

# How the TSP Impacts the Top Line

Robert Musson  
Software Engineering Institute

*With the Team Software Process<sup>SM</sup> (TSP<sup>SM</sup>), developers are reporting significant productivity improvements. But what does this mean to the profitability of the corporation? How do these productivity improvements affect the balance sheet? This article compares the development costs associated with teams in a traditional test-based organization to those costs measured on TSP teams. It also presents product and quality data from several TSP projects at one industry organization.*

When thinking about productivity improvements, engineering management does not usually have precise data from which to determine the implications to the profit-and-loss statement of the organization. Deploying a new technology such as object oriented design is often done because everyone else in the industry is doing it, and the engineers want to use the latest hot technology. It is unclear how such process changes will impact the balance sheet; there is often no way to actually measure improvements. As a result, many organizations go year after year without knowing if meaningful improvement has occurred. Few organizations know the actual dollar cost of a thousand lines of code (KLOC); they think about head-count on a project, or person-years of effort, and so on. But when attrition is high and team turnover is not accurately measured, development costs become clouded.

This article will present a model of determining process effectiveness using the Team Software Process<sup>SM</sup> (TSP<sup>SM</sup>). The costs to produce a KLOC using the TSP will be compared to a more traditional process focused on testing in quality. Finally, total lifetime costs of the TSP will be compared to the test-based process.

## Software Development Cost Model

A simple model of a development process is used as the basis for comparison. The model will have five major phases: requirements (REQ), high-level design (HLD), implementation (IMPL), test (TEST), and release (REL). The detailed phase breakdown is shown in Table 1. These phases represent the work required of a software development project and whether or not a process actually produces these products. All systems have requirements; however, not all processes produce them.

To determine cost, two pieces of information are needed: how much time is spent in each phase and the cost of an engineer's time. The problem is that most

software organizations do not have this information available. Therefore, it is necessary to take known aspects of development and infer other information in order to determine how much time is spent in each activity.

For example, many organizations have measured the time to find and fix defects in various test activities, along with the expected numbers of defects found in these activities [1, 2]. Then this information can be used to determine the time spent in test. Using such data, Table 2 (see page 10), on percentage of time spent in phase, was created.

---

*"The cost to find and fix defects once the software has been released to the customer represents a significant portion of the total development cost ... By eliminating most of the time required to fix defects in a released product, an organization can focus its resources on new opportunities."*

---

The numbers for the traditional test-based process agree with studies that indicate testing can take two-thirds of the development effort [2, 3]. The TSP model agrees with data from one TSP organization [4]. The TSP costs are roughly 25 percent less than the costs required to produce one KLOC using the traditional

test-based approach. But this is only the cost to get the software into a state sufficient for first-customer use. In a traditional development process, once the software escapes from system testing, it undergoes extended periods of field trial and beta testing.

Defect repair costs typically soar up to 10 times the cost of integration and system testing. In fact, as much as half the cost to produce software is contained in repairing the software after the system test phase (termed release in this model, it includes beta trial, acceptance test, and customer use). Table 3 (see page 10) includes the costs associated with the field test and warranty periods. These numbers are based on the author's experience with several actual TSP development projects and are consistent with data from other organizations [4, 5].

Table 1: Cost Model Phases

Major Phase	Detailed Phase Breakdown
	Planning
Requirements (REQ)	Requirements
	System test plan
	REQ inspection
High-Level Design (HLD)	High-level design
	Integration test plan
	HLD inspection
Implementation (IMPL)	Detailed design (DLD)
	DLD review
	Test development
	DLD inspection
	Code
	Code review
	Compile
	Code inspection
	Unit test
TEST	Integration test
	System test
Release (REL)	Field trial
	Acceptance test
	Release

Traditional Process					Team Software Process				
Phase	% Total Effort	Detailed Phase Breakdown	% Total Effort	Total Hours	Phase	% Total Effort	Detailed Phase Breakdown	% Total Effort	Total Hours
	0.99%	Planning	0.99%	1.3		4.76%	Planning	4.76%	4.4
REQ	4.22%	Requirements	2.22%	2.9	REQ	19.87%	Requirements	9.93%	9.3
		System test plan	0.89%	1.1			System test plan	4.97%	4.6
		REQ inspection	1.11%	1.4			REQ inspection	4.97%	4.6
HLD	4.22%	High-level design	2.22%	2.9	HLD	18.06%	High-level design	9.03%	8.4
		Integration test plan	0.89%	1.1			Integration test plan	4.52%	4.2
		HLD inspection	1.11%	1.4			HLD inspection	4.52%	4.2
IMPL	24.63%	Detailed design	2.22%	2.9	IMPL	41.25%	Detailed design	8.21%	7.7
		DLD review	0.00%	0.0			DLD review	4.12%	3.8
		Test development	0.00%	0.0			Test development	4.11%	3.8
		DLD inspection	0.00%	0.0			DLD inspection	3.27%	3.1
		Code	11.65%	15.0			Code	7.43%	6.9
		Code review	0.00%	0.0			Code review	3.71%	3.5
		Compile	2.22%	2.9			Compile	1.26%	1.2
		Code inspection	0.78%	1.0			Code inspection	3.27%	3.1
		Unit test	7.77%	10.0			Unit test	5.87%	5.5
TEST	65.94%	Integration test	17.15%	22.1	TEST	16.05%	Integration test	7.20%	6.7
		System test	48.80%	62.8			System test	8.85%	8.3
<b>Sub-total</b>	<b>100%</b>		<b>100%</b>	<b>128.7</b>	<b>Sub-total</b>	<b>100%</b>		<b>100%</b>	<b>93.3</b>

Table 2: Cost to Market

The total lifetime cost for the TSP is less than 40 percent of the total cost of the traditional process. The total code produced by a traditional development organization is about four KLOC per engineer per year, assuming 1,000 useful hours per engineer year divided by 252 hours per KLOC. This number agrees

with those measured by various studies [5, 7, 8]. A TSP organization will produce just over 10 KLOC per engineer year, again assuming 1,000 useful hours divided by 95.6 hours per KLOC.

In the traditional process, early phase activities of requirements and architectural design tend to be cut short.

Table 3: Lifetime Software Development Costs

Traditional Process					Team Software Process				
Phase	% Total Effort	Detailed Phase Breakdown	% Total Effort	Total Hours	Phase	% Total Effort	Detailed Phase Breakdown	% Total Effort	Total Hours
	0.51%	Planning	0.51%	1.3		4.65%	Planning	4.65%	4.4
REQ	2.15%	Requirements	1.13%	2.9	REQ	19.40%	Requirements	9.70%	9.3
		System test plan	0.45%	1.1			System test plan	4.85%	4.6
		REQ inspection	0.57%	1.4			REQ inspection	4.85%	4.6
HLD	2.15%	High-level design	1.13%	2.9	HLD	17.64%	High-level design	8.82%	8.4
		Integration test plan	0.45%	1.1			Integration test plan	4.41%	4.2
		HLD inspection	0.57%	1.4			HLD inspection	4.41%	4.2
IMPL	12.58%	Detailed design	1.13%	2.9	IMPL	40.28%	Detailed design	8.02%	7.7
		DLD review	0.00%	0.0			DLD review	4.03%	3.8
		Test development	0.00%	0.0			Test development	4.01%	3.8
		DLD inspection	0.00%	0.0			DLD inspection	3.19%	3.1
		Code	5.95%	15.0			Code	7.25%	6.9
		Code review	0.00%	0.0			Code review	3.63%	3.5
		Compile	1.13%	2.9			Compile	1.23%	1.2
		Code inspection	0.40%	1.0			Code inspection	3.19%	3.1
		Unit test	3.97%	10.0			Unit test	5.73%	5.5
TEST	33.69%	Integration test	8.76%	22.1	TEST	15.67%	Integration test	7.03%	6.7
		System test	24.93%	62.8			System test	8.64%	8.3
<b>Sub-total</b>	<b>51.09%</b>		<b>51.09%</b>	<b>128.7</b>	<b>Sub-total</b>	<b>97.64%</b>		<b>97.64%</b>	<b>93.3</b>
REL	48.91%	All post TEST	48.91%	123.3	REL	2.36%	All post TEST	2.36%	2.3
<b>Total</b>	<b>100%</b>		<b>100%</b>	<b>252.0</b>	<b>Total</b>	<b>100%</b>		<b>100%</b>	<b>95.6</b>

Additionally, inspection activities take noticeably less time than in the TSP process for three reasons: They are eliminated under time pressure, engineers do not use sound data-based review methods, and work products are often not in a format that can be reviewed. The traditional process produces code very quickly, and then the real work begins in test. In contrast, the costs for the TSP tend to be front-loaded in early phases. More emphasis is placed on personal review and team inspections. This causes almost half of the effort to be expended before any code is even written. Figure 1 shows this graphically.

Notice that the phase investment of the TSP is relatively constant over the development cycle. Traditional processes have a higher cost in coding, and exponentially growing costs in test. The crossover point where the investment is equal in both processes occurs near the end of integration testing. At this point, we are indifferent to the two processes from an investment point of view; costs are roughly the same. Unfortunately, a traditional process has only just begun to pour dollars into the effort at the start of the system test.

### Payback Time

The TSP does not come for free. The training class requires two weeks of classroom time and several homework assignments. Most engineers complete training in 100 (plus or minus 20) hours, or about three weeks. Additionally, a TSP coach/Personal Software Process<sup>SM</sup> instructor is needed for every 50 to 100 engineers. That is the equivalent of another week per engineer for the whole organization to support the coach. Therefore, the incremental cost to deploy per engineer is about three weeks of fixed training cost, plus one week of variable cost per year. A payback graph of this appears in Figure 2.

In this model, this cost is paid back in 1.6 KLOC. This compares well with the 1,200 lines of code as reported by Teradyne [4].

### Conclusions

The total cost to get software to market using the TSP is much less than the cost of using a process focused on testing. However, this represents only a portion of the total lifetime cost to develop software. The cost to find and fix defects once the software has been released to the customer represents a significant portion of the total development cost. The elimination of this cost results in the bulk of

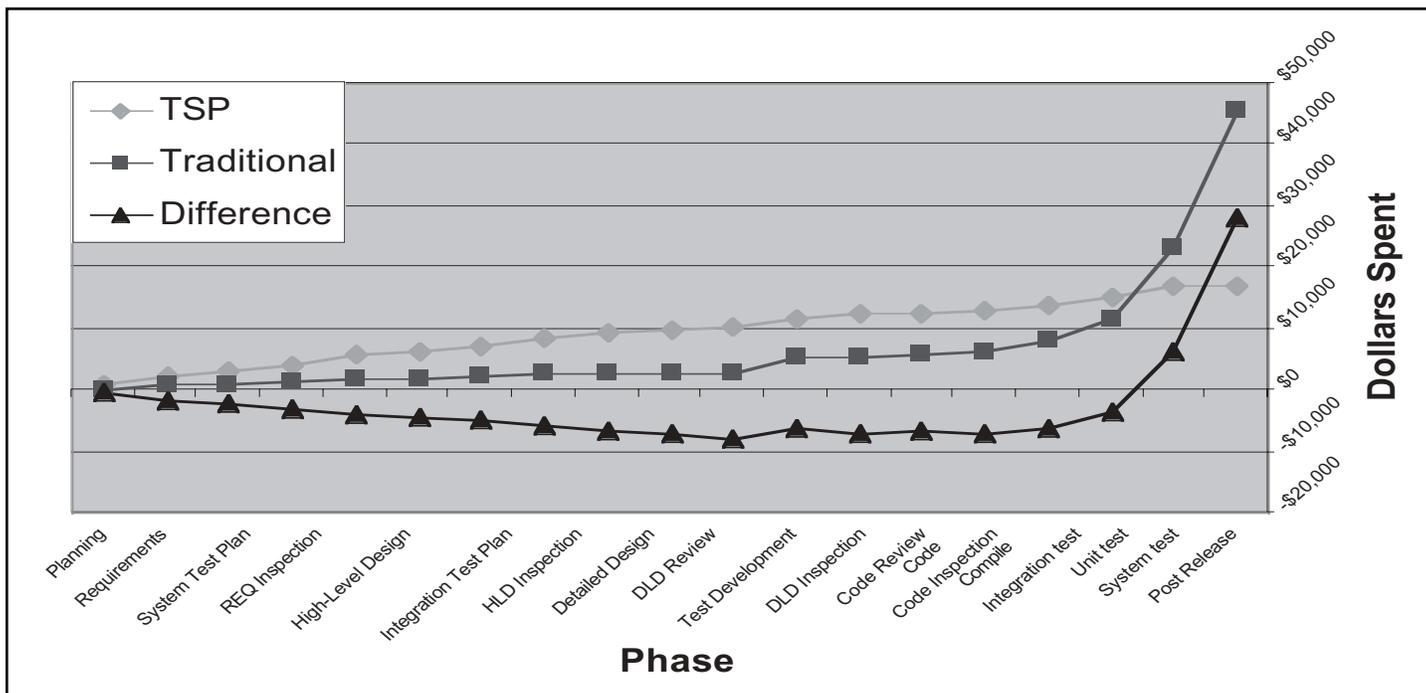


Figure 1: Cost per Phase

the savings for an organization using the TSP. By eliminating most of the time required to fix defects in a released product, an organization can focus its resources on new opportunities. The only investments required are training and a willingness to change. ♦

**References**

1. Davani-Chulani, Sunita. Modeling Software Defect Introduction. University of Southern California. Center for Software Engineering, 1998.
2. Cole, Oliver E. The Cost of Debugging. OC-Systems White Paper. Available at: <www.ocsystems.com>.
3. Colburn, Timothy R., James H. Fetzer, and Terry L. Rankin, ed. Program Verification: Fundamental Issues in Computer Science. Kluwer Academic Publishers, 1993.
4. Musson, Robert. "The Results of Using a Team Software Process." Presentation at the Software Engineering Symposium. Software Engineering Institute, Sept. 1999. Pittsburgh, PA.
5. Reifer, Donald J. "Let the Numbers Do the Talking." CrossTalk Mar. 2002: 4-9.
6. Hatton, Les. "How Do We Satisfy Customers in the Long Run?" ESCOM 2001.
7. Caron, Michael. "Cost Justifying a Test Coverage Analyzer Tool." Newsletter of the Boston Software Process Improvement Network. Nov. 1995.

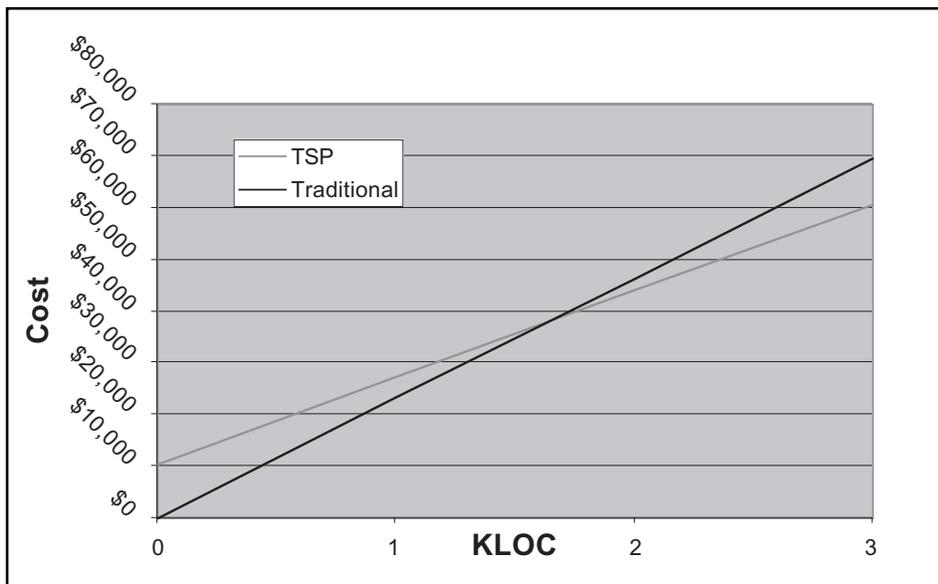


Figure 2: Payback on Training

**About the Author**



**Robert Musson** has more than 25 years of software experience as a development engineer and in various management positions. He spent 15 years at Teradyne helping bring to market a variety of products for the telecommunications industry. While there, he helped deploy the Team Software Process<sup>SM</sup> (TSP<sup>SM</sup>) to the first industry site. He was vice president of business strategy at a small start-up before becoming a member of the TSP

Initiative at the Software Engineering Institute. He has a master's degree in computer science from Illinois Institute of Technology and a master's degree in business administration from Northwestern University's Kellogg School of Management.

Software Engineering Institute  
 Carnegie Mellon University  
 Pittsburgh, PA 15213-3890  
 Phone: (412) 268-9130  
 Fax: (412) 268-5758  
 E-mail: ram@sei.cmu.edu