2 An Analysis of Code Defect Injection and Removal in PSP

Diego Vallespir, Universidad de la República
William Nichols

2.1 Introduction

A primary goal of software process improvement is to make software development more effective and efficient. Because defects require rework, one path to performance improvement is to quantitatively understand the role of defects in the process. We can then make informed decisions about preventing defect injection or committing the effort necessary to remove the injected defects. The PSP establishes a highly instrumented development process that includes a rigorous measurement framework for effort and defects. After examining a large amount of data generated during PSP classes, we can describe how many defects are injected during the PSP Code phase, the types of defects injected, when they are detected, and the effort required to remove them. We have found that even using a rigorous PSP development process, nearly a quarter of all defects injected will escape into unit test. Moreover, finding and removing defects in unit test required seven times as much effort as removing them in earlier phases. The purpose of this study is not to measure the effectiveness of PSP training, but rather to characterize the defects developers inject and must subsequently remove. By examining the characteristics of defect injections and escapes, we might teach developers how to define and improve their own processes and thus make the product development more effective and efficient.

Watts Humphrey describes PSP as "a self-improvement process that helps you to control, manage, and improve the way you work" [Humphrey 2005]. This process includes phases that you complete while building the software. For each phase, the engineer collects data on the time spent in the development phase and data about the defects injected and removed.

During the PSP course, the engineers build programs while they progressively learn the PSP. We analyzed eight exercises from this PSP course version. In this section, we present an analysis of defects injected during the Code phase of the last three PSP programs (6, 7, and 8). The engineers used the complete PSP when they built these programs.

We focused on defects injected during the Code phase because these specific data had not been studied before. Recently we made a similar analysis but focused on the defects injected during the Design phase of PSP [Vallespir 2011]. Previous studies did not have all the defect data, such as defect types and individual defect times; they had only summaries.

Our analysis of the complete data available from individual defect logs shows not only that the defects injected during Code are more expensive to remove in Test than in previous phases of the process, but also that they are easy to remove in the Code Review phase. The difference is striking: it costs seven times more to remove a defect in the PSP Unit Test than it does to remove a defect during code review.

To show this, we observed how defects injected during Code escaped into each subsequent phase of the PSP and how the cost to remove them was affected by defect type and phase. We describe the different defect types injected during Code and how these defect types compare with respect

to the find-and-fix time. From this analysis, we show that "syntax" type of defects are the most common Code phase defect (around 40% of all the defects), that personal code review is an effective removal activity, and that finding and fixing Code defects in the Code Review phase is substantially less expensive than removing them in the Test phase.

Other studies have examined software quality improvement using PSP [Paulk 2006, 2010; Wohlin 1998; Rombach 2008; Hayes 1997; Ferguson 1997]. In the context of PSP, quality is measured as defect density (defects/KLOC). Our study differs from the other studies in that we focused on code defects, considered the defect type, and did not consider defect density. Instead, we concentrated on the characteristics of the defects introduced in Code. Our findings resulted from analyses of the defect types injected, how they proceeded through the process until they were found and removed, and the cost of removal in subsequent development phases. In the literature, we do not know of any other study that has the characteristics of our research.

2.2 The Personal Software Process and the Collection of Data

For each software development phase, the PSP has scripts that help the software engineer follow the process correctly. The phases include Planning, Detailed Design, Detailed Design Review, Code, Code Review, Compile, Unit Test, and Post Mortem. For each phase, the engineer collects data on the time spent in the phase and the defects injected and removed. The defect data include the defect type, the time to find and fix the defect, the phase in which the defect was injected, and the phase in which it was removed. Figure 1 shows the guidance, phases, and data collection used with the PSP.

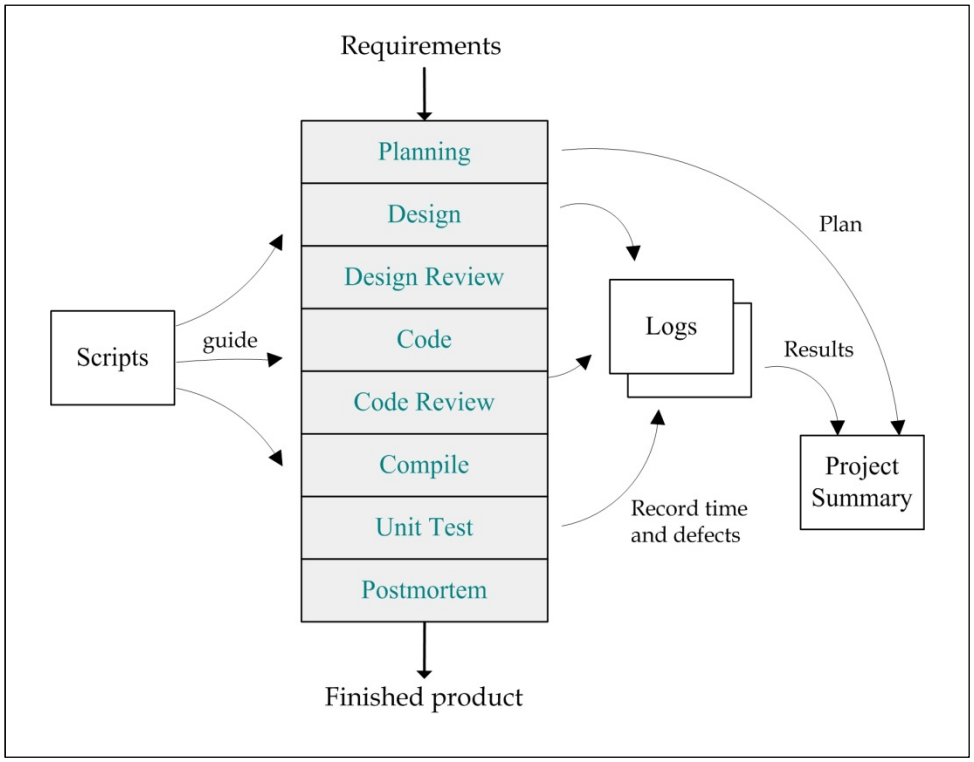

Figure 1: The PSP Phases, Scripts, Logs, and Project Summary

Some clarifications are needed to understand the measurement framework. The phases should not be confused with the activity being performed. Students are asked to write substantial amounts of code, on the scale of a small module, before proceeding through to reviewing, compiling, and testing. Once a block of code has passed into a phase, all time is logged in that phase, regardless of the developer activity. For example, a failure in test will require some coding and a compile, but the time will be logged in the "Unit Test" phase. If a personal review is performed prior to compiling, the compile can serve as an independent check of review effectiveness. We expect the compiler to remove the simple and inexpensive defects; however, if the review was effective the compile should be clean. When a defect is found, data recorded includes the phase of removal, the direct effort required to find and remove that defect, the phase in which it was injected, and the defect type and description.

The PSP defines 10 types of defects to be used during the course [Humphrey 2005; Chillarege 1996]. Table 1 presents these types of defects together with a brief description of which defects should be registered for each type.

Table 1: Defect Types in PSP

Defect Type	Possible Defects for the Type
Documentation	Comments, messages
Syntax	Spelling, punctuation, typos, instruction formats
Build/Package	Change management, library, version control
Assignment	Declaration, duplicate names, scope, limits
Interface	Procedure calls and references, I/O, user formats
Checking	Error messages, inadequate checks
Data	Structure, content
Function	Logic, pointers, loops, recursion, computation, function defects
System	Configuration, timing, memory
Environment	Design, compile, test, or other support system problems

The time to find and fix a defect is a direct measure of the time it takes to find it, correct it, and then verify that the correction made is right. In the Design Review and Code Review phases the time to find a defect is essentially zero, since finding a defect is direct in a review. However, the time to correct it and check that the correction is right depends on how complex the correction is. These variable costs are distinct from the predictable cost of performing a review. The point is that the variable cost of defects found in review can be directly measured and compared to similar costs in unit test.

On the other hand, both in the Compile and the Unit Test phases, finding a defect is an indirect activity. First, there will be a compilation error or a test case that fails. If that failure is taken as a starting point (compilation or test), what causes it (the defect) must be found in order to make the correction and verify if it is right.

During the PSP course, the engineers build programs while progressively learning PSP planning, development, and process assessment practices. For the first exercise, the engineer starts with a simple, defined process (the baseline process, called PSP 0); as the class progresses, new process steps and elements are added, from Estimation and Planning to Code Reviews, Design, and Design Review. As these elements are added, the process changes. The name of each process and which elements are added in each one are presented in Figure 2. The PSP 2.1 is the complete PSP process.

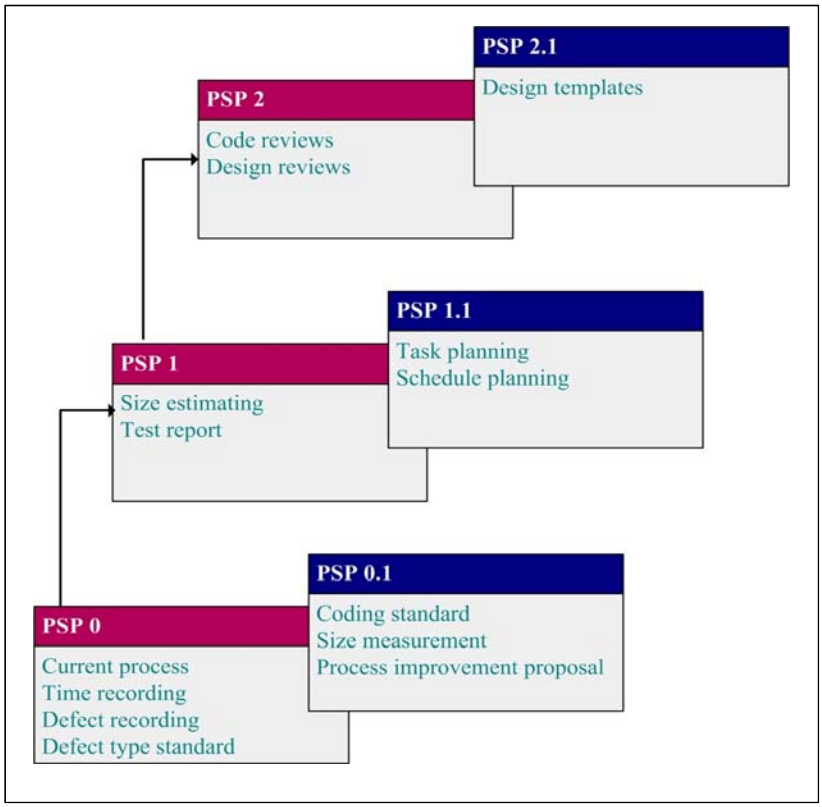

Figure 2: PSP Process Level Introduction During Course

In this section, we present an analysis of defects injected during the Code phase of the PSP, in programs 6, 7, and 8 (all developed using PSP 2.1, the complete PSP). In PSP 2.1, students conceptualize program designs prior to coding and record the design decisions using functional, logical, operational, and state templates. Students then perform a checklist-based personal review of the design to identify and remove design defects before beginning to write code. After coding, students perform a checklist-based personal review of the code. After the review they compile the code, and finally they make unit testing.

2.3 The Data Set

We used data from the eight-program version of the PSP course (PSP for Engineers I and II) taught between October 2005 and January 2010. These courses were taught by the SEI at Carnegie Mellon University or by SEI partners, including a number of different instructors in multiple countries.

This study is limited to considering only the final three programs of the PSP course (programs 6, 7, and 8). In these programs, the students apply the complete PSP process, using all process elements and techniques. Of course, not all of these techniques are necessarily applied well because the students are in a learning process.

This relative inexperience of the students constitutes a threat to the validity of this study, in the sense that different (and possibly better) results can be expected when the engineer continues using PSP in his or her working environment after taking the course. This is due to the fact that

the engineer continues to assimilate the techniques and the elements of the process after having finished learning the PSP.

We began with the 133 students who completed all programming exercises of the courses mentioned above. From this we made several cuts to remove errors and questionable data and to select the data most likely to have comparable design and coding characteristics.

Because of data errors, we removed data from three students. Johnson and Disney reviewed the quality of the PSP data [Johnson 1999]. Their analysis showed that 5% of the data were incorrect; however, many or most of those errors in their data were due to process calculations the students made. Process calculations are calculations made to obtain the values of certain derived measures the process uses to make estimates for the next program, such as the defects injected per hour in a certain phase or the alpha and beta parameters for a linear regression that relates the estimated size to the real size of the program.

Because our data were collected with direct entry into a Microsoft Access tool, which then performed all process calculations automatically, the amount of data removed (2.3%) is lower than the percentage reported by Johnson and Disney; however, this amount seems reasonable.

We next reduced the data set to separate programming languages with relatively common design and coding characteristics. As we analyze the code defects, it seems reasonable to consider only languages with similar characteristics that might affect code size, modularity, subroutine interfacing, and module logic. The students used a number of different program languages, as shown in Figure 3.

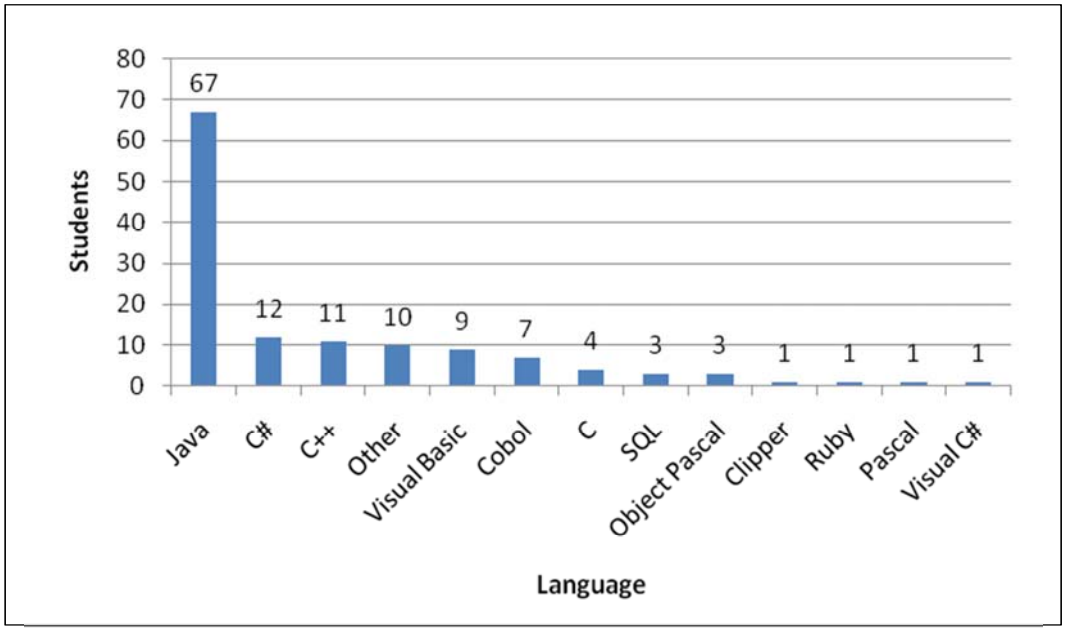

Figure 3: Quantity of Students by Program Languages

The most common language used was Java. To increase the size of the data set, we decided to include the data generated by students who used Java, C#, C++, and C. The languages in this group use similar syntax, subprogram, and data constructs. For the simple programs produced in the PSP course, we judged that these were most likely to have similar modularization, interface, and data design considerations. This cut reduced our data to 94 subjects.

Because our intent was to analyze defects, we removed from consideration any data for which defects were not recorded. From the 94 engineers remaining, two recorded no defects at all in the three programs considered. Our data set for this analysis was, therefore, reduced to 92 engineers. In the following sections we present the types of defects that were injected in the Code phase, when the defects were removed, and the effort required to find and fix these defects.

Our analysis included the defect injection and removal performance of individuals and the performance range of variation among them. It should be noted that this is different from analyzing the behavior of a team. That is, we wanted to characterize the work of individual programmers, which is why we calculated individual performance for each one of the subjects. After obtaining the performance of each subject, we computed the mean and spread of individual averages (as opposed to the global average for all defects). We calculated an estimate of the mean percentage, the 95% confidence interval for that mean (to characterize the standard error on the mean), and the standard deviation of the distribution, to characterize the spread among individuals. For these calculations, as previously mentioned, only programs 6, 7, and 8 of the PSP course were used in order to consider the complete PSP.

For this study we included the 92 engineers who recorded defects. In several cases, the number of engineers included varies, and in each of these cases the motive for varying is documented.

2.4 Where the Defects Are Injected

The first goal of our analysis was to better understand where defects were injected. We expected injections to be dominated by the Design and Code phases, because they are the construction phases in PSP. We began by explicitly documenting the phase injection percentages that occur during the PSP course.

For each PSP phase and for each individual, we calculated the percentage of defects injected. The distribution statistics are shown in Table 2**.**

Table 2: Mean Lower, Upper Confidence Interval Values and Standard Deviation of the Percentage of Defects Injected by Phase

	DLD	DLDR	Code	CR	Comp	UT
Mean	46.4	0.4	52.4	0.3	0.03	0.5
Lower	40.8	0.2	46.7	0.0	0.00	0.2
Upper	52.0	0.7	58.1	0.7	0.09	0.9
Std. Dev.	27.2	1.7	27.4	1.8	0.30	1.8

The Design and Code phases have similar injection percentages both on average and in the spread. Their mean of the percentage of defects injected is near 50% with lower and upper CI bounds between 40% and 58%. Both standard deviations are around 27%. So, in the average of this population, roughly half of the defects were injected in the Design phase and the other half were injected in the Code phase. On average, the defect potential of these phases appears to be very similar. The standard deviation shows, however, that the variability between individuals is substantial. Nonetheless, as we expected, in the average almost 99% of the defects were injected in the Design and Code phases with only around 1% of the defects injected in the other phases.

The Design Review, Code Review, Compile, and Unit Test phases also have similar average defect potentials. The average in all these cases is less than 0.5% and their standard deviations are small, the largest being 1.8% in Code Review and Unit Testing. This shows that during verification activities in PSP, the percentage of defects injected is low but not zero. From time to time, developers inject defects while correcting other defects. We will study these secondary injections in a later study.

The variability between individuals and the similarity between the Code and Design phase is also presented in Figure 4. Note that the range in both phases is from 0% to 100% (all possible values). The 25th percentile is 26.34 for Design and 35.78 for Code, the median is 45.80 for Design and 52.08 for Code, and the 75th percentile is 64.22 for Design and 71.56 for Code.

Despite a high variability between individuals, this analysis shows that the great majority of defects are injected in the Design and Code phases. Slightly more defects are injected during Code than during Design, but the difference is not statistically significant. We could, therefore, focus on the defects injected in the Design and Code phases. In this article, we discuss only the defects injected in the Code phase.

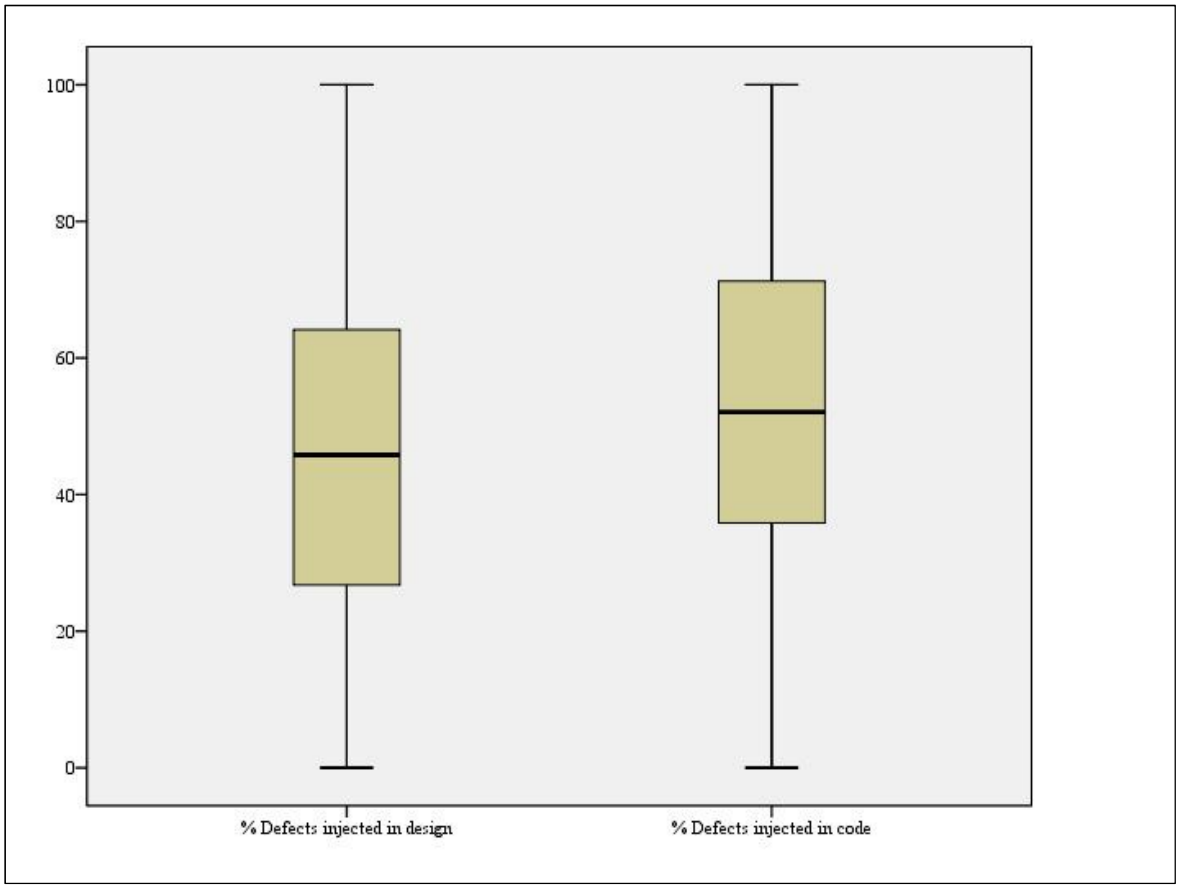

Figure 4: Percentage of Defects Injected by Phase (Box and Whisker Chart)

2.5 Analysis of CODE Defects

From the 92 engineers in our data set there were four whose records of injected defects (injected during Code) were uncertain regarding their correctness; they were therefore dismissed for this analysis. Also, eight engineers did not record defects in the Code phase, so they were dismissed, as well.

Our data set for analysis of the code defects was, therefore, reduced to 80 engineers. In the following sections we discuss the types of defects that are injected in the Code phase, when those defects are removed, and the effort required to find and fix the defects.

2.5.1 Defect Types Injected During the Code Phase

To improve the detection of code defects, we first wanted to know which types of defects were injected during the Code phase. Table 3 shows the mean of the percentage of the different defect types injected. It also presents the lower and upper bound of the 95% confidence interval for the mean (a measure of the standard error) and the standard deviation of the distribution.

None of the engineers registered system type defects injected during the Code phase. The Mean D. line presents what was found in our previous work of analysis of the defects injected during the Design phase, so these results could be comparable.

Table 3: Percentage of Defect Types Injected During Code

	Documentation	Syntax	Build Package	Assignment	Interface	Checking	Data	Function	System	Environment
Mean	3.8	40.3	0.6	14.0	5.5	2.7	5.8	26.4	0	0.9
Mean D.	6.9	6.0	0.1	12.6	10.0	4.6	9.8	46.6	0.2	3.1
Lower	1.5	33.7	0.0	9.9	3.1	1.0	3.1	19.9	0	0.0
Upper	6.0	46.9	1.1	18.1	8.0	4.4	8.6	32.8	0	1.7
Std. Dev.	10.1	29.5	2.5	18.4	11.1	7.4	12.4	29.1	0	3.9

When seeking improvement opportunities, a Pareto sort can be used to identify the most frequent or most expensive types. These types can then be the focus for future improvement efforts. For the following analysis we sorted defects by frequency, and then segmented the defect types into three categories of increasing frequency. The first grouping is “very few” defects of this type. In the “very few” category we found system, build/package, and environment types of defects. In our previous work, in which we studied the defects injected in the Design phase, we found within this category the system and build/package defect types, but not the environment type of defect. Considering this work and the previous work, it is clear that, during the PSP course, the build/package and system types of defects were seldom injected. This may be due to the PSP course exercises rather than the PSP. Because the exercises are small, take only a few hours, contain few components, and make few external library references, build packages are usually

quite simple. We expect to find more defects of these types in the TSP [Humphrey 2000, 2006] in industrial-scale projects.

A second grouping is “few defects”; most of the other defect types (all except syntax and function types) are in this category. The percentage of defects in this category ranged from 2.7% to 14.0%. In our previous work, both syntax and environment defect types were in this category. It is reasonable that, when analyzing the design defects mentioned, we find few syntax defects and that the percentage of these defects in relation to the rest of the other types of defects increases when the defects injected during the Code phase are analyzed. It is natural that when coding, more syntax defects are made than when designing, even if the design contains pseudocode, as in the case of the PSP.

The third and final grouping, “many defects,” includes the syntax and function types of defects. The syntax defects injected during Code are around 40% of the total defects and the function defects are around 26%. Approximately two out of three injected defects during the Code phase are of one of these two types.

As mentioned earlier, one out of four (26.4%) defects injected during the Code phase is a function type of defect. This is an opportunity for improvement for the PSP, since this type of defect should be injected (and as far as possible removed) before reaching Code phase. The fact that there is such a large percentage of this type of defect injected indicates problems in the Design and Design Review phases. PSP incorporates a detailed pseudocode in the Design phase using the Logic Template. Therefore, it is in this phase that the function type defects should be injected, and then they should be removed in the Design Review phase.

The lower, upper, and standard deviation data show again the high variability between individuals. This can also be observed in Figure 5; the box and whisker chart shows many observations as outliers. The high variability among individuals is repeated in every analysis conducted, both in this work and in the previous work, which studied the defects injected in the Design phase. A detailed analysis of developer variability is beyond the scope of this paper.

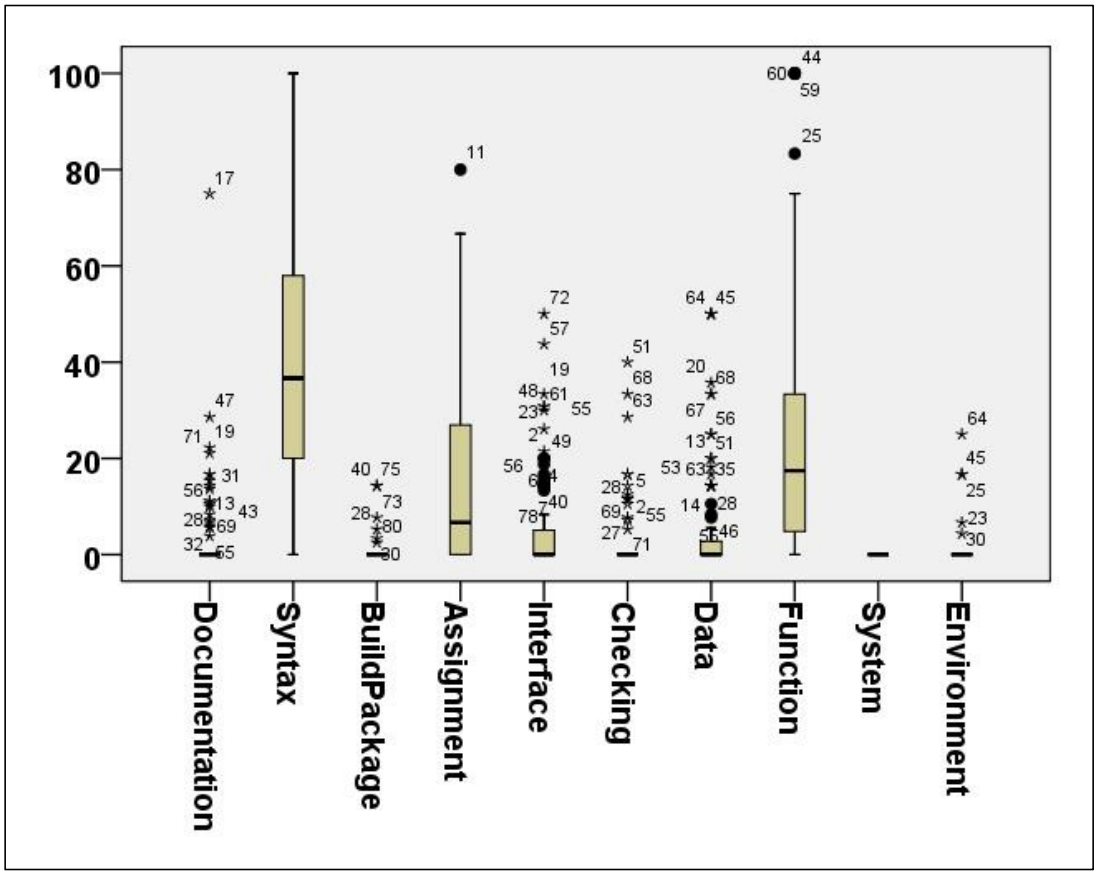

Figure 5: Box and Whisker Plot of the Percentage of Defects Injected During Code

2.5.2 When Are the Defects Injected During Code Removed?

Our analysis indicated the subsequent phases during which the code defects were removed. While our data set was large, it remained too small for us to examine the removals based on defect type. Still, for each engineer who injected defects in the Code phase, we identified the phases in which the engineer found the defects; then, for every phase, we determined the percentage of the defects that were found in that phase.

Table 4 shows the mean (with 95% confidence interval) and standard deviation for the different phases. As previously shown, the standard deviation was high, indicating the high variability between individuals. From this we learned that, on average, 62% of the defects injected during Code were found in the Code Review phase.

Table 4: Phases Where the Code Defects are Found (Percentage)

	CODE DEFECTS		
	CR	**Comp**	**UT**
Mean	62.0	16.6	21.4
Lower	55.0	11.7	15.4
Upper	69.0	21.6	27.3
Standard Deviation	31.3	22.4	26.9

In our previous analysis, we found that around 50% of the defects injected during the Design phase were detected in the Design Review phase. This indicates that for the defects injected in both the Design and Code phases, the following Review phases are highly effective.

On the other hand, in the previous study we also found that around 25% of the defects injected in the Design phase are detected only in Unit Test. This happens in 21.4% of the cases, based on our analysis of defects injected in the Code phase. This indicates that a relatively high percentage of defects manage to escape from the different detection phases and reach Unit Test.

We also know, of course, that not all the defects that escape into Unit Test are found in Unit Test. This phase will not have a 100% yield. (That is, we will not find all the defects that are in a given unit when it arrives at Unit Test.) Therefore the percentage of defects found in each of these phases is smaller than reported, while the actual percentage of escapes into Unit Test is a lower limit. An estimate or measurement of the Unit Test yield will be necessary to revise these phase estimates.

Figure 6 shows the box and whisker charts displaying the percentage of defects found in the different phases. Figure 6also shows the high variability between individuals in the percentage of defects found during Code Review, Compile, and Unit Test phases. This variability among individuals was also found in the previous study.

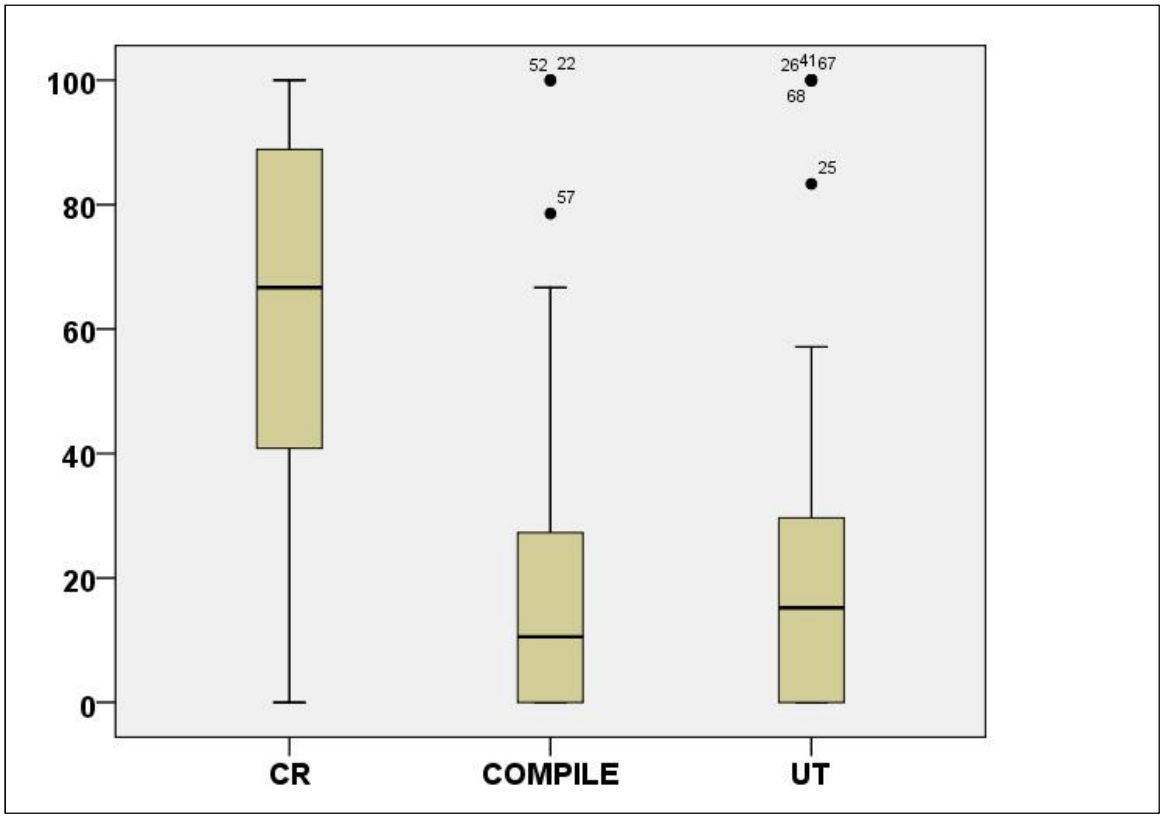

Figure 6: In Which Phase Are the Code Defects Found? – Variability Between Individuals

2.5.3 Cost to Remove the Defects Injected in Code

What is the cost and variation in cost (in minutes) to find and fix the defects that are injected during Code? To determine this, we first analyze the differences in cost, segmented by the removal phase. Second, we study the differences in cost segmented by defect type.

It also would be interesting to segment and analyze both the removal phase and the defect type jointly. Unfortunately, because of limited sample size after a two-dimensional segmentation, we cannot perform that analysis with statistical significance.

2.5.3.1 Phase Removal Cost

What is the cost, in each removal phase, to find and fix a defect injected in Code? Code defects can be removed in PSP in the Code Review (CR), Compile, and Unit Test phases. For each engineer, we calculated the average task time of removing a design defect in each of the different phases. Because some engineers did not remove code defects in one or more phases, our sample size varied by phase. We had data from 72 engineers for Code Review, 44 for Compile, and 51 for Unit Test.

Table 5 shows the mean, lower, and upper 95% confidence intervals and the standard deviation for the find-and-fix time (in minutes) for code defects in each of the studied phases.

Table 5: Cost of Find and Fix Defects Injected in Design Segmented by Phase Removed

	CODE DEFECTS		
	CR	**Com**	**UT**
Mean	1.9	1.5	14.4
Lower	1.5	1.1	9.8
Upper	2.3	1.9	19.0
Standard Deviation	1.9	1.3	16.4

As we expected, the average cost of finding code defects during Unit Test is much higher than in the other phases, by a factor of seven. We are not stating here that the cost of finding and fixing a particular code defect during Unit Test is seven times higher than finding and fixing the same particular code defect in Code Review or Compile. We are stating that with the use of PSP, the code defects that are removed during Unit Test cost seven times more than the ones that were removed in Code Review and Compile (these are different defects).

In our previous study we also found that the Design injection defects find-and-fix times in Design Review and Code Review are a factor of five smaller than the find-and-fix times in Unit Test. We also found that the defects injected in Design and removed in Unit Test have an average find-and-fix time of 23 minutes. Considering the two analyses, these are the defects that are most costly to remove. Defects injected in Code and removed in Unit Test follow with an average of 14.4 minutes. Testing, even at the unit test level, is consistently a more expensive defect removal activity than alternative verification activities.

We also found high variability among individual engineers. This variability can be seen in the box and whisker chart in Figure 7. We tested for normal distribution after log transformation of find-and-fix times for Code Review, Compile, and Unit Test. Only Unit Test is consistent ($p > 0.05$

using Shapiro-Wilk test) with a log-normal distribution. The test for normality is primarily useful to verify that we can apply regression analysis to the transformed data; however, understanding the distribution also helped to characterize the asymmetry and long-tailed nature of the variation. The log-normality confirmed our intuition that some defects found in Unit Test required far more than the average effort to fix, making Unit Test times highly variable. We observe that both the mean and variation of rework cost in the Code Review and Compile phases were significantly lower than Unit Test in both the statistical sense and the practical sense.

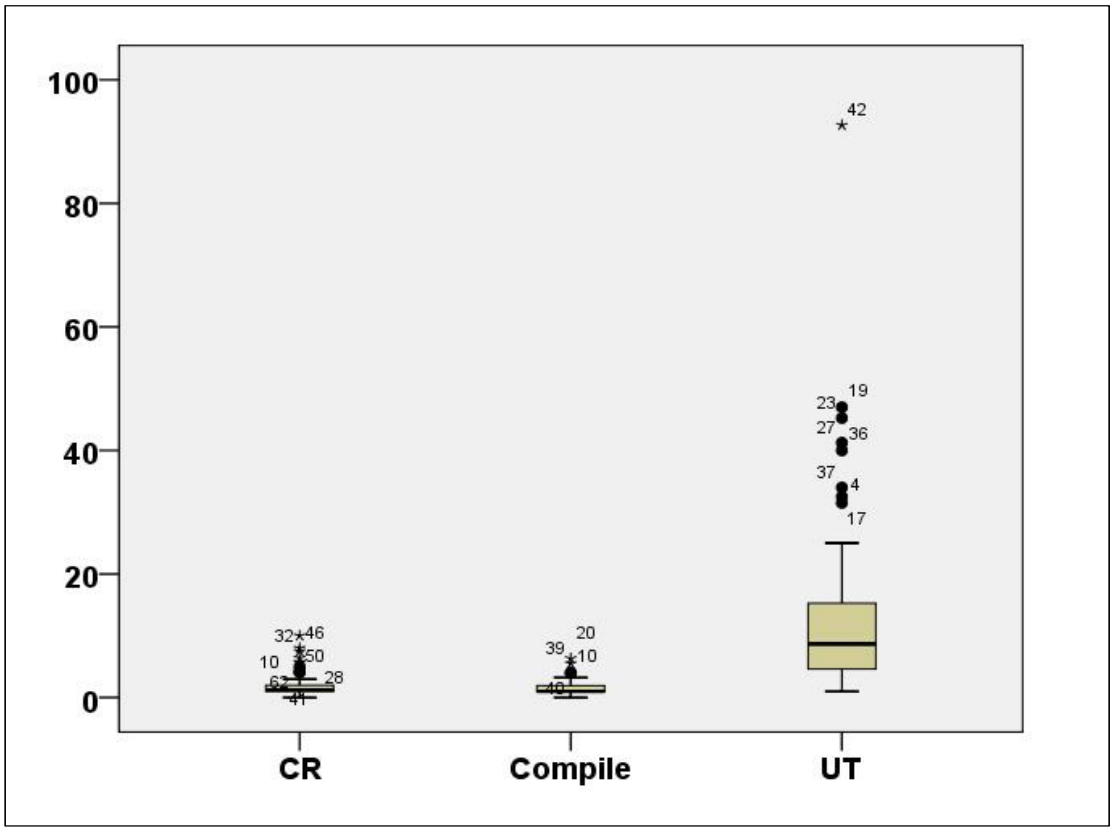

Figure 7: Box and Whisker Plot of the Cost to Find and Fix a Code Defect Segmented by Phase Removed

2.5.3.2 Defect Removal by Type

What is the find-and-fix cost, per defect type, of defects injected during Code? We did not have enough data for a statistically significant analysis of the cost of removing the build/package, checking, system, and environment types of defects. However, we were able to analyze the remaining defect types.

Table 6 presents the mean, lower, and upper 95% confidence interval and the standard deviation categorized by defect type then sorted by the find-and-fix cost of code defects. For prioritization, the cost, in minutes, for "find-and-fix" fell into three groups:

1. A group that has a mean around 2-3 minutes: documentation, syntax, assignment, and interface defects
2. A group, composed only of function defects, that has a mean around 9 minutes

3. A group, composed only of data defects, that has a mean around 12 minutes

Function type defects (injected during Code phase) take three times longer to find and fix than documentation, syntax, assignment, and interface defects. Data type defects take four times as much time.

Table 6: Cost of Find-and-Fix Defects Injected in Code Segmented by Defect Type

	Documentation	Syntax	Assignment	Interface	Data	Function
Mean	3.4	1.9	2.7	2.3	12.2	9.4
Lower	1.3	1.4	1.8	1.4	0.0	6.8
Upper	5.4	2.3	3.7	3.3	27.2	12.1
Standard Deviation	4.2	2.0	3.1	2.2	32.9	10.7

As in the other cases, the variation among individual developers was high. This can be seen using the standard deviation, as well as the box and whisker chart that is presented in Figure 8.

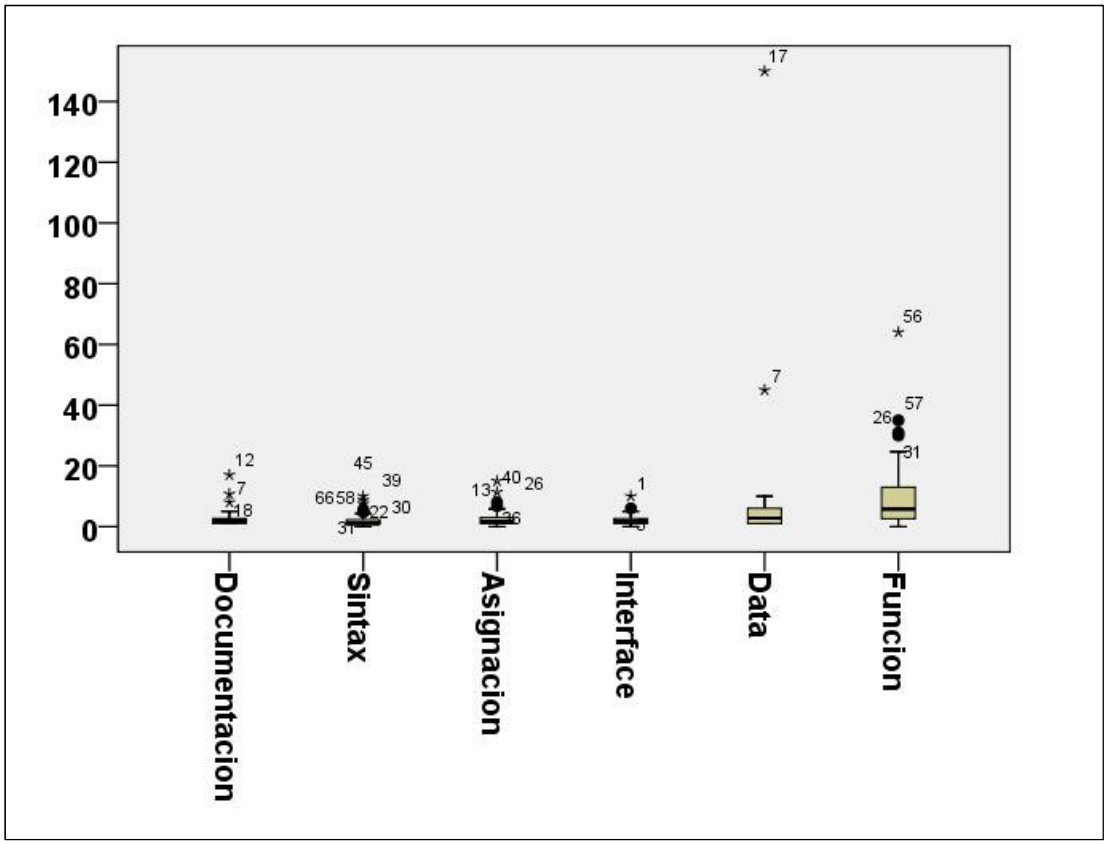

Figure 8: Box and Whisker Plot of the Cost to Find and Fix a Defect Segmented by Defect Type

2.6 Limitations of this Work

There are several considerations that limit the ability to generalize this work: the limited life cycle of the PSP course, the lack of a production environment, that students are still learning process, the students perform the defect categorization, and the nature of the exercises. Because the PSP life cycle begins in Detailed Design and ends in Unit Test, we do not observe all types of defects and specifically do not observe requirements defects or those that would be found in the late testing such as Integration Test, System Test, and Acceptance Test. This also implies that finds in Unit Test are only a lower estimate of the actual escapes into Unit Test. Defects such as build and environment or requirements injections are not considered.

The second consideration is that the PSP exercises do not build production code. Code is not intended to be "bullet proofed" or production ready. This is most likely to affect the rigor of Unit Test. Students often run only the minimum tests specified. This will likely lead to fewer defects being found and higher overall development rates. For example, coding rates are typically much higher than those found in industry. Also excluded is the production practice of peer inspections.

A third consideration is that, students using PSP are still learning techniques of design and personal review. The results after gaining experience may differ from those found during this course.

A fourth consideration is that precision of the student categorization of defect types has not been precisely measured. That is, students are learning how to categorize defects and should receive

guidance from the instructor. Nonetheless, there will certainly be some inconsistency among the students in how specific defects are categorized.

Finally, the problems, though modestly challenging, do not represent a broad range of development problems.

2.7 Conclusions and Future Work

In this analysis, we considered the work of 92 software engineers who, during PSP course work, developed programs in the Java, C, C#, or C++ programming languages. In each of our analyses, we observed a high variation in range of performance among individuals; we show this variability using standard deviation and box and whisker charts to display the median, quartiles, and range.

After considering this variation, we focused our analysis on the defects injected during Code. Our analysis showed that most common code defects (40%) are of syntax type. This type of defect is the cheapest to find and fix (1.9 minutes). The types of defects injected in Code that are most expensive to correct are the data (12.2 minutes) and function (9.4 minutes) defect types.

In addition, the analysis showed that build/package, systems, and environment defects were seldom injected in the Code phase. We interpreted this as a consequence of the small programs developed during the course, rather than as a characteristic of PSP as a development discipline.

We found that defects were injected roughly equally in the Design and Code phases; that is, around half of the defects were injected in code. 62% of the code defects were found early through appraisal during the Code Review phase. However, around 21% were discovered during Unit Test, where mean defect find-and-fix time is almost seven times greater than find-and-fix time in review.

While this analysis provided insights into the injection and removal profile of code defects with greater specificity than previously possible, a larger data set would allow us to consider more detail, such as the costs of defects discriminated by defect type in addition to removal phase. A more complete analysis may enable us to analyze improvement opportunities to achieve better process yields. In future analysis, we will examine the relationship between Design and Code activities and the defects found in the downstream phases.

2.8 Author Biographies

Diego Vallespir
Assistant Professor
School of Engineering, Universidad de la República

Diego Vallespir is an assistant professor at the Engineering School at the Universidad de la República (UdelaR), where he is also the director of the Informatics Professional Postgraduate Center and the Software Engineering Research Group. He is also a member of the Organization Committee of the Software and Systems Process Improvement Network in Uruguay (SPIN Uruguay).

Vallespir holds Engineer, Master Science, and PhD titles in Computer Science, all from UdelaR. He has had several articles published in international conferences. His main research topics are empirical software engineering, software process, and software testing.

William Nichols

Bill Nichols joined the Software Engineering Institute (SEI) in 2006 as a senior member of the technical staff and serves as a PSP instructor and TSP coach with the Team Software Process (TSP) Program. Prior to joining the SEI, Nichols led a software development team at the Bettis Laboratory near Pittsburgh, Pennsylvania, where he had been developing and maintaining nuclear engineering and scientific software for 14 years. His publication topics include the interaction patterns on software development teams, design and performance of a physics data acquisition system, analysis and results from a particle physics experiment, and algorithm development for use in neutron diffusion programs. He has a doctorate in physics from Carnegie Mellon University.

2.9 References/Bibliography

[Chillarege 1996]
Chillarege, R. "Orthogonal Defect Classification," 359–400. *Handbook of Software Reliability Engineering*. Edited by M.R. Lyu. McGraw-Hill Book Company, 1996.

[Ferguson 1997]
Ferguson, Pat; Humphrey, Watts S.; Khajenoori, Soheil; Macke, Susan; & Matvya, Annette. "Results of Applying the Personal Software Process." *Computer 30*, 5, (May 1997): 24–31.

[Hayes 1997]
Hayes, Will & Over, James. *The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers* (CMU/SEI-97-TR-001). Software Engineering Institute, Carnegie Mellon University, 1997. http://www.sei.cmu.edu/library/abstracts/reports/97tr001.cfm

[Humphrey 2006]
Humphrey, Watts S. *TSP: Coaching Development Teams*. Addison-Wesley, 2006. http://www.sei.cmu.edu/library/abstracts/books/201731134.cfm

[Humphrey 2005]
Humphrey, Watts S. *PSP: A Self-Improvement Process for Software Engineers*. Addison-Wesley Professional, 2005. http://www.sei.cmu.edu/library/abstracts/books/0321305493.cfm

[Humphrey 2000]
Humphrey, Watts S. *The Team Software Process* (CMU/SEI-2001-TR-023). Software Engineering Institute, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/library/abstracts/reports/87tr023.cfm

[Johnson 1999]
Johnson, Philip M. & Disney, Anne M. "A Critical Analysis of PSP Data Quality: Results from a Case Study." *Empirical Software Engineering 4*, 4 (March 1999): 317–349.

[Paulk 2010]
Paulk, Mark C. “The Impact of Process Discipline on Personal Software Quality and Productivity.” *Software Quality Professional 12*, 2 (March 2010): 15–19.

[Paulk 2006]
Paulk, Mark C. “Factors Affecting Personal Software Quality.” *CrossTalk: The Journal of Defense Software Engineering 19*, 3 (March 2006): 9–13.

[Rombach 2008]
Rombach, Dieter; Munch, Jurgen; Ocampo, Alexis; Humphrey, Watts S.; & Burton, Dan. “Teaching Disciplined Software Development.” *The Journal of Systems and Software 81*, 5 (2008): 747–763.

[Vallespir 2011]
Vallespir, Diego & Nichols, William. “Analysis of Design Defects Injection and Removal in PSP,” 19–25. *Proceedings of the TSP Symposium 2011: A dedication to excellence*. Atlanta, GA, September 2011. Software Engineering Institute, Carnegie Mellon University, 2011.

[Wohlin 1998]
Wohlin, C. & Wesslen, A. “Understanding software defect detection in the Personal Software Process,” 49–58. *Proceedings of the Ninth International Symposium on Software Reliability Engineering*, Paderborn, Germany, November 1998. IEEE, 1998.