Number: 6 Author: mernst Subject: Highlight Date: 1/9/2018 8:02:25 PM

Do you mean there's no guarantee that they won't be in that order again? Or do you mean there's no guarantee that you can reproduce it by getting them to run in that order again during the debugging session?

Number: 7 Author: mernst Subject: Highlight Date: 1/9/2018 8:05:11 PM

Classic work from here at UW is Eraser.

watches for unsynchronized access to shared variables, which points to the affected values without giving information about the context and ordering of execution of the threads.

Traditionally, techniques for avoiding race conditions involve avoiding sharing data between threads and using mutexes aggressively. Sharing data is not always avoidable. Mutexes, by definition, limit concurrent processes with respect to a given resource and can slow down the execution of code if used incorrectly, or worse, can lead to deadlocks caused by threads attempting to acquire the same mutex.

I propose a monitoring tool that tracks the ordering of execution of processes and writes the result to the file. This output could ideally be used to recompile the trouble code and force the same code execution ordering. By being able to reproduce the ordering, tracking down the bugs caused by interweaved processes becomes simpler.

Difficulty: Writing tests can be tedious, but maintaining those tests can be even more tedious. Upon refactoring a piece of code, it's necessary to update or even rewrite the corresponding tests. It's easy enough to look for the references to a changed piece of code in tests but is harder when you're dealing with tests that indirectly rely on the changed piece of code. To express it more concretely, consider 4 functions: A(), B(), TestA(), and TestB(), and let B() call A(). If you change the behavior of A(), you may need to update the code of B(), TestA() and TestB(). Because the behavior in TestB() *indirectly* depends on the behavior of A(), it can be easy to miss updating TestB() solely via a search for references.

I propose adding a tool that tracks indirect references to changed code and highlights the potentially impacted tests. This could be color-coded and match with the original code changes, so in the case that a programmer changes 2 different pieces of code, they can identify which change impacts which tests.

Current solutions don't really consider this a problem, as far as I've found. In theory, a programmer would continually run a suite of unit tests between code changes so they never run into the case where they've changed several things and every test is broken. However, with large tests suites that are slow to build and run, this does not happen in practice, forcing the programmer to backtrack and undo each change. By mapping code changes to individual affected tests, the programmer can reduce their search space in finding which change broke which test.

Page: 2

Number: 1 Author: mernst Subject: Highlight Date: 1/9/2018 8:08:43 PM

What are the challenges to crating such a tool? On the instrumentation side, one problem is forcing a global ordering (you might want to use a technique such as vector clocks when there is no single global ordering) and the likelihood of slowdowns if you do. The slowdowns may or may not be acceptable.

I believe that the greater technical challenge would be forcing determinist agree execution. There many sources non-determinism in the program but even forcing a specific thread interleaving would require deep integration with the runtime system. I recall reading work that does this. I don't have it at the tip of my tongue but can help you look it up in the literature if you can't find it.

Number: 2 Author: mernst Subject: Highlight Date: 1/9/2018 8:05:25 PM

At the very least, this would make it much easier to reproduce the problem.

Number: 3 Author: mernst Subject: Highlight Date: 1/9/2018 8:10:02 PM

In the literature this is called "impact analysis". I recall older work by Ryder and Tip, but there is much other work as well.

Number: 4 Author: mernst Subject: Highlight Date: 1/9/2018 8:15:55 PM

Why do you think they take that point of view? Are they right? Maybe they are ignoring the problem because it doesn't come up in practice, or maybe they are ignoring the problem because they didn't notice, or maybe they are in the problem because it's too hard.

Extreme programming literature has a lot of discussion of designing for testability. That is related to what you brought up.

You might want to think about whether the specification of function A or function B is changing. Presumably it is relatively uncommon for the specification to change. However, function B could return a different value anyway. Consider that function A has been made more efficient by use of a hash table and now it returns its results in a different order that is still permitted by it specification. If the test for B was general, then you should still pass; however, if the test for B required a specific, exact output (in unnecessarily strong test oracle) then it would need to be changed.
Another possibility is that the specification of the function has not changed but his behavior has because it had a bug.
Any way, think about the use case; focusing on a specific one is likely to help you brainstorm solutions more effectively.

Number: 5 Author: mernst Subject: Highlight Date: 1/9/2018 8:16:38 PM

This feels like a separate issue to me. It's also important, but feels orthogonal.