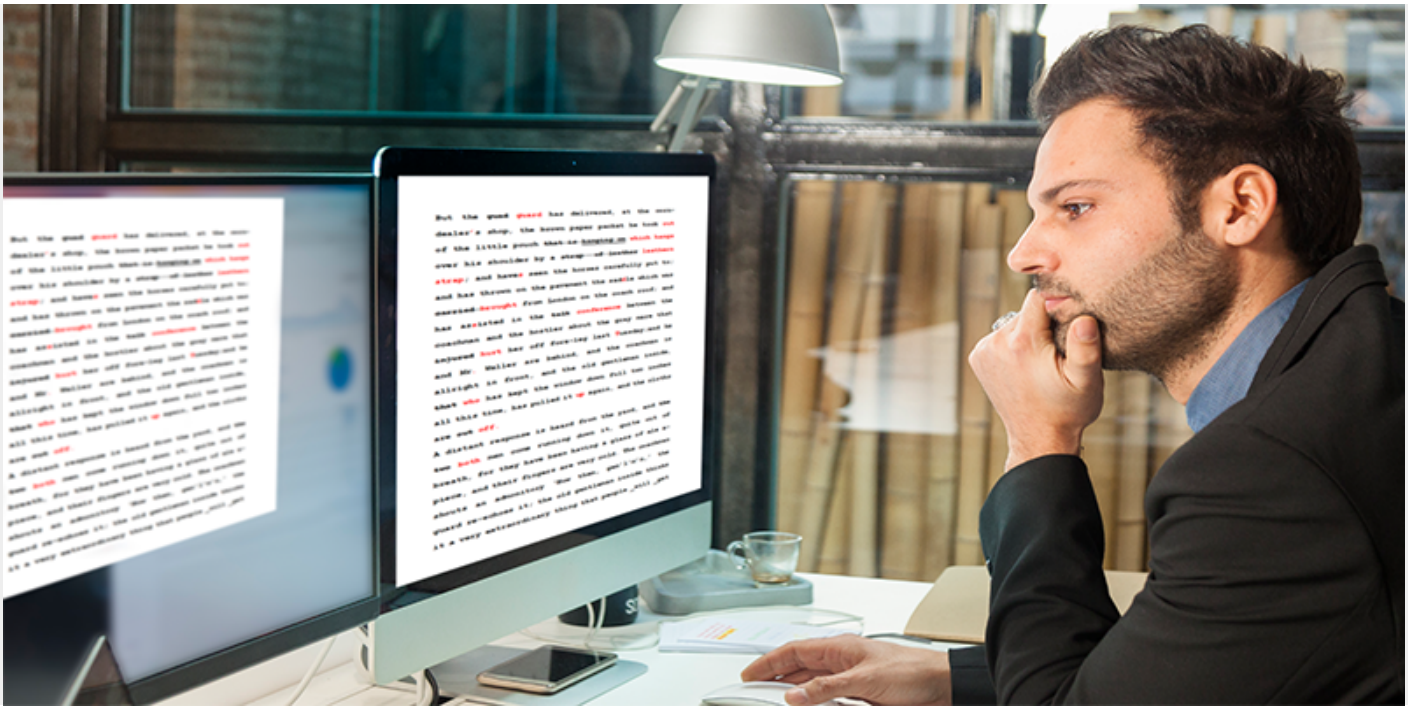
SPELL CHECK, SHIFT LEFT AND AUTOMATED TESTING



Overview: “Shifting left” and the benefits of automated testing aren’t new ideas. Everyday tools, like modern spell check features, show how this methodology improves quality, increases productivity, and lets users focus on the project at hand, rather than error checking.

Early word processors had this great feature to check spelling—I know I needed it. When you were done typing your document you would select the spell check option and it would find your errors so you could correct them. This made me much more productive because I could put out a quality document faster. I wasn't slowed down by manually checking my spelling, so I was able to focus more on the content.

But the feature evolved to make it even more useful. Up to that point, I was happy, but someone was thinking about ways to make me happier. That's great product management. In looking at the process, they realized that this spell check feature should always be on, allowing users to find errors immediately. Why does this matter? Because it's easier for the user to find errors as they are typing versus later, when they are focused on finalizing the document.

This led to a big, but subtle, improvement in my productivity. I was able to fix errors faster and without really thinking about it. Over time, the quality and accuracy of suggestions got better and better. Then I noticed that it was looking at more than spelling; it was looking at sentence structure. This started to make me a better writer, not only by making my sentences better, but by gradually training me to avoid those errors. I learned and, with time, saw those suggestions less often. It taught me in a much better manner than what I had experienced in school. I didn't know it at the time, but

with spell check, I was "shifting left" in my writing process. I was testing changes as I entered them and correcting errors *before* they got into production.

Test Early & Often

Looking at how we write code, there is something similar now: automated code quality and testing. In the past, you might wait for a compile to spot an issue; you might have all of your code ready before doing a test. Today we probably only look right before the deadline, or maybe each sprint, but what if we did it at every commit?

This is an example of the shift left in software development, but as I mentioned above, this idea isn't new. What *is* new is doing it for mainframe development. You may start minimizing errors with Topaz Editor's syntax assistance. Then you probably save your changes and wait for the compile to find errors. Once you decide to compile, you wait for those results to come back and tell you of any compile errors. You fix them and do it again. Then, when do you test, it is set up manually. This is where most of us stop – *manual* testing. Instead of testing after each save, or commit, we wait until we have "enough" to make the test worth the time and effort. By then it is late in the process, and we've already committed a lot of code. Waiting until this point in the process makes it much harder to determine the exact spot where the error was introduced.

We now have the tools to enable a quick check of your program as soon as you are ready to check it in. Using SonarQube initiated from a Jenkins script, you can get instant feedback on your changes. This can be tied into your compile process so that if there are serious errors, you won't waste the time taken to compile—it will come right back to you for changes. Once it passes that verification, it can be compiled. That's great because you have fast feedback. But it can be improved.

But Does it Work?

Just because your code compiled, that doesn't mean it will work, or that you haven't broken something. This is where automated testing with Topaz for Total Test comes in. It will automatically run a test and pass back your results. Almost immediately after saving your changes you will have a quick assessment of code quality, then the results of an actual test.

Automated testing makes turnaround faster by allowing you to check code in smaller increments, more often. This is shift left in action. You get several benefits here, like the obvious quicker fixing of problems, but some of the benefits are more subtle. You will be trained to make smaller, more frequent commits of code. And you'll want to see if there are any errors in what you have produced so you can fix them before going forward, generating a feedback loop of productivity.

Learn From Your Mistakes & Open the Door to Innovation

The next benefit, as I mentioned with the spell check feature, is being trained by the software itself. Seeing the error so close to when I created it became a learning experience, making me a better writer. That same thing will happen with coding: you will see traps you're prone to fall into and correct them before they occur.

The last benefit I'll mention is that it can unleash the creativity we all need to be great developers. With less time dedicated to testing, we can focus on the important aspects of the code itself, knowing there is a safety net right beneath us.

Reducing the drudgery of manually checking or testing code gives us more time to produce better applications for our users. And isn't that the point?