Bug Automation

01 December 2015 · automation, bug reports, ruby, Scientific Method, tools

In many of my clients, more effort is spent on ‘test automation’ than on other forms of testing or quality assurance. That can be the right choice, for example, I worked on a Data Warehousing project where we needed to write some test automation before we could test the data and its processing.

Many other projects in different technology areas also spend a lot of time on their test automation. To be precise, they spend an increasing amount of time fixing & maintaining old ‘tests’ and ‘frameworks’.

There are great tools around to help us write these automated checks quickly. But as with many software systems: maintenance, in the long term, is where the time and money goes. That is why I’m surprised we don’t use short term automation more. We have the skills.

One good example of short term automation is Bug Automation. A simple script / executable that recreates or demonstrates a bug. This isn’t a new idea, I’ve been doing it for years and I know other people have to.

Its common on open source projects to report an issue with example code, to clarify the exact issue you are reporting. Its a quick way to demonstrate the issue.

I’m not referring here to the idea of building a regression test suite from ‘tests’ (checks) from each bug fix. You can do that if you want, It can be very useful, but you are back to maintenance overhead.

By Bug Automation I’m referring to a disposable script that proves the system is broken. We can falsify (https://explorable.com/falsifiability) the assumption that we have ‘working’ software. We can’t prove the system is bug-free with our automation, but we can show its broken.

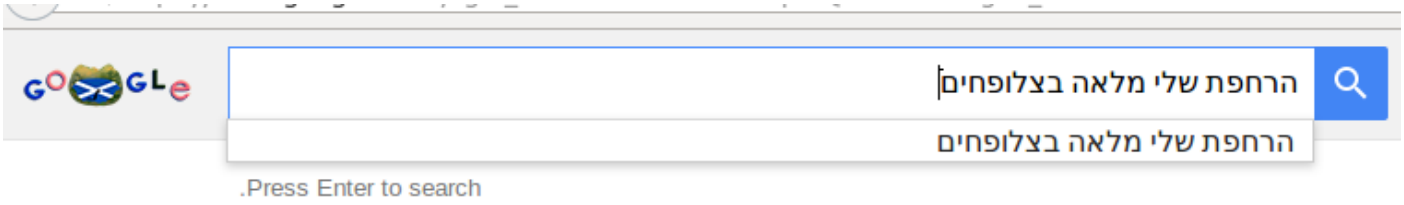
The automation isn’t there to indicate when we have fixed the issue - but to highlight that we, as a team, have created one.

In many situations a quick chat, screen-shot or URL is enough to help a developer fix a bug. But not always. For example, A tool like BlueBerry Test Assistant (http://www.bbsoftware.co.uk/BBTestAssistant/Support.aspx) could help demonstrate a bug quicker than I can explain it. But in some contexts the best tool is code.

For example: I discovered a security flaw in an open source Content Management System (http://www.investigatingsoftware.co.uk/2014/03/a-security-bug-in-symphonycms.html) used by several large media corporations, including my client at the time. I could have described the issue to people, but that would have been a poor substitute to an actual demonstration.

Its hard to persuade someone that their ‘secure’ random token generator isn’t random - its easier to show them. So I wrote some Bug Automation, and sent this along with a summary of the issue. (And together we figured out a more secure solution)

Another simple example: Google has a minor bug whereby if you enter Hebrew or Arabic text (with white-space) the full stop on the ‘Press Enter…’ message is placed at the wrong end of the sentence.



While the issue isn’t hard to describe, or screen grab (see above). Recreating the issue might not be so easy. Therefore we can create some simple Bug Automation, like this (https://github.com/phoughton/examples/blob/master/rtl_google_fail/hebrew_search_example.rb).

Other members of your team can run this script and see the issue on their own PC. They don’t have figure out how to type Right To Left languages or battle an OS or bug tracking system that doesn’t like you to copy and paste such things. Used purely as a communication aid, It also doesn’t have the maintenance overhead of trying to maintain a ‘proof’ of a fix long term.

Bug Automation is already a multimillion dollar industry, its called the Zero Day Exploit industry (http://www.wired.com/2015/04/therealdeal-zero-day-exploits/). Unfortunately that automation is often used for nefarious purposes. But as an example of positive deviance (https://en.wikipedia.org/wiki/Positive_deviance), it might be wise to pick-up on the clever things other developers & testers are doing, and use them for ourselves and for good.