Automated Testing Importance and Impact

Joseph Bosas
Department of Energy's Kansas City National Security Campus
Honeywell Federal Manufacturing & Technologies, LLC
Kansas City, Missouri, USA
jbosas@kcp.com

Abstract—**History is filled with examples where software testing and adherence to Capability Maturity Model Integration may have prevented disaster. Consider just two past examples, the EDS Child Support System and the Ariane 5 Flight 501, where software error-related losses totaled approximately $16 billion, not including the loss of goodwill and the negative social impact.**

Honeywell manages the National Security Campus in Kansas City, MO and Albuquerque, NM, for the U.S. Department of Energy's National Nuclear Security Administration. The Kansas City National Security Campus provides technology solutions to national security challenges and is commissioned to increase the integrity of software developed. Honeywell is therefore beginning to utilize automated regression testing, not only to achieve cost savings (as compared to manual performance of tests) but also to push the envelope of traditional automated testing through an entire software diagnostics suite that includes the following: analysis of algorithmic time efficiency, percentage of code executed by tests, and automated capture/storage of testing reports for historical purposes.

I. Software Defects Within the Industry

Besides the aforementioned software defects, the software-engineering industry has contributed to numerous other defect-related costs. For example, in 2003 a memory error occurred in GE's XA/21 alarm management system, causing a blackout in the northeast portion of the United States. The outage's approximate economic cost was between $7 and $10 billion [6]. Another software error, this one in 2011, was caused by Knight Capital's new trading software, which caused computers to rapidly buy and sell millions of shares in 100+ different stocks for about 45 minutes. This inflated stock prices and resulted in a $440 million loss when the company had to sell the overvalued shares back into the market at a lower price [3].

II. KCNSC Call to Action

Software complexity is increasing within the automated test equipment (ATE) at the Kansas City National Security Campus (KCNSC). More hardware is added, greater timing accuracy is required through field-programmable gate arrays, distributed computing is needed to offload acquisition/calculations, and the number of signals within each nonnuclear missile component is increasing (Fig. 1).

The Department of Energy's Kansas City National Security Campus is operated and managed by Honeywell Federal Manufacturing & Technologies, LLC under contract number DE-NA0002839.

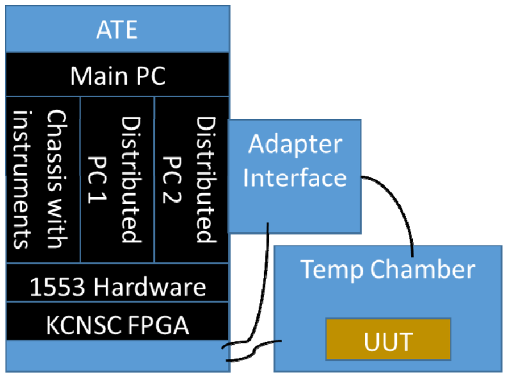


Fig. 1. The increasing complexity of automated test equipment at KCNSC.

Noting the relative increase in software complexity, KCNSC has considered the past impacts of software failures and has taken action to help prevent these catastrophes. Realizing that these software errors, and hundreds of others, are better prevented by adhering to a disciplined testing approach, KCNSC is employing Test Driven Development (TDD)—a development methodology whereby tests are always developed prior to writing software. The use of TDD does improve code quality and is proven through the use of the testing effectiveness calculation shown in Fig. 2. For example, a 2012 study proved that code quality averaged 5.66% higher in an experimental group (using test-first) than in a control group (not using test-first). This difference amounted to 244 errors in the experimental group and 330 errors in the control group [1]. To demonstrate the effectiveness of the "test often" paradigm, IBM was able to prove and document a 50% reduction in defects as a result of TDD [5].

$$Q_{code}(i) = 1 - \frac{N_{FTC}(i)}{N_{TC}}$$

Fig. 2. How to calculate an engineer's code quality where i represents an enginer, NTC represents the total number of test cases created by all engineers, and $NFTC(i)$ represents the number of an engineer's failing test cases.

III. Cost Savings vs Manual Testing

Manual testing is labor-intensive and expensive, often fails to catch software errors, and is not repeatable. Consider, for example, updating web page software. To test even a basic website a user would have to test many buttons, a couple of dropdown items, and maybe a scrolling advertisement. To manually ensure that everything on the page operates correctly, one could easily spend two full hours after every software change. Even then, it is unlikely that everything would be tested exactly as it was tested during previous changes (process variability with human involvement). Because software changes often, a reasonable conservative estimate is that a page will change twice monthly for a total of 48 testing hours yearly (or ~\$1,053.60 per year at \$43.90[1] per hour [3]). Note, however, that there is a one time fixed cost to develop the automated tests which is \$209.88[2]. When expanded using the example below, the initial development cost is negligable compared to the savings.

Given the rate at which ATE software changes while in development, it is not unreasonable to expect software to change three times per week. Assuming this rate, the yearly testing cost quickly rises to ~\$13,696.80[3]. In one department alone, KCSNC has at least eight testers in development, skyrocketing this overhead cost to ~\$109,574.40 per year. Because seven departments are actively maintaining/developing software, the cost of testing the changes manually across all ATE-developing departments is ~\$767,020.80.

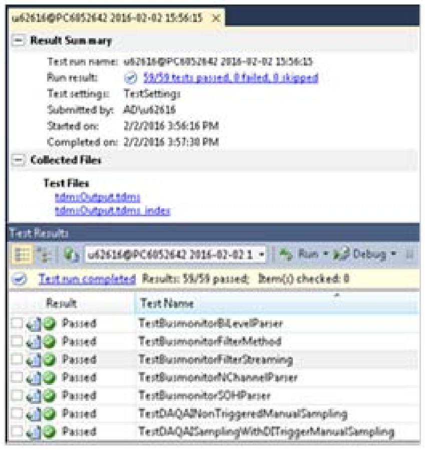
KCNSC has learned a more effective way to ensure that software changes are correct and without impact to prior functionality, and that costs stay consistent even as new features are added during development. The key is utilizing features already offered by many modern integrated development environments (IDEs). These IDE-provided tools help not only to achieve the documented cost savings, but also to go beyond the basic requirements/benefits of automated testing (read algorithmic time analysis, code coverage, and testing reports sections).

Through the use of IDE tools, the once labor-intensive testing process can be achieved in seconds (Fig. 3). In one example (Fig. 3), 59 tests were performed, were completed in 74 seconds, and were executed in the same manner as before (no process variability). At the click of a button, KCNSC is now not only able to ensure that a tester software change has not damaged any existing functionality, but also to have a level of confidence that new capability is implemented correctly.

[1] Represents Applications Software Developer average median wage in Missouri.

[2] 120 lines of test code per page at a productivity of 25.1 lines of code per hour at \$43.90 per hour. 25.1 is a KCNSC software engineer's known productivity rate and 120 lines of code is based on an actual KCNSC total.

[3] 52 weeks * 3 modifications per week * \$43.90 per hour * 2 hours every change.



u62616@PC6052642 2016-02-02 15:56:15

Result Summary

Test run name: u62616@PC6052642 2016-02-02 15:56:15
Run result: 59/59 tests passed, 0 failed, 0 skipped
Test settings: TestSettings
Submitted by: AD\u62616
Started on: 2/2/2016 3:56:16 PM
Completed on: 2/2/2016 3:57:30 PM

Collected Files

Test Files
tdms:Output.tdms
tdms:Output.tdms_index

Test Results

Test run completed Results: 59/59 passed; Item(s) checked: 0

Result	Test Name
Passed	TestBusmonitorBiLevelParser
Passed	TestBusmonitorFilterMethod
Passed	TestBusmonitorFilterStreaming
Passed	TestBusmonitorNChannelParser
Passed	TestBusmonitorSOHParser
Passed	TestDAQAINonTriggeredManualSampling
Passed	TestDAQAISamplingWithDITriggerManualSampling

Fig. 3. Example of automated testing output (summary rather than full report).

Besides ~\$750 thousand in yearly cost savings, KCNSC now also has empirical evidence of the following:

- Decreased frustration when an engineer transitions on a project that is backed by an automated testing framework
- Decreased learning time
- Increased confidence in newer engineers making software changes

Over time, KCNSC has learned that automated testing software not only provides a framework for consistent regression testing, but also harnesses the power to quickly teach new engineers what output to expect from each function, help them learn system error conditions (because system errors are checked within the automated tests), encourage them to write/update software without fear (because they can easily check the change against the tests), and help them learn what lower level functions are used throughout the code (see section V. Code Coverage for more detail).

Fig. 4. Sample automated testing timing breakdown, where the functions within LiveNotifyObservers are analyzed with respect to which consume the most testing time.

IV. Algorithmic Time Analysis

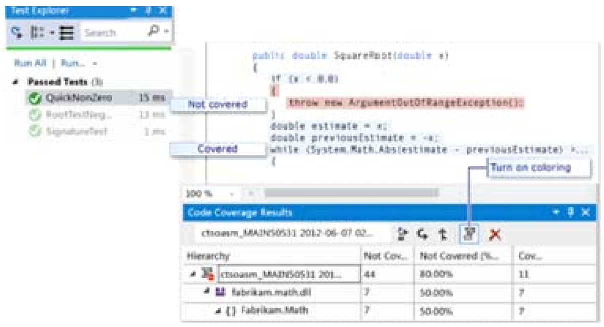
Return on investment is not the only benefit of automated testing. If the correct tools are used, total test time reduction and enhanced runtime efficiency are also paired. At the surface level, one could use asymptotic time complexity to understand an algorithm's worst-case operation (i.e., obtain an overall picture of expected performance bottlenecks). For example, knowing that a function operates at worst-case O(n^2) implies an exponential (to the 2nd power) growing function whereby total computation time increases exponentially as more elements are added. This knowledge can be used in efforts to optimize it down to something that operates in O(n), i.e., a linearly growing function whose computation time increases linearly as more elements are added. While this provides an overall picture for what may be contributing to increased total test time, automated testing tools also have the capability to analyze actual time consumed by a particular function (Fig. 4).

Fig. 4 illustrates automated testing results that highlight which functions consume the most testing time [4], corresponding directly to what consumes the most time when an operator uses the ATE in a production environment. The business impacts are reduced testing time and higher Unit Under Test (UUT) throughput, which allow operators to execute more tests. KCNSC engineers now use these results to isolate the exact functions that take up the most testing time, then target those functions for improvement. With this automated testing tool addition, KCNSC ATE now runs more efficiently and allows operators to increase UUT completion volume.

V. Code Coverage

Tools exist within modern IDEs not only to analyze exact timing, but also (in most cases) to offer code coverage analysis. Testing reports color the sections of code that have not been tested (Fig. 5), which guarantees that the engineer has covered at least some error conditions, boundary conditions, and normal conditions through the automated tests. Nevertheless, one cannot assume that a line of code being executed means that all logical operations within that line are executed (e.g., short circuit logic may require additional test cases to be written).

[4] Elapsed inclusive values represent the total time spent executing a function **and** its child functions. Elapsed exclusive values represent time spent executing a function **but not** time spent in child functions.



```
public double SquareRoot(double x)
{
    if (x < 0.0)
    {
        throw new ArgumentOutOfRangeException();
    }
    double estimate = x;
    double previousEstimate = -x;
    while (System.Math.Abs(estimate - previousEstimate) >...
    {
```

Hierarchy	Not Cov...	Not Covered (%...	Cov...
ctsoasm_MAINS0531 201...	44	80.00%	11
fabrikam.math.dll	7	50.00%	7
{ } Fabrikam.Math	7	50.00%	7

Fig. 5. Sample code coverage output.

VI. Testing Reports

Although KCNSC has established cost savings, performance analytics, and code coverage, without history of test execution the quality organization is left without evidence that rigor has been applied to our ATE development. As a solution, an automated build, test, and report-upload process has been established to show traceability over time. This process addresses the following:

- What was tested
- When testing occurred
- A differential in algorithmic performance between the baseline and new changes
- Which test(s) failed
- Which engineers could have contributed to the test failures

Because each test run report is committed to a version-control system, our quality organization is left with ample evidence that development processes were followed and that defects (i.e., test errors) are decreasing as time goes on. During the release process for the first ATE to undergo this technology, KCNSC discovered not only that many hours were saved in retroactively creating test cases, but also that fewer hours were spent handling last-minute requirement negotiations due to software bugs and/or missed requirements[5].

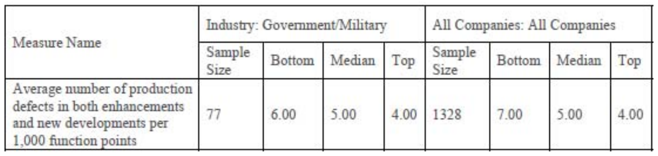
VII. Concluding Remarks

Humans will make mistakes. Unfortunately such mistakes can result in billions of wasted dollars and in risks to life. Comparative data taken on June 21, 2018, (Table 1) from the American Productivity & Quality Center (APQC), for all companies in the defense industry, show that top quartile performers have 4 defects per 1000 function points,[6] while bottom quartile companies have 7 defects per 1000

[5] Due to the low volume KCNSC ATE, there are not yet enough data points to publish a confirmed/consistent cost savings during the quality release process.
[6] A function point is the total number of user-available internal/external files, external inputs/outputs, and inquiries.

function points [2]. Although this reflects only a 3-defect difference between the upper and lower quartiles, history has shown that just 1 software defect can cause catastrophic impact. Minimizing the chance that a defect lives within the final product requires disciplined software engineering automated-testing practices. The process described here has been proven to provide thousands in cost savings, to reduce software defects, to minimize frustration during the release process, to assist in teaching newer engineers about the technology that ATEs support, and to provide a documented way to track the overall reduction in errors during development. Along with all the other benefits, utilizing these tools brings KCNSC one large step closer to its vision of "Mission First, People Always."

TABLE I. COMPARATIVE DEFECT NUMBERS IN WORLDWIDE COMPANIES OF ANY SIZE WITHIN MILITARY AND GOVERNMENT INDUSTRY



Measure Name	Industry: Government/Military				All Companies: All Companies			
	Sample Size	Bottom	Median	Top	Sample Size	Bottom	Median	Top
Average number of production defects in both enhancements and new developments per 1,000 function points	77	6.00	5.00	4.00	1328	7.00	5.00	4.00

ACKNOWLEDGMENT

The Milwaukee School of Engineering has significantly contributed to my passion for automated testing. Only through years of lab work deliverables requiring automated tests did such testing become part of every piece of software that I write. My academic and professional careers have provided years of reinforcement regarding the impact that automated testing can have on even the simplest software.

REFERENCES

[1] A. Cauevic, S. Punnekkat, and D. Sundmark, "Quality of testing in test driven development," 2012 Eighth International Conference on the Quality of Information and Communications Technology. Lisbon, pp. 266–271, 2012. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6511824&isnumber=6511765

[2] APQC, Average number of production defects in both enhancements and new developments per 1,000 function points, June 21, 2018. Retrieved from https://www.apqc.org/

[3] B. Eha. "Is Knight's $440 million glitch the costliest computer bug ever?", CNN, August 9, 2012. Retrieved June 21, 2018, from http://money.cnn.com/2012/08/09/technology/knight-expensive-computer-bug/index.html

[4] Career Cruising. "Computer Software Engineer Earnings". 2018. Retrieved July 7, 2018, from https://www2.careercruising.com/careers/Earnings/640

[5] E. M. Maximilien and L. Williams, "Assessing test-driven development at IBM," 25th International Conference on Software Engineering, 2003 Proceedings, pp. 564–569, 2003. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1201238&isnumber=27042

[6] M. Zhivich and R. Cunningham, "The real cost of software errors", Secure Systems, IEEE Security & Privacy, March/April 2009, pp. 87–90. Retrieved June 21, 2018 from https://www.ll.mit.edu/mission/cybersec/publications/publication-files/full_papers/2009_03_01_Zhivich_IEEES-P_FP.pdf