CODE COVERAGE BEST PRACTICES

Dinesh Dulipsingh,
dinesh@lexingtonsoft.com
Managing Director
Lexington Soft LLC
Boston, MA, USA

August 8, 2023

ABSTRACT

Code Coverage is a promising measure of test effectiveness. It identifies which portions of your software are tested/un-tested. An increasing number of Software R&D organizations are interested in cost-effective levels of coverage that provide sufficient fault removal with contained testing effort. While there's no silver bullet in code coverage, the findings from various organizations reveal that increasing code coverage decreases field related defects and increases a high level of confidence in the quality of the code that is being deployed. This paper discusses how a leading Fortune 100 company made Unit Testing & Code Coverage Part of their Continuous Integration Flow by leveraging an industry leading code coverage solution, ***Testwell CTC++*** from Verifysoft GmbH. The study was across 22 projects over a 20-week development cycle. This paper analyzes *Adoption Rates*, *Defect Ratio* as a function of code coverage, *Defects/KLOC* (Kilobyte Lines of Code) and *Return on Investment* (ROI) with Code Coverage. The paper discusses:

1. How software development teams can leverage the data from complex coverage levels (*MC/DC*) to achieve effective code coverage and test effectiveness.
2. What are the best practices that are referenced or enforced to enable the engineering team compliance to comply with coverage requirements?
3. Does high code coverage percentage equate to high quality in the test coverage?
4. How do you measure ROI with Code Coverage?

What are some of the ways that teams have uplifted and maintained consistently high coverage across rapid development?

Figure 1. demonstrates the benefits of **Testwell CTC++**, Code Coverage during Software Development & Continuous Integration (CI) across 22 projects resulting in a decrease in QA reported defects. The client is one of the world's leading consumer giants, successfully deploying CTC++ in one of the most challenging build/development environments. The results indicate that increase in test coverage is correlated with the decrease in QA & Field reported problems, supporting the use of code coverage measurement as a quality control criterion.

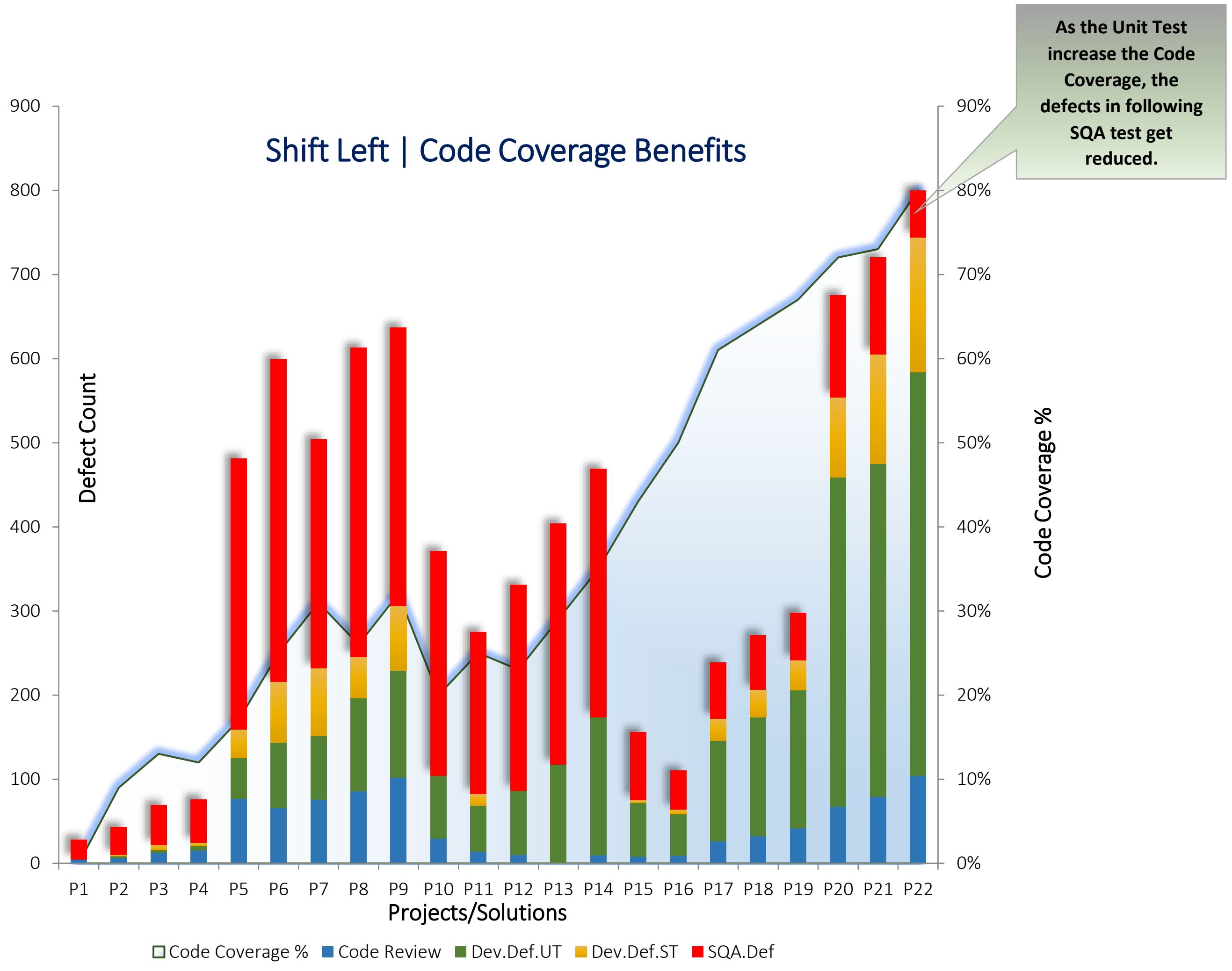


Figure 1.

The green and orange legend represent Developer Defined Unit Testing & Developer Defined System Testing respectively. The red legend represents the defects identified during Software QA. Initially for smaller projects P1 – P4, unit testing during development and system testing was little or none. Code Coverage % was less than 5%. Gradually the ***Software Agile Practice Teams started to introduce and incentivize*** development teams to write and perform unit testing and increase coverage thresholds. As the unit testing and coverage threshold increased (Refer Projects P20 – P22) the defects identified during Software QA were reduced significantly.

Figure 2. highlights the Defect Ratio as a function of Code Coverage. By introducing unit testing and increasing code coverage during development the defect ratio reduced from 84% (P1) identified during QA to 7% (P22) reinforcing the benefits of shifting left your testing.

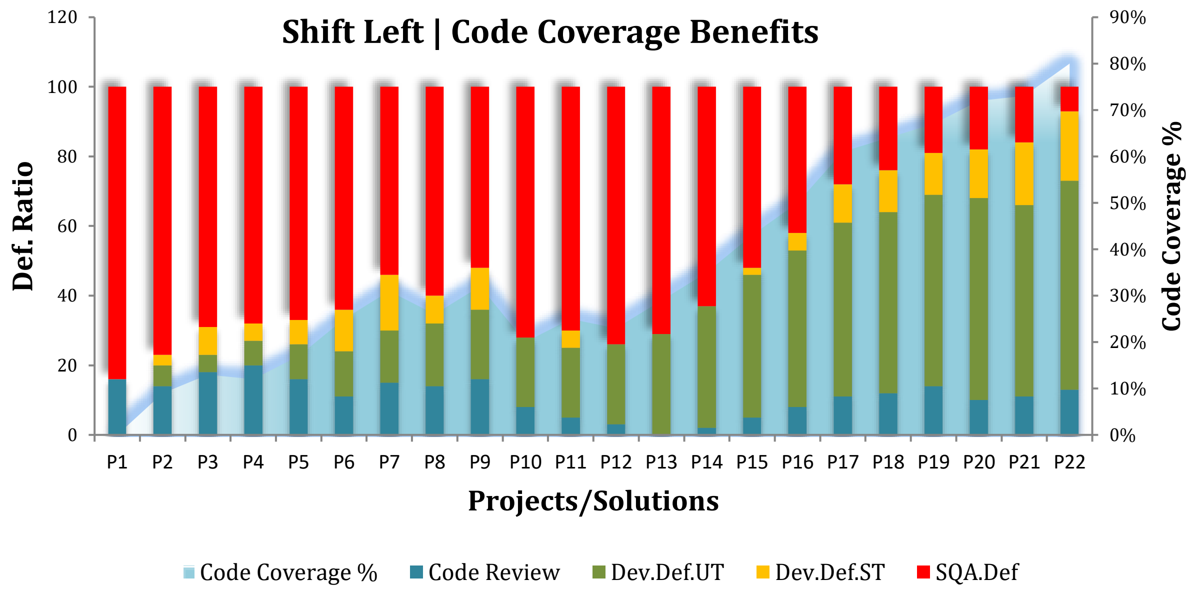


Figure 2.

The study highlighted that code coverage was associated with fewer field failures and a lower probability of field defects when adjusted for the number of pre-release changes.

The projects identified for this report are of varying degrees of complexity ranging from 10 KLOC to 70 KLOC. **Figure 3** shows the correlation between Defects/KLOC captured Pre & Post Release. It is important to observe the Defects/KLOC for Pre & Post Release identified on the Y-Axis and infer that an increase in Defects/KLOC identified during Pre-Release **inversely** affects the Defects/KLOC identified in Post Release. This strongly suggests that code coverage is a sensible and practical measure of test effectiveness.

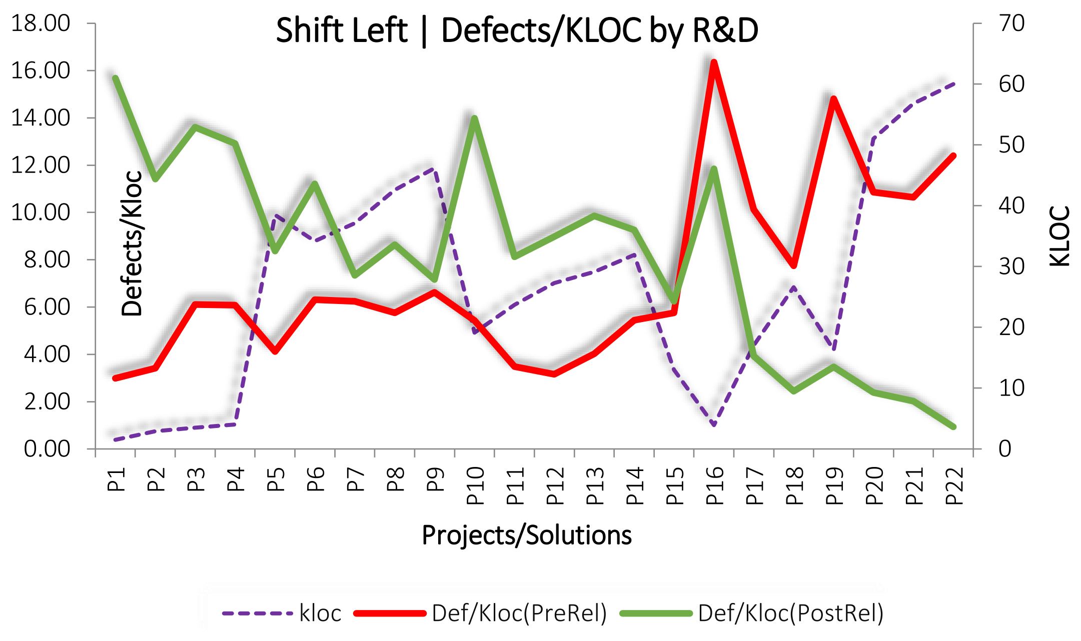


Figure 3.

What are the best practices that are referenced or enforced to enable the engineering team compliance to comply with coverage requirements?

Best practices & enforcement of code coverage will vary across organizations; however, the commonality identified is the culture shift within development/project teams that the engineering leadership needs to overcome, as was the case with this client with over **1000+** developers across **175+** projects. It would fair to state that code coverage alone does not reduce defects, our collective experience working with many clients across different industry verticals has demonstrated that efforts in increasing code coverage **WILL** lead to culture changes in engineering excellence that result in developers writing higher quality code to begin with and in the long term reduces software defects.

Reaching the organization goals of Code Coverage thresholds will happen overtime. Bear in mind that the test effort during development increases significantly to achieve a higher level of coverage. In this scenario (**Figure 4.**) the client set a threshold level of 80% and tracked the progress over 20 weeks.

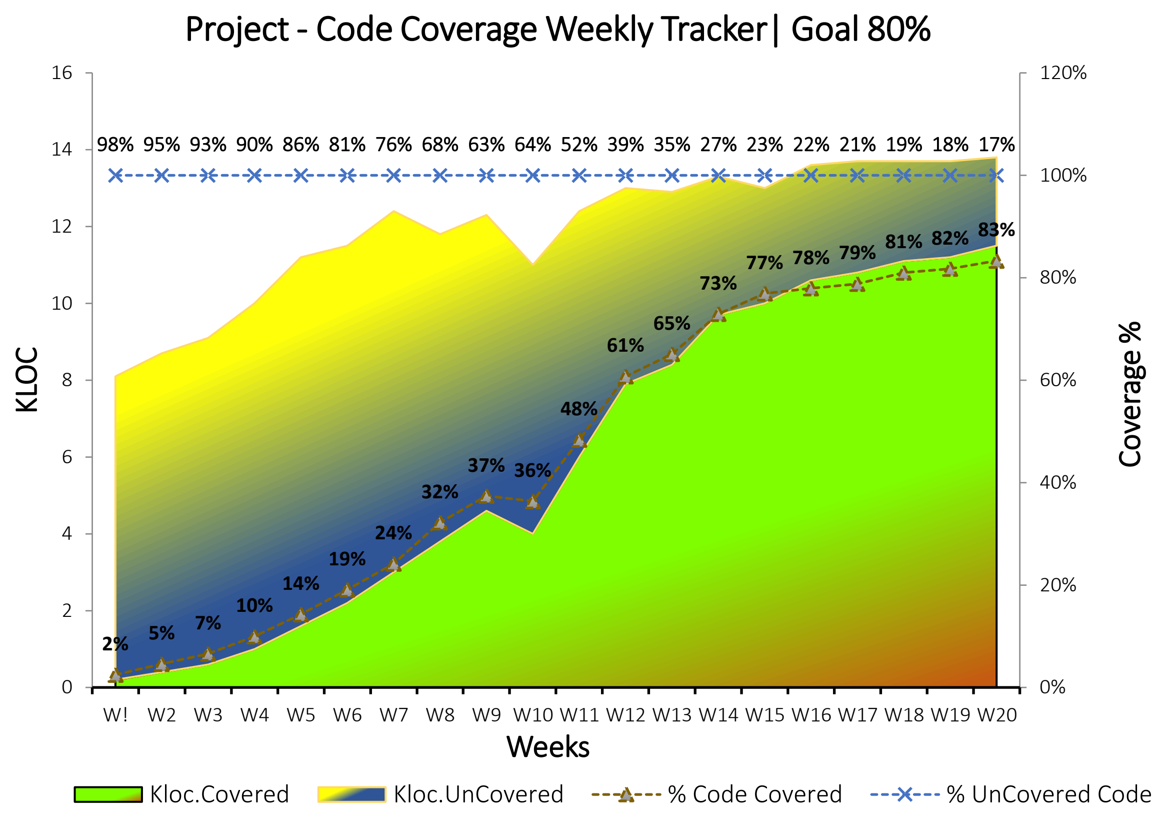


Figure 4.

The progress was gradual and the gap between Uncovered and Covered shortened over time. A best practice adopted by the Agile Software Teams to enable the engineering team compliance to coverage requirements was **recognizing & rewarding developers**.

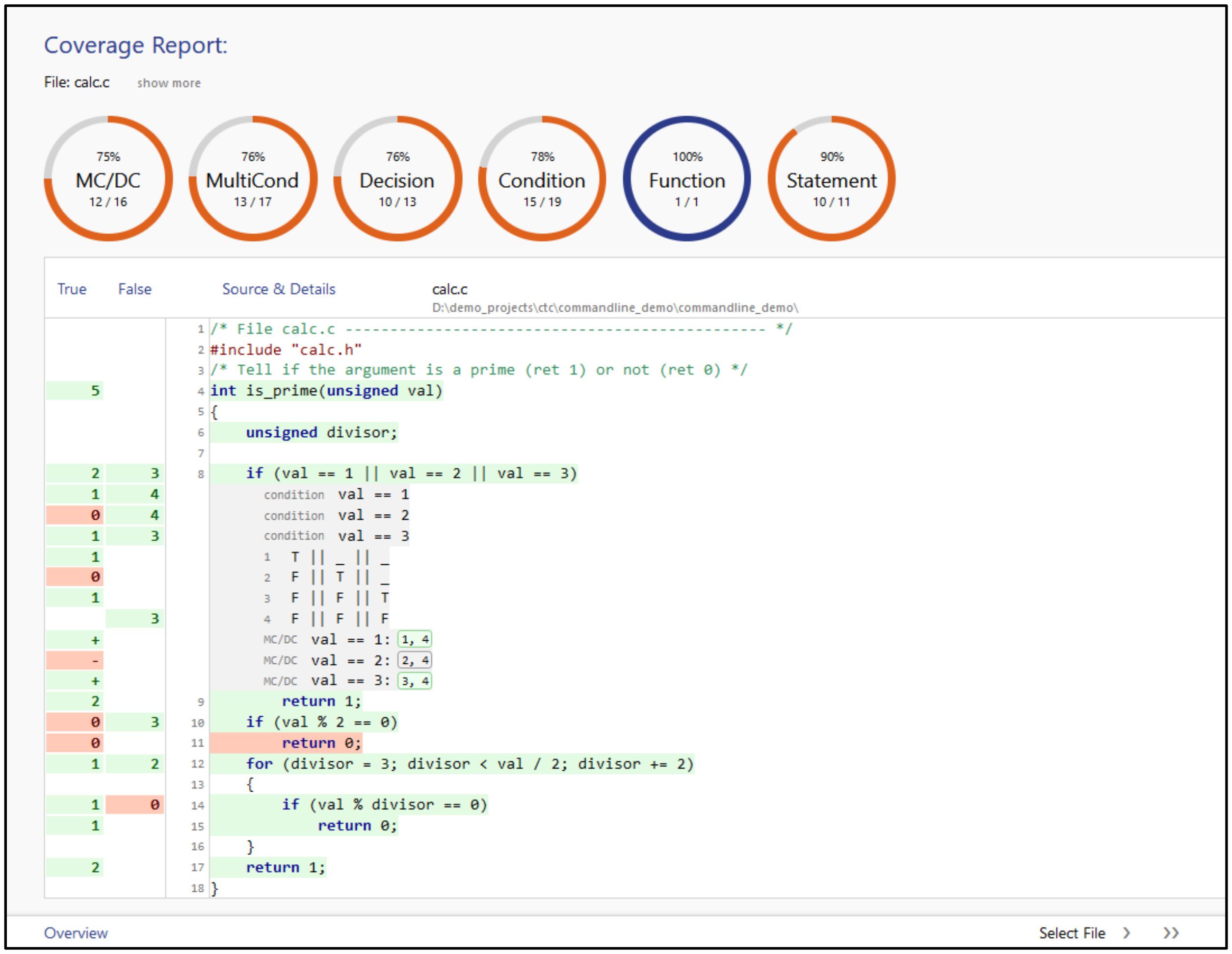
Does high code coverage percentage equate to high quality in the test coverage?

Focusing on just a percentage can lead to a false sense of security. It is important to understand the role of coverage levels to get a true measure of test effectiveness with code coverage. Testwell CTC++ support all coverage levels (***Function, Statement, Decision, Condition/Branch, Modified Condition/Decision Coverage, Multiple Condition Coverage MCC & MC/DC***)

Just functional or statement coverage can be classified as low-quality tests and can result in poor quality code being pushed to production. This is in part due to missing tests which did not cover a specific path of code. For more effective coverage and test effectiveness it is important to have a high coverage threshold for the more complex coverage levels (*Condition, MCC & MC/DC*). Take for example the most complex coverage level MC/DC, that checks if:

1. Each entry and exit point are invoked.
2. Each decision takes every possible outcome.
3. Each condition in a decision takes every possible outcome.
4. Each condition in a decision is shown to independently affect the outcome of the decision.

Testwell CTC++ shows a consolidated view (Figure 5) of all coverage levels achieved.


Coverage Report:

File: calc.c   show more

```
True  False   Source & Details     calc.c
                                   D:\demo_projects\ctc\commandline_demo\commandline_demo\
                /* File calc.c --------------------------------------------- */
                #include "calc.h"
                /* Tell if the argument is a prime (ret 1) or not (ret 0) */
   5            int is_prime(unsigned val)
                {
                    unsigned divisor;

   2      3         if (val == 1 || val == 2 || val == 3)
   1      4           condition  val == 1
   0      4           condition  val == 2
   1      3           condition  val == 3
   1                  1  T || _ || _
   0                  2  F || T || _
   1                  3  F || F || T
          3           4  F || F || F
   +                  MC/DC  val == 1: 1, 4
   -                  MC/DC  val == 2: 2, 4
   +                  MC/DC  val == 3: 3, 4
   2                        return 1;
   0      3         if (val % 2 == 0)
   0                        return 0;
   1      2         for (divisor = 3; divisor < val / 2; divisor += 2)
                    {
   1      0             if (val % divisor == 0)
   1                        return 0;
                    }
   2                return 1;
                }
```

Figure 5.

This visual representation arms developers with accurate information on which portions of your code has been executed/not executed and more specifically facilitates developers to write unit test cases so that each condition should be evaluated at least once which affects the decision outcome independently.

MC/DC coverage is a requirement in many industries to ensure compliance with Safety Critical Standards like ISO 26262 (Automotive) and DO-178C (Aerospace). Development teams that give priority for the more complex coverage levels derive the intangible benefit of incorporating stronger testability into their product design, which results in higher quality code and reducing defects downstream in the Software Development Life Cycle (SDLC).

How do you measure ROI with Code Coverage?

ROI is a measure of tangible and intangible costs. Tangible engineering costs can be computed by the number of defects identified during the SDLC. Defect Prevention Cost varies based on

1) *When in the SDLC the defect is identified and fixed*
2) *Defect Severity*
3) *Average number of developers hours it takes to develop a fix*

Engineering costs for identifying & fixing defects increase exponentially when defects are found further right in the SDLC.

www.isixsigma.com
Defect Prevention Cost

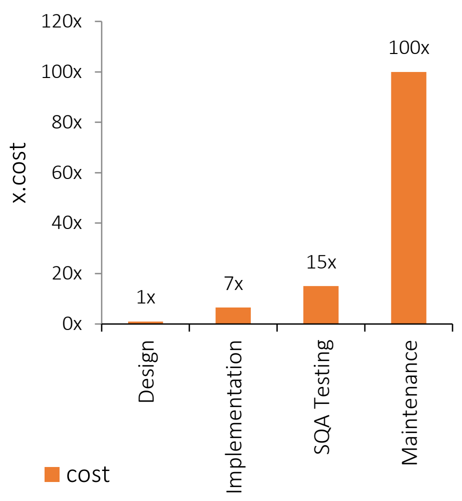
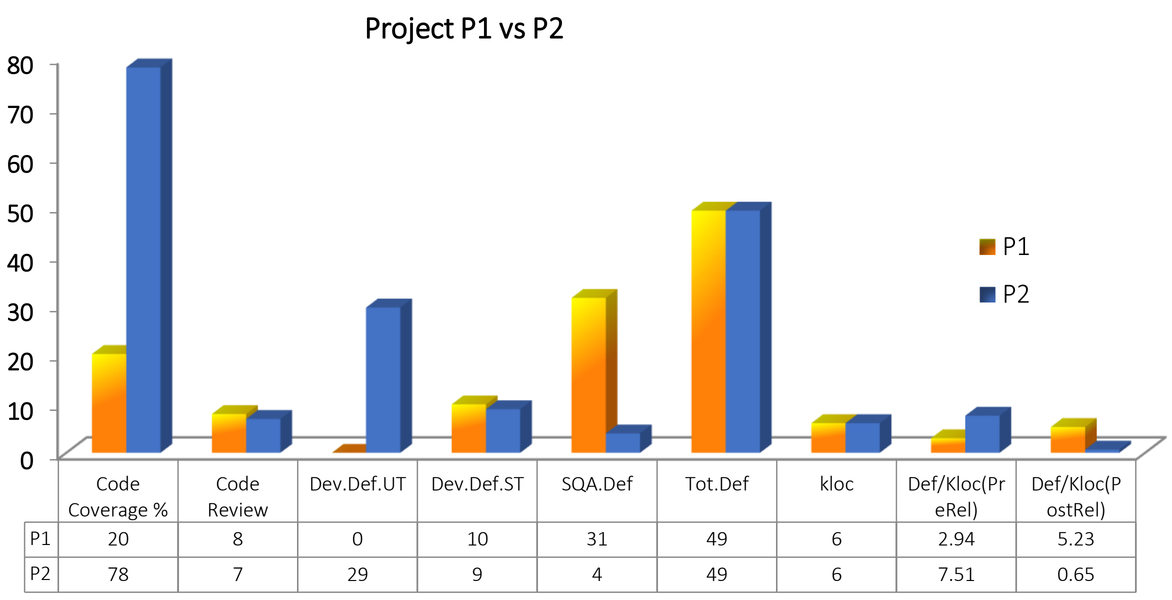


Figure 6.

The report compares project P1 vs. P2. In summary Project P2 identified more defects during Unit & System Testing with a higher Defect/KLOC (Pre-Release) and a low Defect/KLOC (Post-Release). Refer **Figure 7.**



	Code Coverage %	Code Review	Dev.Def.UT	Dev.Def.ST	SQA.Def	Tot.Def	kloc	Def/Kloc(PreRel)	Def/Kloc(PostRel)
P1	20	8	0	10	31	49	6	2.94	5.23
P2	78	7	29	9	4	49	6	7.51	0.65

Figure 7.

Effectively the total cost of poor quality (Figure 8.) = [Costs associated with identifying and fixing defects during Unit & System Testing] – [Costs associated with identifying and fixing defects QA Testing]. In this example the difference in the cost of poor quality between P1 & P2 was **~233x**.

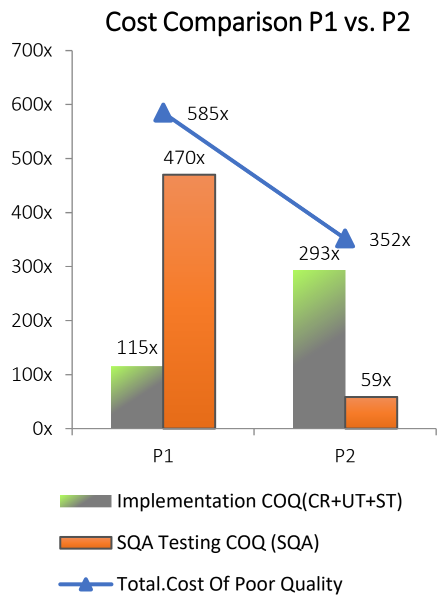


Figure 8.

SUMMARY

This paper discusses how Testwell CTC++ was leveraged to make Unit Testing & Code Coverage part of the Continuous Integration flow. Key findings from this study revealed:

- An increase in unit testing and coverage threshold levels resulted in a decrease in defects identified during Software QA.
- An increase in Defects/KLOC identified during Pre-Release inversely affects the Defects/KLOC identified in Post Release
- A culture shift with development teams is required to bridge the gap between uncovered and covered code.
- Complex coverage levels (MC/DC) can be leveraged to develop unit test cases to achieve effective code coverage and test effectiveness.
- Leveraging code coverage during unit and system testing results in a high ROI.

We hope that the data and experiences described in this paper motivate software development teams to make code coverage an integral part of their development workflow. We value your feedback, please email us at info@lexingtonsoft.com.

Testwell CTC++
33+ YEARS OF CONTINUOUS DEVELOPMENT
PROVEN SUCCESS SINCE 1989
700+ LONG TIME CUSTOMERS IN 40+ COUNTRIES

REQUEST FREE EVALUATION