The Perceived Pros and Cons of Unit Testing

Kevlin Henney
kevlin@curbralan.com
@KevlinHenney
Why isn’t unit testing a priority? Developers give the following reasons:

- They don’t know about it
- Good unit tests are hard to write
- It’s a waste of time and productivity
- Writing the tests would take too long (especially if they’re doing frequent iterations)
- Regression testing is more effective

http://itknowledgeexchange.techtarget.com/software-quality/is-unit-testing-beneficial/
If all you could make was a long-term argument for testing, you could forget about it. Some people would do it out of a sense of duty or because someone was watching over their shoulder. As soon as the attention wavered or the pressure increased, no new tests would get written, the tests that were written wouldn't be run, and the whole thing would fall apart.

Kent Beck, Extreme Programming Explained
I think that three of the best-documented tenets of economic psychology can help explain why we collectively took on the loans that events have proved were so unwise.

The first is materialism. [...]  
The second is money. [...]  

Finally, we have the most spectacular deviation from rationality: the massive myopia with which we approach choices between good things that will arrive at different points in the future. Humans are quite hopeless at such "inter-temporal choice", consistently choosing to take small benefits sooner rather than large benefits later.

The assertion that we can learn something from every failure is often heard. This study by Earl Miller and his colleagues Mark Histed and Anitha Pasupathy of the Massachusetts Institute of Technology's Picower Institute for Learning and Memory tests that notion by looking at the learning process at the level of neurons. The study shows how brains learn more effectively from success than from failure. [...] Brain cells keep track of whether recent behaviours were successful or not. When a certain behaviour was successful, cells became more finely tuned to what the animal was learning. After a failure, there was little or no change in the brain - nor was there any improvement in behaviour.

What do programmers believe they do?
Write code
What do programmers actually do?
Read code
Debug
STFW
Meetings
Discuss
Wrestle build system
Drink coffee
Copy and paste
...
Write code
...
Programmers today aren't sure their code is bug-free because they've relinquished responsibility for thoroughly testing it. It's not that management ever came out and said, "Don't worry about testing your code—the testers will do that for you." It's more subtle than that. Management expects programmers to test their code, but they expect testers to be more thorough; after all, that's Testing's full-time job.

*Steve Maguire, Writing Solid Code*
BUGRAPPORT
Using autodetected IRQ (11) to improve performance.

Ifcst (PC/TCP Class 1 packet driver - DIX Ethernet) initiates 5 free packets of length 160, 5 free packets of length 159.

The kernel is using asynchronous sends.

The Resident Module occupies 0 bytes of conventional memory.
<table>
<thead>
<tr>
<th>Bus No.</th>
<th>Device No.</th>
<th>Func No.</th>
<th>Vendor/Device Class</th>
<th>Device Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>1</td>
<td>1106 0571</td>
<td>0101 IDE Cntrlr</td>
</tr>
</tbody>
</table>

**PCI device listing ...**

- Bus No. 0
- Device No. 7
- Func No. 1
- Vendor/Device Class 1106 0571
- Device Class 0101 IDE Cntrlr
Nederland

Planner Plus

- java.lang.NullPointerException
Combining Patterns

Microsoft Visual C++ Runtime Library

Runtime Error
Program: C:\Program Files\Internet Explorer\explore.exe

6025 - pure virtual function call

Pattern-Based Software Development

© Cusbalen Ltd
Our search function is currently not available.

Search feedback
Did you find what you were looking for?  
Yes  
No
Optional

send
Teams do deliver successfully using manual tests, so this can't be considered a critical success factor. However, every programmer I've interviewed who once moved to automated tests swore never to work without them again. I find this nothing short of astonishing.

Their reason has to do with improved quality of life. During the week, they revise sections of code knowing they can quickly check that they hadn't inadvertently broken something along the way. When they get code working on Friday, they go home knowing that they will be able on Monday to detect whether anyone had broken it over the weekend—they simply rerun the tests on Monday morning. The tests give them freedom of movement during the day and peace of mind at night.

Alistair Cockburn, *Crystal Clear*
Clean code that works, in Ron Jeffries' pithy phrase, is the goal of Test-Driven Development (TDD). Clean code that works is a worthwhile goal for a whole bunch of reasons.

- It is a predictable way to develop. You know when you are finished, without having to worry about a long bug trail.
- It gives you a chance to learn all of the lessons that the code has to teach you. If you only slap together the first thing you think of, then you never have time to think of a second, better thing.
- It improves the lives of the users of your software.
- It lets your teammates count on you, and you on them.
- It feels good to write it.

Kent Beck, *Test-Driven Development by Example*
Very many people say "TDD" when they really mean, "I have good unit tests" ("I have GUTs"?) Ron Jeffries tried for years to explain what this was, but we never got a catchphrase for it, and now TDD is being watered down to mean GUTs.

http://alistair.cockburn.us/The-modern-programming-professional-has-GUTs
Would you do anything differently in the development of AWK looking back?

One of the things that I would have done differently is instituting rigorous testing as we started to develop the language. We initially created AWK as a ‘throw-away’ language, so we didn’t do rigorous quality control as part of our initial implementation.

I mentioned to you earlier that there was a person who wrote a CAD system in AWK. The reason he initially came to see me was to report a bug in the AWK compiler. He was very testy with me saying I had wasted three weeks of his life, as he had been looking for a bug in his own code only to discover that it was a bug in the AWK compiler! I huddled with Brian Kernighan after this, and we agreed we really need to do something differently in terms of quality control. So we instituted a rigorous regression test for all of the features of AWK. Any of the three of us who put in a new feature into the language from then on, first had to write a test for the new feature.
Everybody knows that TDD stands for Test Driven Development. However, people too often concentrate on the words "Test" and "Development" and don't consider what the word "Driven" really implies. For tests to drive development they must do more than just test that code performs its required functionality: they must clearly express that required functionality to the reader. That is, they must be clear specifications of the required functionality. Tests that are not written with their role as specifications in mind can be very confusing to read. The difficulty in understanding what they are testing can greatly reduce the velocity at which a codebase can be changed.

Nat Pryce and Steve Freeman
"Are Your Tests Really Driving Your Development?"
def testSetIXMpiEnv(self):
    ""
    Equivalence classes:
    comm:
    V:='salt, myr, sca, ilmpi, openmpi, mpi
    IV:='', None, 'unknown'
    forceArch:
    V:='auto, 32, 64
    IV:='', 128, None
    ixRunCmd:
    V:='ix aim'
    IV:='ix, 'aim', '', 'eclipse', None
    self.system.system_type:
    V:='pc, linux, sgi, sun, ppc64
    IV:='', 128, None
    chip:
    V:='','x86_64','ia64'
    IV:='other',None
    SECRETS:
    Intel MPI and Scali MPI paths are hardcoded here, but should be really
    read dynamically.
    ""
    ilmpiPth="/Program Files/Intel/MPI-RT/3.2.0.011/ia32/bin/mpivars.bat"
    scaOptVars = "/opt/scali/etc/scampivars.sh"
    # Currently Intersect only supports Ilmpi on Windows and scali, ilmpi
    # and Linux and openmpi on PPC64.
    # this method is only valid for IX (we are only testing IX here)
    for sysType in [Pc, Linux, Ppc64, Sgi, 'xOS', 'none', None]:
        system.SetSystemType(sysType)
        for forceArch in [UsrArch32, UsrArch64, UsrArchAuto, '']:
            for ixRunCmd in ['ix aim', None, '', 'ecl2ix']:
                for comm in [Sca, Ilmpi, Openmpi, Alt, Msmpi, Mpi, None, '', 'invalid']:
                    for chip in ['x86_64', 'ia64', 'ia64', None]:
                        self._unsetEnvVars()
                        eclrunIxMpiRunCmd = system.SetIXSerialEnv(comm, forceArch,
                            ixRunCmd, eclrun.eclipse_release, chip)
                        eclrunIxMpiRunCmd = eclrunIxMpiRunCmd.replace("","")
                        eclrunIxMpiRunCmd = eclrunIxMpiRunCmd.replace("","")
                        if sysType == 'pc' and ixMpi == "ilmpi":
                            self.assertEqual(eclrunIxMpiRunCmd, "Serial Intersect on Windows 
                            should be run as: '%s' while 
                            ECLRUN formed the command: "
                            self._isIlMpiEnvSet(chip)
                        elif ixMpi == "ilmpi":
                            self.assertEqual(str(ixRunCmd), eclrunIxMpiRunCmd,
                                "When running serial Intersect with Intel MPI" 
                                "executable then the common run command: '%s'
                                " should be left such as is (only 
                                "PATH/LD_LIBRARY_PATH/I_MPI_ROOT variables 
                                "should be set accordingly). ECLRUN changed 
                                "the cmd to: '%s'." % (ixRunCmd, 
                                eclrunIxMpiRunCmd))
                        elif ixMpi == "openmpi":
                            self.assertEqual(str(ixRunCmd), eclrunIxMpiRunCmd,
                                "When running serial Intersect with Open MPI" 
                                "executable then the common run command: '%s'
                                " should be left such as is (only 
                                "PATH/LD_LIBRARY_PATH/I_MPI_ROOT variables 
                                "should be set accordingly). ECLRUN changed 
                                "the cmd to: '%s'." % (ixRunCmd, 
                                eclrunIxMpiRunCmd))
                        else:
                            self.assertEqual(idealIxRunCmd, eclrunIxMpiRunCmd,
                                "When running serial Intersect with Scali" 
                                "executables then the expected command is: "
                                "ixMpi = eclrun.System.GetIXDefaultComm(comm, sysType)
                                "self.assertEqual(idealIxRunCmd, eclrunIxMpiRunCmd,
So who should you be writing the tests for?
For the person trying to understand your code.

Good tests act as documentation for the code they are testing. They describe how the code works. For each usage scenario, the test(s):

- Describe the context, starting point, or preconditions that must be satisfied
- Illustrate how the software is invoked
- Describe the expected results or postconditions to be verified

Different usage scenarios will have slightly different versions of each of these.

*Gerard Meszaros*

"Write Tests for People"
Ask leaders what they think makes employees enthusiastic about work, and they’ll tell you in no uncertain terms. In a recent survey we invited more than 600 managers from dozens of companies to rank the impact on employee motivation and emotions of five workplace factors commonly considered significant: recognition, incentives, interpersonal support, support for making progress, and clear goals. “Recognition for good work (either public or private)” came out number one. Unfortunately, those managers are wrong.

Having just completed a multiyear study tracking the day-to-day activities, emotions, and motivation levels of hundreds of knowledge workers in a wide variety of settings, we now know what the top motivator of performance is—and, amazingly, it’s the factor those survey participants ranked dead last. It’s progress.

http://hbr.org/2010/01/the-hbr-list-breakthrough-ideas-for-2010/ar/1
How much test coverage should your code have? 80%? 90%? If you’ve been writing tests from the beginning of your project, you probably have a percentage that hovers around 90%, but what about the typical project? The project which was started years ago, and contains hundreds of thousands of lines of code? Or millions of lines of code? What can we expect from it?

One of the things that I know is that in these code bases, one could spend one’s entire working life writing tests without doing anything else. There’s simply that much untested code. [...] Changes occur in clusters in applications. There’s some code that you will simply never change and there’s other areas of code which change quite often. The other day it occurred to me that we could use that fact to arrive at a better metric, one that is a bit less disheartening and also gives us a sense of our true progress.

*Michael Feathers, "A Coverage Metric That Matters"

http://blog.objectmentor.com/articles/2010/05/28/a-coverage-metric-that-matters
@KevlinHenney functionality is an asset, code is a liability

8:14 AM Jun 5th via Twidget

kcpeppe
In the software industry, we've been chasing quality for years. The interesting thing is there are a number of things that work. Design by Contract works. Test Driven Development works. So do Clean Room, code inspections and the use of higher-level languages. All of these techniques have been shown to increase quality. And, if we look closely we can see why: all of them force us to reflect on our code. That's the magic, and it's why unit testing works also. When you write unit tests, TDD-style or after your development, you scrutinize, you think, and often you prevent problems without even encountering a test failure.

Michael Feathers, "The Flawed Theory Behind Unit Testing"
http://michaelfeathers.typepad.com/michael_feathers_blog/2008/06/the-flawed-theo.html
Less unit testing dogma.
More unit testing karma.

Alberto Savoia
"The Way of Testivus"
http://www.artima.com/weblogs/viewpost.jsp?thread=203994