Article
Sean Connelly · Apr 12, 2017 5m read

Bug killing development tips

Does anyone NOT use a debugger? I can't remember the last time I did. It's not because I ~~don't~~ dislike them, I just don't need to use them. The main reason for this is because I have a certain development methodology that either produces less bugs, catches them at a unit test level, or makes tracking them down much easier.

Here are my tips...

1. Write your own COS cheat-sheet.

This is mainly for anyone that is new to COS. One of the biggest bug hazards is not knowing how certain commands and functions work. Take the time to play with a language and try out all of its variances. The process of then documenting your own cheat-sheet will cement the knowledge and provide a quick reference review tool. Not only will you become more productive, but you will certainly write less of those avoidable bugs.

2. Switch on track variables in Studio

If this is not already on then go to studio, tools, options, environment and check the track variables tick box. If you use a variable that has not been set first then the variable will now have a squiggly line under it. This always saves me from silly typos.

You will also see there is an "option explicit". I don't find this makes any difference, but you might find it useful. It forces all variables to have a #dim declaration first.

3. Don't switch off Syntax Checking

This should be on by default, in particular underline errors. I have seen some developers switch this off complaining about performance. There are no excuses, get a faster machine. This is also in the studio options settings.

4. Always use auto complete

This might sound like a silly one, but I never fully type out a property or method name on an object, EVER. I always start off with a few characters and then explicitly select it from the auto complete suggestions. Do this and you will never have an instance member typo bug. You will also catch deeper problems such as not using the class you thought you were using.

If your object is not auto completing then make sure you add a #dim into your code, not only will it then auto complete, but you will also help other developers read your code.

5. Write less code

This is obvious, less code means less opportunities for bugs. I watch some developers and they start bashing out hundreds of lines of code when they could have done it in half or less. I'm not talking about compacting code, just carefully curating it. Yes it might take you longer initially, but you will be using time the other person lost debugging code they wrote last week. Less haste, more speed.

6. Ensure code lives in the right place

If you have an entity class such as patient, then make sure all things that affect patient live on that class. It might sound an obvious one to seasoned OO developers, but I still see far too many developers embed this type of code in their non entity methods.

Moving this code will encourage re-usability of battle tested code, and it will also make your non entity methods smaller and easier to debug.

7. Handle ALL status codes

This is an essential strategy that I do automatically. If anything returns a status code, then I validate it before moving to the next line. If its an error always quit with that status code to a level where you handle errors.

This is probably the biggest problem that I see other developers not doing. If you are not doing this then you should.

If your developing in Ensemble then there is a useful macro that is used everywhere in the Ensemble libs...

```
$$$QuitOnError(sc)
```

I don't like macro soup, but this one is great, it makes code less verbose so that I can read the important stuff.

Also, if you use try catch, then bubble up the catch as a status code from that method.

8. Write modular reusable code

This probably falls under the same heading as write less code, but its specific to writing methods that are as small and functional as possible. Each method should have a single purpose. If you can split the method out into several smaller methods that have a defined input and output, then you can...

9. UNIT TEST, UNIT TEST, UNIT TEST

After doing all of the above, this is the single next biggest thing that will weed out errors long before I would ever need to fire up a debugger. By running unit tests and having them fail enables me to concentrate on weeding out that error from that small block of code. I don't need to debug & step around code looking for this problem, I know exactly where it is.

I have a really nice in-house unit test tool that works from a browser. No complex command line set up nonsense. If anyone is interested then I will open source it. It's particularly good for unit testing Ensemble transformations.

10. And finally, when all else fails...

What would I do without globals so that I can debug code without having to step through it. I will do something along the lines of...

```
Set ^debug($zh,"foovar")=foovar
```

And then just zwrite ^debug from the command line, or put a for loop watch on it.

This will track down my problems much quicker than having to step through lines of code, particularly with async processes.

11. One more thing, face palm moments

This is a bit of a throw away, but I do it all the time. If you are having problems that you can't track down, then take a walk and think outside the box. When you get back you will be able to see the obvious errors that you could not see because you were looking too closely. My classic is being in the wrong namespace - DOH!

Interested to hear what other tips are out there for writing bug-less code.

Updates...

12. Ruber Ducking

https://en.wikipedia.org/wiki/Rubberduckdebugging

13. Code Reviews

Very important suggestion, often missed out. I think style guides would be a good supplement to this.

14. Logging

Another good suggestion. Ensemble has a great set of tools to log and view information about your code that can highlight errors that might have gone silent, as well as tracking down intermittent problems that are hard to find.

Perhaps a community logging solution would be a good idea for Caché developers?

Sean.

#Best Practices #Tips & Tricks #Caché