Whole Lotta Shakin' Going On!

(with apologies to Jerry Lee Lewis)

Come on over baby, whole lotta processes goin' on
Yes I said come on over baby, baby you can't go wrong
We ain't fakin' a whole lotta processes goin' on

I have to admit – I am not a young man anymore. In fact, unless I plan on living into my 120s, I'm not even "middle aged". As a college professor, I promise each class that they will not have to hear more than five "Back when I was your age…." stories per class. (OK – truthfully, I tell them ten). It's hard not to tell those stories – I don't think you can fully appreciate modern technology and processes unless you understand the way "it used to be done". And our profession is so young. Engineers have been building bridges, great walls, and pyramids for thousands of years. Doctors has been practicing medicine a long time – the Hippocratic oath dates from the fifth century BCE. Lawyers and politicians have been around …… never mind.

How long have software developers been around? Well, seventy-five years ago – we were using slide rulers, and we were happy to see electronic calculators (invented in 1972?) Doctors talk about blood-letting in their history, and engineers talk of building bridges with just rock and mortar. Software developers? Younger ones talk about "OMG – I actually had to code a program in FORTRAN!!"

Back in the 1970s, I was an instructor in the basic developer course at Keesler AFB, Mississippi. We taught a basic programming course using, as I recall, a Hughes 407L (although I can find no reference to this computer – my memory might be slipping. Feel free to email if you have a better memory.) Granted – it was an old computer – but still adequate for teaching basic concepts. We used it to teach assembly language programming. To the best of my memory, listed below are the steps involved in running a program. Note that this was only 45 years ago!

0. Before you started, of course, you had to punch your program onto a card deck. We used an IBM 029 model.
1. Take your card deck with you into the computer room during your reserved 30-minute slot. Kick out the developers using the computer, ignoring their "Just 5 more minutes?" please.
2. Power down the computer (which was room sized!) and power it up, to ensure clean memory.
3. Locate the "Operating System" card deck from the card shelf, and place it in the card reader.
4. Go to main panel, and press the "Boot Init" button, which would load the OS deck, and execute the code. OS now running. Return OS deck to card shelf. Locate the Assembler deck, and place it in the card reader.
5. Go to the main console, and type "load/run". This caused the OS to read the Assembler card deck into memory, and execute it. The assembler was now ready for input.
6. Place the Assembler deck back. Load your program into the card reader, go to console, and type "continue". You card deck (your program) was now loaded as data into the assembler, and the assembler, well, "assembled" it.
   a. If there was an assembly error – you got a listing of the error at the printer. You then frantically tried to fix it during your 30-minute slot, and restarted from step 2 above.
   b. However, IF your program had no errors, the card punch produced an object deck – ready to be linked and loaded.
7. Grab the "Link and Load" card deck, add your object card deck to the end, place it in the card reader, and type "load/run" on the console. The "Link and Load" program linked in code for system routines, created an executable module in memory, and executed it.
   a. IF the program ran successfully, your output showed up on the console. Tear off the paper from the console (it was a teletype, NOT a CRT monitor) and you were done!
   b. On the other hand, if an ABEND (Abnormal ENDing) occurred (i.e. the program crashed) you went to the console, typed "dumpmem/all" and retrieved a dump of ALL 32K (!) of memory on the line printer. Try and pour through memory to find error. Go to step 6b above.

Processes are MUCH simpler now. But – you know what? We have forgotten the art of desk-checking code in the last 40 years. The pain and difficulty of the steps above guaranteed that you didn't just casually type up a deck of cards without seriously reading (and re-reading) looking for typos. However, the great part of desk-checking was that you found both syntax AND semantic (logic) errors as you read through your code.

Nowadays, compiles are so quick and easy (typically, "push one button") that we no longer desk-check for syntax ("let the compiler do that!") And, sadly, we seldom desk-check for semantics until we try and run the code. We have forgotten how to individually review code for semantics. We have IDEs (Integrated Development Environments) that compile as we are coding – and fix syntax errors as we type. Why bother to desk-check?

Back in the 1970s, we had a simple process for writing code – it was called the "code and fix" process. Basically, it was a "repeat until somebody gives up" process.

Nowadays – NOBODY uses the "code and fix" process model, right? (Pause for sarcastic and guilty laughter). We have developed more complex and better processes that produce software that is more reliable, understandable, modifiable/maintainable and efficient. The processes improve quality, increase productivity, and reduce wasted time fixing the same error over and over …

But remember that a process does not replace creativity, imagination, and thinking. There is even more of a need to desk-check code. Back in the 1970s, a 2000 line program was considered huge. Now? 7 million lines of code is relatively common. Rather than just reading 2000 lines of a single program, we now need to review the code and the design of a tightly coupled 7 million line, 100+ program system. In fact, we have to review multiple components of the design: architectural, data, interface, and finally module (the code).

There has never been a greater need for good processes. Likewise, there has never been a greater need for developers who understand the process, and use their skills and intelligence and experience to keep alive the spirit of desk-checking. Don't let the process take the place of individual reviews and common sense.

It could be worse. You could still be looking for a FORTRAN compiler card deck to load into a card reader. And "keep on shakin' ".

David A. Cook, Ph.D.
Stephen F. Austin State University
cookda@sfasu.edu