Multiple choice

# 5 key software testing steps every engineer should perform

Malcolm Isaacs
Senior Researcher, Micro Focus

In recent years, the term "shift-left testing" has entered the software engineering vernacular. But what does that mean? In plain English, it means conducting more software testing during the software development phase in order to reduce defects and save the business from costly bugs.

Shift-left testing is often used to describe increased involvement by quality assurance (QA) engineers during the development phase in an effort to detect defects as early as possible, before software engineers have handed the program over to QA for more extensive testing. Most of the time, it means developing and executing ing more automated testing of the UI and APIs.

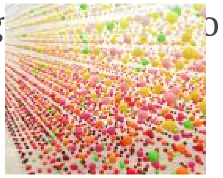
More on Testing

The in crowd
The best sof
conference:
by Linda Rose

However, there are some basic and essential software testing steps every software developer should perform before showing someone else their work, whether it's for shift-left testing, formal testing, ad hoc testing, code merging and the problem to the developer, who then has to reproduce it, debug it, and solve it, before trying agai

by Nikolay Ac

Here are the essential software testing steps every software engineer should perform before showing someone else.



Make your ap

**Why softwa explainable**

by Peter Varh

## 1. Basic functionality testing

Begin by making sure that every button on every screen works. You also need to ensure that you can enter simple text into each field without crashing the software. You don't have to try out all the different combination characters, or edge conditions, because that's what your testers do—and they're really good at that. T this: don't let other people touch your work if it's going to crash as soon as they enter their own nam username field. If the feature is designed to be accessed by way of an API, you need to run tests to m basic API functionality works before submitting it for more intensive testing. If your basic functional detects something that doesn't work, that's fine. Just tell them that it doesn't work, that you're aware of it, and that they shouldn't bother trying it. You can fix it later, just don't leave any surprises in there.

Get the best of TechBeacon
Testing to Security, delivere

## 2. Code review

Another pair of eyes looking at the source code can uncover a lot of problems. If your coding methodology requires peer review, perform this step before you hand the code over for testing. Remember to do your basic functionality testing before the code review, though.

## 3. Static code analysis

issues. Use static code analysis tools to enforce coding standards, and configure those tools to run automatically as part of the build.

## 4. Unit testing

Developers will write unit tests to make sure that the unit (be it a method, class, or component) is working as expected and test across a range of valid and invalid inputs. In a continuous integration environment, unit tests should run every time you commit a change to the source code repository, and you should run them on your development machine as well. Some teams have coverage goals for their unit tests and will fail a build if the unit tests aren't extensive enough.

Developers also work with mock objects and virtualized services to make sure their units can be tested independently. If your unit tests fail, fix them before letting someone else use your code. If for any reason you can't fix them right now, let the other person know what has failed, so it won't come as a surprise when they come across the problem.

## 5. Single-user performance testing

Some teams have load and performance testing baked into their continuous integration process and run load tests as soon as code is checked in. This is particularly true for back-end code. But developers should also be looking at single-user performance on the front end and making sure the software is responsive when only they are using the system. If it's taking more than a few seconds to display a web page taken from a local or emulated (and therefore

## Finding the right balance

Make time to run as many of these tests as possible before you hand your code over to anyone else, because leaving obvious bugs in the code is a waste of your time and your colleagues' time. Of course, you'll need to find the balance between writing code vs. testing that suits you. "Here's the mix that worked for me," said Igor Markov, LoadRunner R&D Manager at HP Software. "40 percent of my time is spent designing and writing code; 5 percent is spent on code review and static code analysis; 25 percent on unit testing and integration testing; and 30 percent on basic functionality testing and single user performance testing,"

Leaving obvious bugs in the code isn't going to do your reputation any good either. "A developer who doesn't find the obvious defects is never going to shine," continued Markov. "Developers need to produce working software that's easy to use. No one wants developers that only do pure coding. What helped me to develop my career was the fact that I always invested more time in designing, reviewing, and testing my code than in actually writing it." Producing working software that's easy to use requires exactly this type of balance. How do you balance your time between designing, reviewing, and testing code?

## Keep learning

- **Take a deep dive into the state of quality** with TechBeacon's Guide. Plus: Download the free World Quality Report 2021-22.

- **Find to tools you need** with TechBeacon's Buyer's Guide for Selecting Software Test Automation Tools.
- **Discover best practices** for reducing software defects with TechBeacon's Guide.
- **Take your testing career to the next level.** TechBeacon's Careers Topic Center provides expert advice to prepare you for your next move.

Read more articles about: App Dev & Testing, Testing