Analysis of Design Defect Injection and Removal in PSP

Diego Vallespir, Universidad de la Republic
William Nichols, Software Engineering Institute

1.1 INTRODUCTION

A primary goal of software process improvement is to make software development more effective and efficient. One way of doing that is to understand the role of defects in the process and make informed decisions about avoiding defect creation or committing the effort necessary to find and fix the defects that escape development phases. Through examination of a large amount of data generated during Personal Software Process (PSP) classes, we can show how many defects are injected during design, what types of defects are injected, and how they are detected and removed in later development phases. We can use this information to teach developers how to define and improve their own processes, and thus make the product development more effective and efficient.

"The Personal Software Process (PSP) is a self-improvement process that helps you to control, manage, and improve the way you work" [Humphrey 05]. This process includes phases that you complete while building the software: plan, 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 development phase and data about 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.

During the Personal Software Process 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 PSP0); as the class progresses, new process steps and elements are added, from estimation and planning to code reviews, to design, and design review.

In this article, we present an analysis of defects injected during the design phase of the PSP programs 6, 7, and 8 (all developed using PSP2.1). In PSP2.1, students conceptualize program design 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.

In this analysis, we focused on defects injected during the design phase because these data had not been specifically studied before. 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 design are the most expensive to remove in test but also these are easy to remove in the review phases. The difference is striking: it costs five times more to remove a defect in test than it does to remove that same defect during review.

To show this, we observed how defects injected during design escaped into each subsequent phase of the PSP and how the cost to remove was affected by defect type and phase. We describe the different defects types injected during design and how these defect types compare with respect to the "find and fix" time. From this analysis, we show that "function" defects are the most common design phase defect, that personal design review is an effective removal activity, and that finding and fixing design defects in review is substantially less expensive than removal in test.

Some other articles study software quality improvement using PSP [Paulk 10] [Wholin 98] [Rombach 08] [Paulk 06] [Hayes 97] [Ferguson 97]. In the context of PSP, quality is measured as defect density (defects/KLOC). Our study differs from these others in that we focus on design defects, consider the defect type, and do not consider defect density. Instead, we focus on the characteristics of the defects introduced in design. Our findings resulted from analyses of the defect types injected, how they proceeded through the process until they were found and removed, and cost of removal in subsequent development phases.

1.2 THE DATA SET

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

This study is limited to only consider the final three programs of the 2006 version 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. Specifically, these exercises include the use of design templates and design reviews. Of course, not all of these techniques are necessarily applied well because the students are in a learning process.

We began with the 133 students who completed all programming exercises. 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 was incorrect; however, many or most of those errors in their data were due to calculations the students made. Because our data were collected with direct entry into a MS Access tool, which then performed all calculations automatically, the lower amount (2.3%) of data removed is lower than the percentage reported by Johnson and Disney but seems reasonable.

We next reduced the data set to separate programming languages with more common design and coding characteristics. As we analyze the design 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 1.

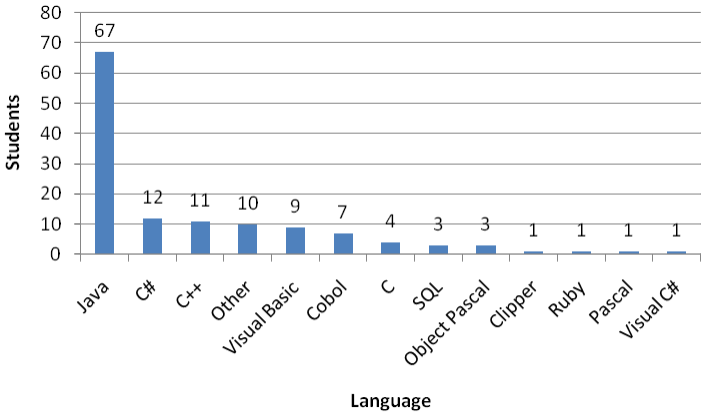

Figure 1: Quantity of students by program languages

The most common language used was Java. To increase the data set size, we decided to include the data generated by students who used Java, C#, C++, and C. This group of languages uses 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 that generated by the programming efforts of 94 students.

Because our intent was to analyze defects injected in the design phase, we removed from consideration any data for which design defects were not recorded. From the 94 data sets remaining, two recorded no defects and 11 recorded no defects injected during design. Our data set for this analysis was, therefore, reduced to 92 engineers or 83 engineers depending upon the specific analysis performed. In the following sections we present the types of defects that were injected in the design phase, when the defects were removed, and the effort required to find and fix these defects.

1.3 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 of course, because they are the construction phases in PSP. We began by explicitly documenting the phase injection percentages.

This analysis studied the defect injection and removal performance of individuals and the performance variation among them. We included the 92 engineers who recorded defects. We began by computing the phase data for each individual and then computed the following statistics of the distribution of individuals:

- an estimate of the mean percentage of defects injected by phase
- the 95% confidence interval for that mean (to characterize the standard error on the mean)
- the standard deviation of the distribution to characterize the spread among individuals

For each phase and for each individual, we calculated the percentage of defects injected in each PSP phase. The distribution statistics are shown in Table 1.

	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	0.0	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	0.3	1.8

Table 1: Mean lower, upper confidence interval values and std. dev. of the % of defects injected by phase

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 confidence interval bounds between 40% and 58%. Both standard deviations are around 27% with. So, in the average of this population, roughly half of the defects were injected in the design phase and the other half of the defects 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 2. 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 design phase.

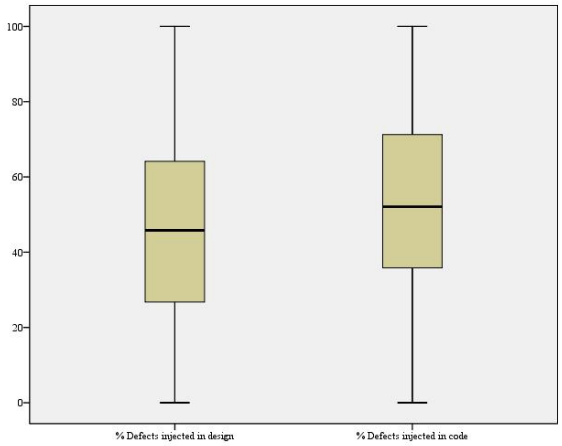

Figure 2: Percentage of defects injected by phase (box and whisker chart)

1.4 ANALYSIS OF DESIGN DEFECTS

From the 94 engineers in our data set there were 11 who recorded no injected defects during design. Our data set for analysis of the design defects was, therefore, reduced to 83 engineers. In the following sections we discuss the types of defects that are injected in the design phase, when those defects are removed, and the effort required to find and fix the defects.

1.4.1 Defect types Injected during Design

To improve the detection of design defects we first wanted to know which types of defects were injected during the design phase. Table 2 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.

We divided these defect types into three categories. The first is "almost not defects of this type." In this category we found system and build/package defects. It is clear that during the PSP course these types of defects were almost never injected. This may be due to the PSP course exercises rather than the PSP. Because the exercises are small, taking only a few hours, and 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 Team Software Process in industrial scale projects. A second category is "few defects;" most of the other defect types (all except Function type) are in this category. The percentage of defects in this category ranged from 3.1% to 12.6%.

The last category, "many defects," includes only one type of defect: function. The great majority of defects injected during design were of the Function type. This type of defect was almost half of all the defects injected during Design. This is an observation familiar to PSP instructors, but not previously reported for a sizable data set.

The lower, upper, and standard deviation data show again the high variability between individuals. This can also be observed in Figure 3; the box and whisker chart shows many observations as outliers. Also, it can be seen that the function defect type goes from 0% to 100% and that the 25 percentile is 21% and the 75 percentile is 70%.

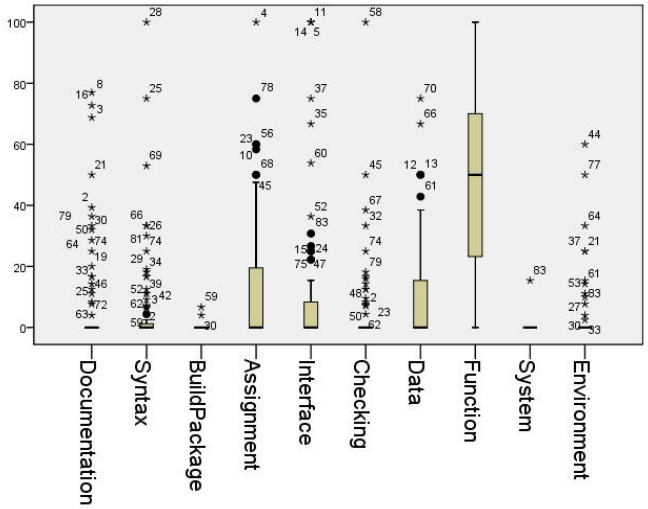

Figure 3: Box and whisker of the percentage of defects injected during design

1.4.2 When Are the Defects Injected During Design Removed?

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

	Docs.	Syn.	Build	Assign.	Inter.	Check	Data	Func.	Syst.	Env.
Mean	6.9	6.0	0.1	12.6	10.0	4.6	9.8	46.6	0.2	3.1
Lower	3.3	2.5	0.0	8.2	5.1	1.6	6.3	39.7	0.0	0.9
Upper	10.5	9.5	0.3	17.0	15.0	7.6	13.3	53.5	0.6	5.3
Std. dev.	16.6	16.0	0.8	20.2	22.5	13.8	16.0	31.5	1.7	10.1

Table 2: Percentage of defect types injected during design

Table 3 shows the mean (with 95% confidence interval) and standard deviation for the different phases. The 95% confidence interval is bounded by 45.8% and 61.0%. As previously shown, the standard deviation was high, indicating the high variability between individuals. From this we learned that approximately 50% of the defects injected during Design were found in the detailed level design review (DLDR) phase.

	DLDR	Code	CR	Comp	UT
Mean	53.4	9.6	8.9	2.5	25.7
Lower	45.8	5.7	5.2	0.0	19.3
Upper	61.0	13.4	12.5	5.2	32.0
Std. dev.	34.8	17.5	16.7	12.3	29.2

Table 3: Phases where are founded the design defects (percentage)

Figure 4 shows the box and whisker charts displaying the percentage of defects found in the different phases and the histogram for the defects founds in DLDR. Figure 4 also shows the high variability between individuals in the percentage of defects found during DLDR and UT.

Code and code review have a similar percentage of defects that were injected during design. Approximately 10% of the defects were found and removed in each of those phases. Approximately 2.5% of the design defects were found during the compile phase; it is likely that the defects found in compile were pseudo-code defects. And finally, around 25% of the defects were found in unit test (UT). This means that in the PSP accounting, one of every four defects injected during design escapes all phases prior to UT. We know, of course, that not all the defects that escape into UT are found in UT. UT will not have a 100% yield; therefore the percentage of defects found in each of these phases is smaller than reported while the actual percentage of escapes into UT is a lower limit. An estimate or measurement of the UT yield will be necessary to revise these phase estimates.

1.4.3 Cost to Remove the Defects Injected in Design

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

It would also 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.

1.4.3.1 Phase Removal Cost

What is the cost, in each removal phase, to find and fix a defect injected in design? Design defects can be removed in the detailed level design review (DLDR), code, code review (CR), compile, and unit test (UT) 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 design defects in one or more phases, our sample size varied by phase. We had data from 67 engineers for DLDR, 29 each for code and CR, six for compile, and 55 for UT. We excluded the cost of finding design defects in the Comp phase because we had insufficient data for that phase.

Table 4 shows the mean, lower and upper 95% confidence interval, and the standard deviation for the find and fix time (in minutes) for design defects in each of the studied phases. We might have expected an increased cost in each phase but this is not what the data showed.

	DLDR	CODE	CR	UT
Mean	5.3	5.1	4.2	23.0
Lower	3.7	2.5	2.6	11.6
Upper	6.9	7.6	5.7	34.3
Std. dev.	6.6	6.7	4.1	42.0

Table 4: Cost of find and fix defects injected in design segmented by phase removed

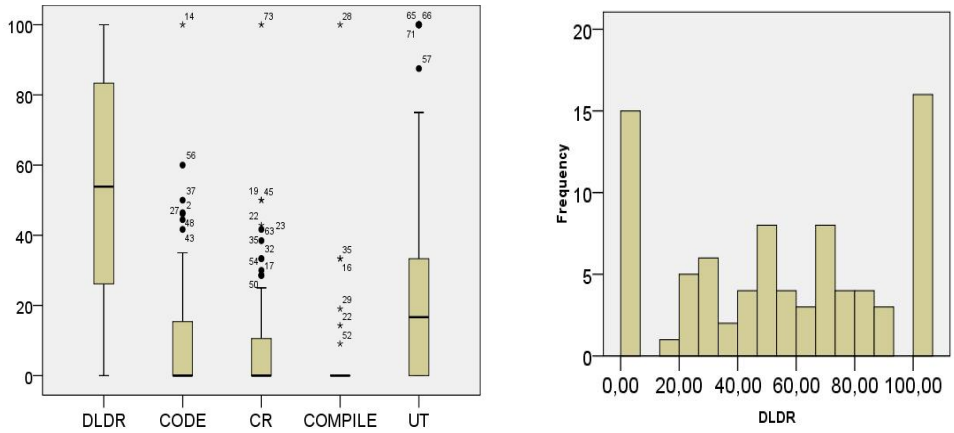

Figure 4: In which phase are the design defects found? – Variability between individuals

Rather, **the "find and fix" cost remained almost constant during DLDR, code, and CR**; in fact, the cost decreased a little in these phases, though the differences were not statistically significant. Further analysis will be needed to determine which of the defect finds in code and code review escaped through design and an effective (as opposed to ineffective) design review. Regardless, any defect discovered in these phases is essentially found by an inspection process where the "fix" time is short because the root cause has been identified.

We are not stating here that the cost of finding and fixing a design defect during DLDR, code, and CR is necessarily the same. We are stating that using PSP, the design defects that are removed during DLDR cost approximately the same as removing the ones that escape from design into code and those that escape from design into CR.

As we expected**, the average cost of finding a design defect during UT is much higher than in the other phases** by almost a factor of 5.

We also found a high variability among individual engineers. This variability can be seen in the box and whisker chart in Figure 5. We tested for normal distribution after log transformation of find and fix times for DLDR, code, CR, and UT and all are consistent ($p > 0.05$ using a Kolmorov-Smirnov and Shapiro-Wilk test) with a log-normal distribution. This test is primarily useful to verify that we can apply regression to the transformed data; however, understanding the distribution also helped to characterize the asymmetry and long tailed nature of the variation. That is, the log-normality affirmed our intuition that some defects found in test required far more than the average effort to fix, making test time highly variable. We also observed that the both the mean and variation of rework cost in the DLDR, code, and CR phases were significantly lower than UT in both the statistical sense and the practical sense.

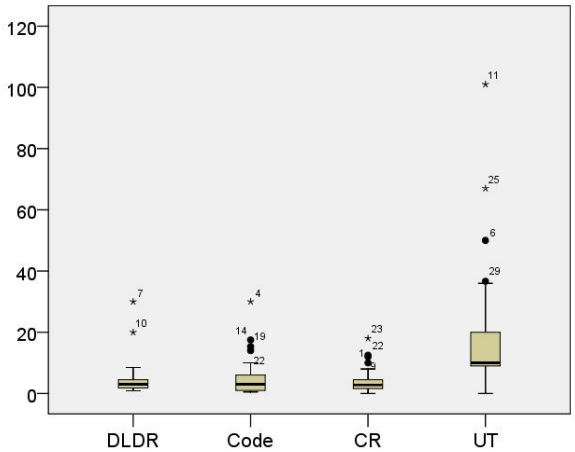

Figure 5: Box and whisker of the cost of find and fix a design defect segmented by phase removed

1.4.3.2 Defect Removal by Type

What is the find and fix cost, per defect type, of defects injected during detailed design? As we mentioned before, we had few build/package and system defects injected during design. As a result, we didn't have enough data for a statistically significant analysis of the cost of removing these two types of defects. However, we were able to analyze the remaining defect types.

Table 5 presents the mean, lower, and upper 95% confidence interval and the standard deviation for the find and fix cost of design defects, segmented by type. The cost, in minutes, for "find and fix" fell into three groups:

- a group that has a mean near 5 minutes: Documentation, Syntax, Interface, Checking
- a group, composed only of Assignment defects, that has a mean near 7 minutes
- a group that has a mean near 10 minutes: Data, Functions, Environment

The confidence interval of the second group is sufficiently wide that it is not clearly distinct from either of the other two groups, falling more or less in between. More data would help to clarify this grouping. We want to emphasize that the third group, including data, functions, and environment, takes twice the time to find and fix the defects than the documentation, syntax, interface and checking types of the first group.

	Docs.	Syn.	Assign.	Inter.	Check	Data	Func.	Env.
Mean	5.6	4.3	7.3	5.4	4.9	11.0	9.3	10.5
Lower	3.6	1.8	1.9	2.5	2.2	2.2	6.9	3.0
Upper	7.6	6.7	12.7	8.2	7.5	19.8	11.7	17.9
Std. dev.	4.1	3.7	16.3	7.3	5.2	25.6	10.1	11.7

Table 5: Cost of find and fix defects injected in design discriminated by defect type

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 6.

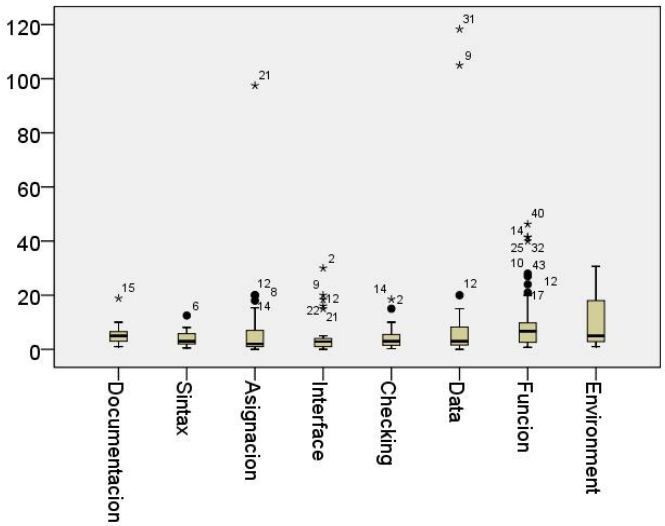

Figure 6: Box and whisker of the cost of find and fix a defect segmented by defect type

1.5 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 observe that there exists 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 design. Our analysis showed that most common design defects (46%) are of type function. This type belongs to the group of the most costly defects to find and fix. Data and environment defect types are in the same cost group category as Function.

In addition, the analysis showed that build/package and systems defects were seldom injected in the design 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.

In the final three PSP course exercises, defects were injected roughly equally in the design and code phases; that is, nearly half of the defects were injected in design. Half of the design defects were found early through appraisal during the detailed design review (DLDR). However, around 25% were discovered during unit test, where defect find and fix is almost five times more expensive in time.

While this analysis provided insights into the injection and removal profile of design 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, including a study about the defects injected during the code phase, 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. In particular, we want to determine how variation in design and design review affects defect leakage into these later phases. During a related analysis, we will examine the effects of design and design review on secondary defects injected in the unit test phase.

1.6 REFERENCES/BIBLIOGRAPHY

[Ferguson 97]
Ferguson, Pat; Humphrey, Watts S.; Khajenoori, Soheil; Macke, Susan; Matvya, Annette. *Results of Applying the Personal Software Process,* Computer, vol. 30, no. 5, pp. 24—31, 1997.

[Hayes 97]
Hayes, Will; Over, James. *The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers,* Technical Report, Carnegie Mellon University, Software Engineering Institute, no. 97-001, 1997.

[Humphrey 05]
Humphrey, Watts S. *PSP A Self-Improvement Process for Software Engineers,* Addison-Wesley, 2005.

[Johnson 99]
Johnson, Philip M.; Disney, Anne M. *A Critical Analysis of PSP Data Quality: Results from a Case Study,* Empirical Software Engineering, vol. 4, no. 4, 317—349, 1999.

[Paulk 06]
Paulk, Mark C. *Factors Affecting Personal Software Quality,* CrossTalk: The Journal of Defense Software Engineering, vol. 19, no. 3, pp. 9—13, 2006

[Paulk 10]
Paulk, Mark C. *The Impact of Process Discipline on Personal Software Quality and Productivity,* Software Quality Professional, vol. 12, no. 2, pp. 15—19, 2010.

[Rombach 08]
Rombach, Dieter; Munch, Jurgen; Ocampo, Alexis; Humphrey, Watts S.; Burton, Dan. *Teaching disciplined software development,* The Journal of Systems and Software, vol. 81, no. 5, pp. 747—763, 2008.

[Wholin 98]
Wohlin, C.; Wesslen, A. *Understanding software defect detection in the Personal Software Process,* Proceedings of the Ninth International Symposium on Software Reliability Engineering, pp. 49—58, 1998.

2. BIOGRAPHY

Diego Vallespir

Assistant Professor
Universidad de la República

Diego Vallespir is an Assistant Professor at the Engineering School at the Universidad de la República, Director of the Informatics Professional Postgraduate Center at the same school, Director of the Software Engineering Research Group (GrIS) at the Universidad de la República and member of the Organization Committee of the Software and Systems Process Improvement Network in Uruguay (SPIN Uruguay).

Vallespir holds an Engineer title on Computer Science from Universidad de la República and a Master Science title in Computer Science from the same University. He has several articles published in international conferences. His main research topics are empirical software engineering, software process and software testing. He is currently finishing his PhD Thesis.

Bill Nichols

Senior Member of the Technical Staff
Software Engineering Institute

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 lead 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 publications 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.