Assignment 1

Rashmi Mudduluru
UW user ID: rashmi4

January 8, 2018

1 Development difficulty 1: Insufficient documentation

ExtendedReflection is an assembly that provides callback hooks into managed .NET code. I wanted to use this profiler to instrument the reads and writes of a program at runtime and perform some analysis. It turned out that these hooks were very poorly documented. For instance, there was a hook named MethodCall, but it was not documented whether it would be inserted before or after making the call. Similarly, there was insufficient documentation for many of the parameters of these callbacks. I had to do quite a bit of reverse engineering to figure out the semantics of the callbacks that I required.

Although this profiler is not used widely, it is a very powerful tool. During the development stage, it might seem that one is wasting time on documentation or adding comments. But, the repercussions of this are that the end users' productivity goes down significantly.

Partially automated tools could help in such scenarios. For instance, a tool that creates a comment header before each method with fields for return type and all the parameters would help the developer by performing a lot of the repetitive documentation. Further, the tool should be able to check if the values of these fields are meaningful. For instance, simply repeating the name of the parameter or writing one word comments, etc. should be pointed out as documentation errors. Another useful feature would be to give an example of the method usage. The automated tool could possibly generate an example with arguments and the developer could elaborate further.

2 Development difficulty 2

Many times, developers introduce new functionality either as part of a new feature or a bug fix. But, they do not add corresponding tests to the test suite. Also, when there is a runtime bug reported that was not caught by the compile

Summary of Comments on muddulururashmi_3710669_46053916_hw1.pdf

Page: 1

Number: 1 Author: mernst Subject: Highlight Date: 1/9/2018 4:38:05 PM
Is this redundant?

Number: 2 Author: mernst Subject: Cross-Out Date: 1/9/2018 4:38:46 PM

Number: 3 Author: mernst Subject: Highlight Date: 1/9/2018 4:39:07 PM
I'm not sure what this adds.

Number: 4 Author: mernst Subject: Highlight Date: 1/9/2018 4:40:12 PM
Minor: this is a weird transition. The first sentence of this paragraph doesn't seem to belong to it.

Number: 5 Author: mernst Subject: Highlight Date: 1/9/2018 4:40:44 PM
Doesn't every IDE do this?

Number: 6 Author: mernst Subject: Highlight Date: 1/9/2018 4:44:52 PM
There are two distinct benefits. One is filling in boilerplate text, which you seem to be focusing on. This is probably less than half of the developers time, but automating it can still be useful.

The other is simply reminding the developer to write the documentation.

Overall, you are thinking about ways to prevent the problem in the first place. That is, you're thinking of ways to force the original developer to write the documentation. A code style tool would also be effective at doing this. (For Java, running the Java dock tool warn about any missing documentation tags, the one ensure that the documentation is high-quality.)

However for most legacy code the original developer is not available. Can you think of ways to help subsequent developers deal with undocumented code once it exists?

Number: 7 Author: mernst Subject: Highlight Date: 1/9/2018 4:46:24 PM
This is important in some cases, but I don't think it would be necessary in all cases. Can you think of ways of determining when it would be useful and when it would merely be clutter?

More generally, it would be interesting to think about how much documentation is necessary. One way would be to measure how much time client developers spend writing fixing bugs. But is there any way to predict that head of time?

time tests, it is difficult to reproduce the error and developers look at logs to identify the issue and provide a fix. In this scenario, adding a test case that replicates the runtime behavior is desirable but often overlooked.

I was part of a team working with a message passing system and encountered a runtime error. The cause for this was that a particular node received an empty event and there was no corresponding action for this event. The exact sequence of events had to be inferred from the sequence of events in the log. This is a cumbersome activity and one could end up spending hours or even days. Typically runtime profiling and analysis creates overheads that are not acceptable in production code. We need analysis tools that can incur minimal runtime overheads while giving the necessary information. One possible solution is to build tools that use compile time knowledge to reduce the amount of information that needs to be tracked at runtime. This information can then be used to deduce the sequence of events that cause the unexpected behavior and the bug can be reproduced.

3 Development difficulty 3

Any team working on building a product follows the modular development approach. Typically an interface for a function specifies the types of inputs that it accepts and the also the return type. Consider the interface for a function that takes an integer as its input and returns an integer. In a modular setting, the implementation of this function might be done by one developer and another developer might call this from a different function. In this particular instance, although the interface only expects an integer, the implementation assumed that it is always positive. So there was a call to this method with a negative integer which resulted in a bug. The test cases were not exhaustive enough to catch this scenario. Automatic test generation could have helped in this case. Also, what might help is to have static analyses integrated with the IDE that generate these kinds of tests seamlessly.

4 Time spent: 5 hours

Page: 2

Number: 1 Author: mernst Subject: Highlight Date: 1/9/2018 4:49:17 PM

What information is this?
This sentence is generic; it's hard to argue with that, but it's also hard to see what it is getting at.

More generally, that is true of the entire section. It doesn't feel as concrete, perhaps because you weren't specific about what your team did in which the worst parts of that experience were.

Number: 2 Author: mernst Subject: Highlight Date: 1/9/2018 4:53:28 PM

How? I'm imagining that the function returned an unexpected value that cause the client to crash. Are you thinking that the library teen crashed? How would an automated tool know what the domain of the function is?

Suppose you have the square root function. It would be undesirable for a test generation tool to pass -1 and report that the implementation is buggy because the implementation crashes.

More generally, a test generation tool either needs a specification of the behavior (and if that existed, your hypothetical confusion between the implementer and client would never arise) or needs to use heuristics to guests at the desired behavior, and these heuristics will always fail in some cases, either letting bugs through or causing false alarms that human must investigate.

Number: 3 Author: mernst Subject: Highlight Date: 1/9/2018 5:06:39 PM

How is this different than the automatic says generation proposed in the previous sentence?