Forced Exception Handling

YOU CAN NEVER DISCOUNT THE HUMAN ELEMENT IN PROGRAMMING.

who is KV?

click for video

Dear KV,

I subscribe to, "The Morning Paper," a daily summary prepared by one person, Adrian Colyer, who curates research papers and sends them out to interested readers (https://blog.acolyer.org).

Last fall he reviewed "Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems" (https://blog.acolyer.org/2016/10/06/simple-testing-can-prevent-most-critical-failures/). It had some surprising results, including:

- Almost all catastrophic failures (48 in total, or 92 percent) are the result of incorrect handling of nonfatal errors explicitly signaled in software.
- Error handlers with TODO or FIXME in the comments. This example took down a 4,000-node production cluster.
- Error handlers that catch an abstract exception type (e.g., Exception or Throwable in Java) and then take drastic action such as aborting the system. This example brought down a whole HDFS (Hadoop Distributed File System) cluster.

And the list went on from there.

I've been reading KV for quite a while, and as I read the review and then the paper itself, it looked like something you would be interested in, so I've sent along the link.

Helpfully Not in Error

Dear Helpfully,

Yes, KV also reads "The Morning Paper," although he has to admit that he does not read everything that arrives in his inbox from that list. Of course, the paper you mention piqued my interest, and one of the things you don't point out is that it's actually a study of distributed systems failures. Now, how can we make programming harder? I know! Let's take a problem on a single system and distribute it. Someday I would like to see a paper that tells us if problems in distributed systems increase along with the number of nodes, or the number of interconnections. Being an optimist, I can only imagine that it's $N(N + 1) / 2$, or worse.

I don't think you pointed out this paper to KV just to hear me bang my head on my desk while thinking distributed systems, so let's assume you're asking the "Why?" question: "Why is it the case that 92 percent of the catastrophic failures in this paper are caused by a failure to handle nonfatal errors?"

Well, let's see what else the paper had to say and then think about how software is actually implemented in the real world, rather than how we believe it ought to be implemented in the illusory world that management and marketing inhabit.

To get to the heart of why nonfatal errors might have led to fatal errors, we need look no further than this snippet from the paper: "This difference is likely because (i) the Java compiler forces developers to catch all the checked exceptions; and (ii) a variety of errors are

expected to occur in large distributed systems, and the developers program more defensively. However, we found they were often simply sloppy in handling these errors" (https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-yuan.pdf).

Hopefully anyone who has been a professional programmer for more than a few days knows that many developers will always write the code they are most interested in, or pressured to deliver first, which is not the error- and exception-handling code, nor is it test code, nor documentation, the latter two of which I have already harangued readers about, ad nauseam. What management and the rest of the team want, is "the code," and what most people see as "the code" is only the part of it that explicitly does the job you're expected to do. It's not even the demands of others that cause this narrow focus; it's often just that the error-handling parts are not as interesting to the person writing the code as getting a result. It would seem that many programmers just want to move those bits, munge that data, and show pictures of cats.

It would seem that many programmers just want to move those bits, munge that data, and show pictures of cats.

In point of fact we have a clear indication of the importance programmers put on the error-handling components of the code by this finding: "Error handlers with TODO or FIXME in the comment." Personally, I prefer XXX, as it reminds me of my time in Amsterdam in the early 1990s, and unless you're working in certain industries—industries that might also serve photos, and might still serve photos of cats—you're unlikely to find XXX as a variable in the code.

We can look at the fact that the Java compiler forces

programmers to catch all the unchecked exceptions in one of two ways. If we are charitable—and KV is the heart and soul of charity—we assume that the Java language and compiler developers are simply helping programmers make fewer mistakes and make sure that their code not only does what it is meant to do, but also acts appropriately when things go awry.

If we are less charitable, or perhaps more honest and realistic, we see this enforcement quite differently: as a naked attempt to control programmers and make them do what the language and compiler people thought was right at the time. "Programmers don't do proper error handling. I know, we will MAKE them handle errors, or their programs won't compile at all!" I believe this is said in the voice of an overbearing schoolteacher. "You *will* dot your i's! You *will* catch all exceptions!" Except that unlike dotting an i, there are ways to skate around handling the exception that was meant to be handled. In a rush? Well then, just add a TODO or FIXME or XXX in the comments and move on. You'll come back to it later... of course you will.

Both sides are a little bit wrong in this case. We can all point fingers at the person who leaves a trail of FIXMEs in the code, but who among us is without blame in that regard? We can also blame the pedants who thought that forcing every exception to be caught was doing us a favor. You can never discount the human element in programming. For everything you try to force on someone, there is something they will work to avoid if at all possible. Tool builders need to understand that the people who use their tools are often trying to get a very narrow job completed with a minimum amount of effort. Was it wrong

to add the forced exception handling into the tool? Maybe and maybe not. In the hands of someone with the time and inclination to do the right thing, these errors are a welcome way of finding problems that they do have to handle.

Clearly, in the hands of a large percentage of programmers who work on some of the most complex systems yet devised, the feature is actually a nuisance, and it is likely time to rethink how this particular exception ought to be handled.

KV

Kode Vicious, *known to mere mortals as George V. Neville-Neil, works on networking and operating-system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. Neville-Neil is the co-author with Marshall Kirk McKusick and Robert N. M. Watson of The Design and Implementation of the FreeBSD Operating System (second edition). He is an avid bicyclist and traveler who currently lives in New York City.*

CONTENTS