SOFTWARE QUALITY AND PRODUCTIVITY: ARE THEY COMPATIBLE?

Rick Long
TYBRIN Corp

Peter Crump
TYBRIN Corp

ABSTRACT

Many view quality and productivity as competing concepts. After all, doesn't high software quality come at a high cost? Doesn't it mean that a large amount of "extra stuff" needs to be done during the software development cycle? And, doesn't that mean that software productivity takes a back seat to (and a major hit from) quality efforts? This paper will explore these issues.

This paper provides some preliminary data that supports how a disciplined software engineering process can (and has) resulted in high quality software while actually increasing productivity. Data has been gathered on organizations that have a disciplined, quality-oriented software engineering process in place. That data shows that quality and productivity can (and do) coexist. The data will be discussed along with an explanation of how these results can be achieved.

KEY WORDS

Software engineering, quality, productivity, Personal Software Process, Team Software Process

INTRODUCTION

People in the software industry frequently discuss both productivity and quality. However, the two are rarely mentioned in the same sentence. Why? Quality efforts are generally regarded as counter to productivity efforts. Doesn’t all the “stuff” one does to ensure high quality products get in the way of the software engineer and cause them to delay delivery? In the highly competitive software industry, any delay in fielding a new product could be catastrophic to a company. So, how does one ensure that high-quality software products are delivered in a timely manner? This paper will explore the elements

of software productivity as well as the elements of software quality and compare the value of one on the other.

BODY

At its most basic level, software productivity is how fast a software engineer can produce software code. How fast they can generate code is a driver as to how fast software products can be produced and, subsequently, fielded. Anything that gets in the way of the engineer is detrimental to the development organization.

Software productivity is typically measured in lines of source code (LOC) developed per hour or per day. One way to calculate productivity is to "divide the amount of product produced by the hours you spent." [Humphrey] In software terms, this is all the software produced (we'll use lines of code as our measure) divided by the number of hours spent producing this new and/or changed software LOC.

In his book Quality Is Free, The Art of Making Quality Certain, Crosby defines quality as "conformance to requirements." [Crosby] To divide this notion into parts, we find that 1) a software product must do what the user needs it to do, 2) it must do it when the user needs it, and 3) the software product must work. The basic measure of quality for this paper is the number of defects per thousand lines of code (KLOC).

Traditionally, software engineers avoid quality-oriented tasks like the plague. Why would they want to inject all that overhead into their task, anyway? Software quality tasks are cumbersome and time-consuming. Wouldn't it just make them less productive? After all, if they aren't performing coding doesn't their productivity suffer?

That may not be the case. Many believe that quality efforts and software productivity are compatible. In fact, Watts Humphrey (Software Engineering Institute Fellow and developer of the Capability Maturity Model) recently stated that one should start with quality in mind then productivity would fall into place. This would seem to indicate that if an organization wanted to increase productivity, it should institute a software quality program. Then, the productivity issue should resolve itself.

An interesting question to pose is: does quality efforts actually increase software productivity? The question seems counter-intuitive. How is it possible to inject a serious amount of quality overhead into the development process and see productivity actually increase? The simple answer is yes … because of the positive impact a quality product has on time spent in testing.

The largest impact that quality has on the software development lifecycle can be observed in the testing phases. The entire thrust of quality injection into the software

development process is to reduce defects. The impact of the thrust will manifest itself in the reduction of defects remaining in the product to be found in unit, integration and system testing. Naturally, this will reduce the amount of time required for testing.

Test phase costs (which directly relate to effort) for a given software development range between 30 and 50 percent of the development lifecycle while the range for implementation is between 17 and 28 percent. [Sommerville] Any positive impact on test times will reduce the total lifecycle time – regardless of what causes the impact. So, will injecting a lot of extra quality "stuff" into the development phases pay off enough in testing to make up for the extra time spent in quality efforts? Results from quality efforts are beginning to filter in that indicate that, in fact, it can. Actually, some indicate that productivity can even increase (in an absence of productivity-increasing efforts) simply because of quality efforts. How is this possible? What follows is a short example.

Let's take an example development (at the far end of the test spectrum) that requires 50 percent of the development lifecycle for testing and 20 percent of its time in design and coding. Then, inject a quality process on the development that slightly increases design and coding time in those phases. Obviously, this will cause the developer's productivity to decrease somewhat during the design and coding phase. However, take a look at what it might do for the test phases. Results from both the Department of Defense and private industry indicate a dramatic positive impact in unit, integration and system test phases.

The following data results from studies on both the Personal Software Process [SM] (PSP) and the Team Software Process [SM] (TSP). The PSP is a quality-oriented software development process used for the development of a unit or component of code. The PSP focus is on the individual engineer. The TSP is also a quality-oriented software development process, geared toward teams of PSP-trained individuals. The following PSP information pertains to the detailed design, design review, code, code review and unit testing of a unit or component. It does not include the requirements, high-level design, integration testing or system testing phases of a system or set of related unit or component.

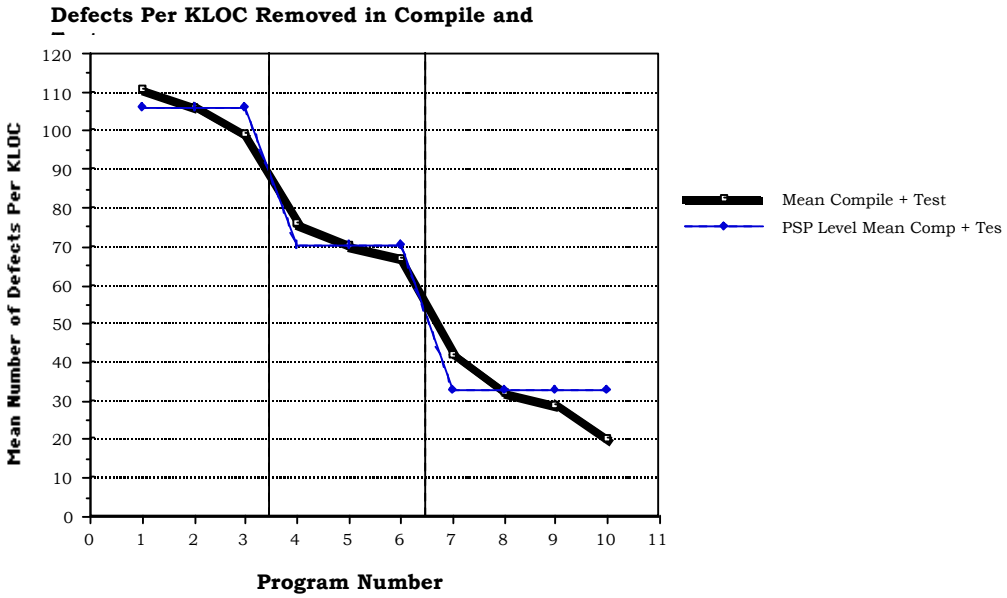
Personal Software Process Data

The Software Engineering Institute (SEI) has gathered quality and productivity results from 298 students going through the PSP for Engineers class. The data shows that *both* quality and productivity increases during development of a unit or component. Defects go from an average of about 110 per KLOC at the beginning of the class to about 20 per KLOC at the end. If one looks at what the students are taught and the techniques employed during the PSP class, it becomes obvious why this happens – PSP injects many

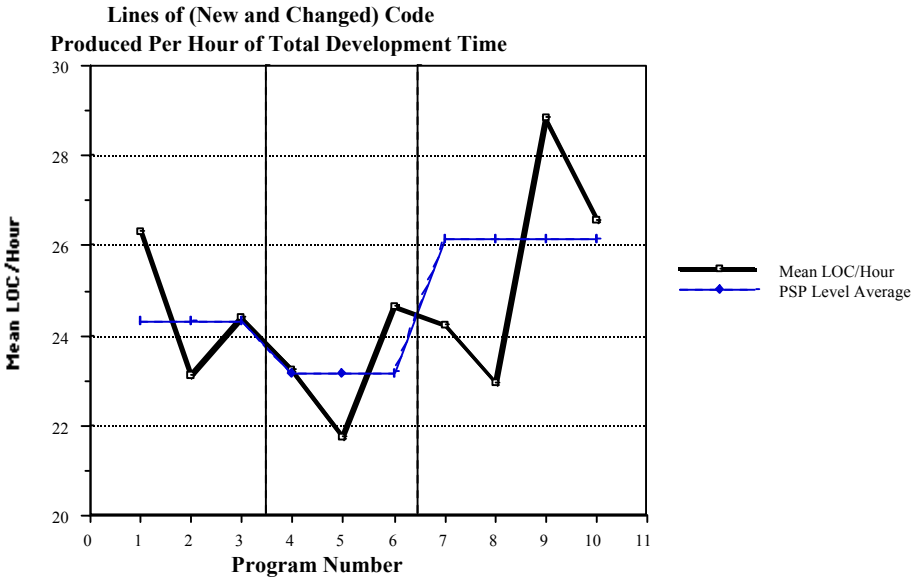
---

[SM] Personal Software Process, PSP, Team Software Process and TSP are service marks of Carnegie Mellon University.

tasks to ensure a quality product. Here comes the zinger. Productivity rates increase at the same time. Initially, student productivity rates decrease while learning estimating techniques; however, they increase dramatically after quality techniques are introduced. This would indicate that quality actually impacts productivity in a positive way.



PSP Class Defect Data



PSP Class Productivity Data

Industry reports (SEI) similar effects when implementing a quality-oriented process (PSP). One company saw their defect rates drop by orders of magnitudes when implementing quality-oriented software development techniques. Another showed drastic schedule savings. Their test times for medium-sized projects went from months to days, dramatically impacting their total development times.

Non-PSP Project	KLOC	Months Late	Acceptance Defects
1	24.6	9	N/A
2	20.8	4	168
3	19.9	3	21
4	13.4	8+	53
5	4.5	8+	25
PSP	22.9	1	1

Industry PSP Data

System Test Time before PSP Training	
Project A1 (15.8 KLOC)	1.5 months
Project C (19 requirements)	3 test cycles
Project D (30 requirements)	2 months
Project H (30 requirements)	2 months
System Test Time after PSP Training	
Project A2 (11.7 KLOC)	1.5 months
Project B (24 requirements)	5 days
Project E (2.3 KLOC)	2 days
Project F (1.4 KLOC)	4 days
Project G (6.2 KLOC)	4 days
Project I (13.3 KLOC)	2 days

Industry Project PSP Data

Team Software Process Data

The TSP is a new technology that is just now beginning to exit its prototyping stage. As a result, the SEI is just beginning to receive data from projects that have developed systems using TSP teams. The results are also dramatic. One TSP team of 15 engineers report integration test defects at a .2 defect per KLOC rate, system defects at a .4 defect per KLOC rate and field test defects at a .02 defects per KLOC rate. As a result of the low defect densities, field testing decreased from months per KLOC to merely days per KLOC.

	TSP Project		Non-TSP Project
	Plan	Actual	Actual
Size (KLOC)	110	89.9	9.5
Integration Test (Defects/KLOC)	1.0	0.2	
System Test (Defects/KLOC)	0.1	0.4	
Field Trial (Defects/KLOC)	0.0	0.02	5+
Field Testing Engineering Trial		2 weeks	3 months
Field Testing Acceptance Trial		3 weeks	6 months

A TSP Industrial Project

Another example of what a quality process can do for quality and productivity comes from the Department of Defense. Results from the TaskView project (which used TSP) at

Hill AFB, Utah indicate that quality and productivity can live on the same project. [Webb, Humphrey] Their productivity increased more than two times above their previous project. Prior to the TaskView project, typical total test days per KLOC ranged from 0.94 to 2.89 and system test defects per KLOC ranged from 2.21 to 4.78. The TaskView project showed dramatic improvements: total test days per KLOC was 0.22 and system defects per KLOC was 0.52.

So, how did this increase in quality with its resulting reduction in test time impact the overall schedule? The article states "Typical TIS projects require 22 percent of the project schedule (in days) to perform the final two TIS test phases. The TaskView project, using TSP, sharply reduced this percentage to 2.7 percent. This is a schedule savings of nearly 20 percent."

	Non-TSP Projects			TaskView
Size (in KLOC)	67.3	7.9	86.5	25.8
Test Days	63	23	92	6
Test Days/KLOC	0.94	2.89	1.06	0.23
Defects/KLOC • System Test • Acceptance Test	 2.21 N. A.	 4.78 1.89	 2.66 0.07	 0.54 0.15
Total Time for Integration and System Test	22% of Development Time			2.7%
Productivity (Relative)				• More than 2 times previous. • 16% over org. ave.

A TSP DoD Project

CONCLUSION

Both the informal discussion of the relationship between quality and productivity and the data presented indicate that quality and productivity can coexist in a software development. It stands to reason that, if one can reduce the amount of testing because there is a higher-quality product going into test, one should notice a decrease in total development time. In fact, the data strongly suggests that if an organization implements a quality-oriented process for software developments, software productivity will benefit. Both PSP and TSP data for class and industry would tend to support this conclusion. The bottom line: if one needs an increase in productivity, implement a quality program. If one simply wants an increase in product quality, implement a quality program. It will not negatively impact software productivity.

REFERENCES

Crosby, Philip, Quality Is Free, The Art of Making Quality Certain, McGraw-Hill, New York, Published November 1978.

Humphrey, Watts, "Planning II – Measuring Software Size," A Discipline for Software Engineering, Addison-Wesley, Published May 1995, Pg. 88.

Sommerville, Ian, The Software Process, Software Engineering, Fourth Edition, Addison-Wesley, New York, 1992, Page 9.

Webb, David and Humphrey, Watts, "Using the TSP on the TaskView Project," CrossTALK, The Journal of Defense Software Engineering, Volume 12, Number 2, February 1999, Pg. 3.