

Critical Factors Affecting Personal Software Processes

Personal software process quality helps determine the success of software projects and organizations. Although encouraging, previous studies treated the Personal Software Process approach as a black-box tool for personal process improvement. We dig deeper into the factors affecting personal processes.

Xiaoming Zhong, *McGill University*

Nazim H. Madhavji, *University of Otago*

Khaled El Emam, *National Research Council, Canada*

Software development projects are plagued with quality problems, cost overruns, and schedule slippage. In part, these problems are rooted at the individual level.^{1,2} For example, defect quality problems in delivered software could result from flaws in low-level design and algorithmic logic championed by individuals in a project; delays in system delivery could result from excessive rework by individuals; and so on. These localized problems can eventually add up to project-level problems.

For an organization to improve the quality of its software products and processes, it is important that individual developers have disciplined personal processes.²

We can observe the performance of an individual's software process through particular attributes, such as lines of code developed per hour (LOC/hour) or number of defects per thousand lines of code. Clearly, to better understand an individual's processes and therefore control and improve such processes, knowing the specific attributes of personal software processes should interest the individual and the organization alike. For example, what factors in an individual's process are likely to impact his or her productivity? With such knowledge, we can then focus on relevant factors during personal process improvement, instead of floundering on a multitude of fuzzy factors.

This article discusses the results of an experiment we conducted at McGill University to study the critical factors affecting personal processes.

A Closer Look

Most process improvement models have tended to address organization-level rather than individual-level issues and measures. Examples are the Capability Maturity Model for software, Software Process Improvement and Capability dEtermination, Quality Improvement Paradigm, Total Quality Management, and the Process Cycle.³⁻⁷ A notable exception is Watts Humphrey's Personal Software Process, which focuses on helping individuals acquire a disciplined approach to developing software.²

PSP has a maturity framework, which Humphrey structured as an evolving se-

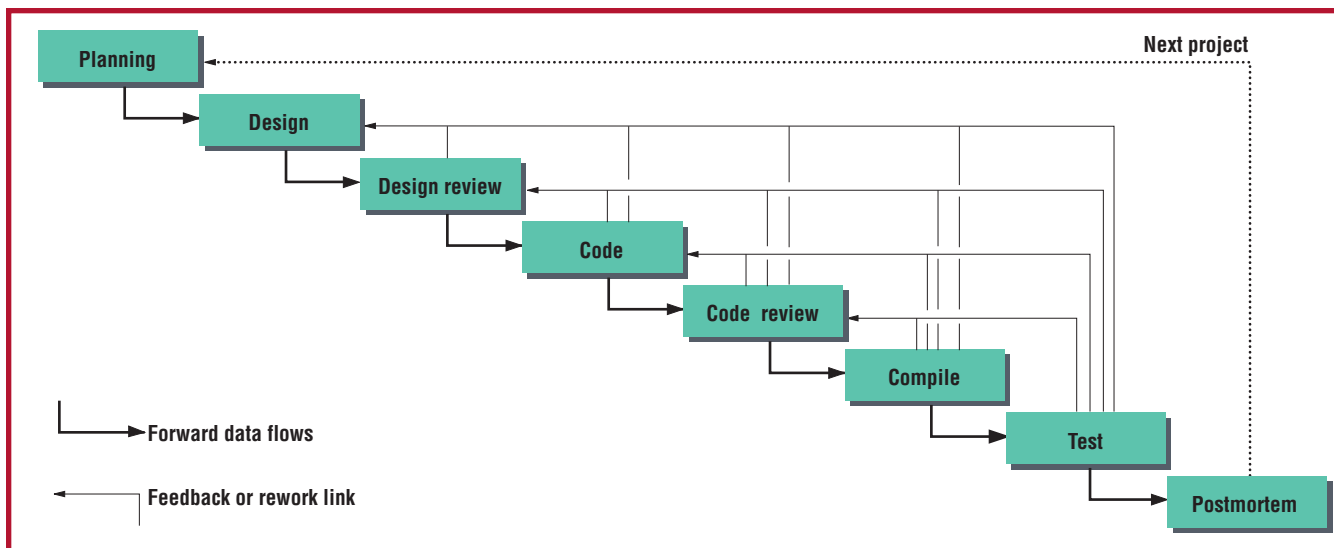


Figure 1. A sample Personal Software Process model.

quence of seven upward-compatible personal processes. Each process is defined, which helps guide individuals through an evolutionary path from initial process concepts, such as project planning, to subsequent concepts, such as defect prevention.

By and large, the many studies conducted on PSP⁸⁻¹² have shown that the approach (as a whole, or certain specific elements of it, such as code review) does positively impact personal processes and the resulting software. However, these studies generally do not dig deeply into personal processes—for example, the average number of phases backtracked to fix a defect (Abtk)—to identify which factors affect the specific attributes of personal processes, such as defect density (Dds), defect removal rate (Drr), and productivity (LOC/hour).

Studying such low-level details of a personal software process is important because they give a deeper insight into what drives process improvement (over and above the more abstract PSP elements, such as code reviews and size estimation). For example:

- When we use Dds to evaluate software quality, Yield (the percentage of defects removed before the first compile) might be important to consider.
- A decrease in Abtk contributes to the increase in Drr and LOC/hour.

We know that low Dds suggests high product quality. However, low Dds coupled with low Yield might be a result of poor defect detection, and this would imply poor product quality.

If we see a low Abtk value—implying that software defects pertain to the current

or recent development phase—we can remove the defects relatively easily,¹³ leading to higher productivity and lower costs.

Typically, such detailed points result from quantitative studies, which focus on specific software process variables.

Experimental Study

We conducted the McGill University experiment with students as subjects and their class assignments as programming tasks. We collected data from 53 students taking an undergraduate course entitled Personal Software Engineering (in which the instructor taught Humphrey's PSP concepts) in fall 1997. The students all had a similar computer science background. Prior to taking this course, they had been exposed to the object-oriented paradigm and software engineering. We asked each subject to carry out eight projects (once per week) in the programming language C++ using their personal software processes. We used product and process templates to collect software development data.

Process Model

PSP provides a well-defined framework to assist individuals in building a disciplined personal software process. Figure 1 shows the process model defined in PSP and used by the subjects in our experiment. In this model, project life-cycle activities are divided into chronological phases: planning, design, design review, code, code review, compile, test, and postmortem. The origin of the thin arrows indicates the phase during which an individual detects a defect, and an arrowhead indicates the phase during which the individual injects the defect. The

Table 1**Process Attributes**

Attribute	Definition
Dds	Number of defects removed per thousand lines of code developed
Drr	Number of defects removed per hour
Productivity	Lines of code developed in an hour
A/FR	Time spent in design review and code review as a percentage of time spent in compile and test
Yield	The percentage of defects removed before the first compile
Abtk	Average number of process phases backtracked to fix a defect
DT/TDT	Time spent in a design phase as a percentage of time spent in total development time
TT/TDT	Time spent in a test phase as a percentage of time spent in total development time
NTD/ND	Number of defects detected during testing as a percentage of the total number of defects

solid arrows indicate the flow of artifacts. The dotted line from Postmortem back to Planning indicates feed-forward into a subsequent project.

Process Attributes

Each programming assignment resulted in some 70 pieces of data collected by each student. From this, we identified 10 attributes, shown in Table 1, to characterize a personal software process. These attributes form a basis for data analysis.

All the attributes in Table 1 are important in software engineering. We elaborate on those considered somewhat uncommon: appraisal-to-failure ratio (A/FR) and Abtk.

A/FR gives us a way to balance between the review effort and the compile-and-test effort. A high A/FR implies that individuals are detecting most defects earlier in the life cycle (which should reduce software costs). However, putting forth an inordinate amount of review effort (that is, having an extremely high ratio) would be counterproductive.

Also, a high Abtk value implies that individuals are detecting defects much farther away from the defect injection points, and hence that could lead to high software costs and prolonged cycle time.

Descriptive Models

We organized the attributes listed in Table 1 into a defect-based quality model and a development productivity model. The former describes a personal process's elements, through the process attributes, with a focus on defect quality of the software developed. The latter describes a personal process's elements with a focus on development productivity.

One clear defect-quality indicator is Dds. Drr is also a defect quality indicator, but it's not so common in the literature. A high Drr value suggests that defects are fixed closer to their source, implying that software is

high in quality at release time. The defect-based quality model is

$$Q(Dds, Drr ; A/FR, Yield, Abtk, DT/TDT, TT/TDT, NTD/ND)$$

where Q denotes a particular process's or product's quality. Dds and Drr are the dependent variables of interest. A/FR, Yield, Abtk, DT/TDT, TT/TDT, and NTD/ND are the independent variables of interest.

For the productivity model, LOC/hour is a widely used measure. The development productivity model is

$$P(LOC/hour ; A/FR, Yield, Abtk, DT/TDT, TT/TDT, NTD/ND)$$

where P denotes productivity. LOC/hour is the dependent variable and A/FR, Yield, Abtk, DT/TDT, TT/TDT, and NTD/ND are the independent variables. In both the quality and productivity models, the separation of variables into dependent and independent variables facilitates data analysis in terms of the impact of the latter variables onto the former ones.

Data Analysis Model

Regression methods show the relation between dependent and independent variables whose relation is imperfect in that we do not have one y for each x .¹⁴ In software engineering, we can cite the relation between software size and development time as an example of an imperfect relationship in that no one-to-one relationship exists between the two. Researchers have employed various regression models to estimate the relationships between one variable and another by expressing one in terms of a regression function (such as a linear function, quadratic function, or log-linear function) of the other.

Typically, two methods exist for choosing

Table 2**Regression Equations for Dds**

Independent variable	Quadratic equation	R ²
A/FR	$Dds = -0.0002 \times A/FR^2 + 0.0354 \times A/FR + 45.073$	0.0151
Yield	$Dds = -0.0148 \times Yield^2 + 1.2290 \times Yield + 31.392$	0.2584
Abtk	$Dds = -4.5312 \times Abtk^2 + 21.2540 \times Abtk + 23.584$	0.0450
DT/TDT	$Dds = -0.0033 \times DT/TDT^2 - 0.4208 \times DT/TDT + 51.052$	0.0164
NTD/ND	$Dds = -0.0107 \times NTD/ND^2 + 0.9520 \times NTD/ND + 36.406$	0.1829
TT/TDT	$Dds = 0.0043 \times TT/TDT^2 + 0.1290 \times TT/TDT + 40.715$	0.0190

a particular regression function: analyzing the phenomenon concerned, and examining the scatter diagrams plotted from the observed data.¹⁵ Researchers often prefer the former method when the assumed function represents some basic, or causal, mechanism associated with the experimental units and the factors under investigation.

In software development, however, the interrelationships among the process attributes (see Table 1) are not so obvious as to make linear regression the method of choice for data analysis. (For example, a requirements defect costs exponentially more if fixed in later software development phases.)^{16,17} Therefore, we must examine the need for higher-order data analysis models (for example, a quadratic function).

So, we performed an F-test to ascertain the need for the quadratic effect in the regression model. Our analysis suggests a quadratic-effect presence between the independent and dependent variables.

We therefore performed quadratic regression analysis between each dependent variable and its corresponding independent variables shown in the two descriptive models, Q and P. A quadratic regression function's form is

$$DEP = a \times INDEP^2 + b \times INDEP + c$$

where a equals the quadratic coefficient, b equals the linear coefficient, and c equals the intercept.

Given this regression equation, the value of the independent variable that corresponds to the optimal value of the dependent variables is given by the formula:

$$INDEP = -b / 2a.$$

Data Collection and Threats to Validity

We collected data using Humphrey's templates, which several institutes and organizations that teach or practice PSP have used for several years. They facilitate the recording of software and process data, such as size, time,

defect types and frequency, and defect injection and removal phases.² Every week, we gave two concept-oriented lectures to the subjects. On a daily basis, we provided an adequate contact period between the subjects and the coaches through tutorials, one-on-one sessions, and emails. We validated the gathered data every week and returned them for correction to the subjects in case of inconsistencies and errors. All these activities contributed to a rigorous process of data collection and integrity, thereby validating the empirical procedures followed.

Results and Implications

Our study's purpose was to investigate which factors most influence software quality and productivity—for example, Dds, Drr, and LOC/hour.

Critical Factors Affecting Defect Density

Table 2 lists the quadratic relationship between dependent variable Dds and its independent variables. It shows that Dds has a relatively strong relationship only with Yield. Figure 2 charts the relationship between those two variables. The maximum Dds value is associated with projects with a Yield $(-1.229 / (2 \times (-0.01481)))$ of 42%. Both low and high Yield values are related to low Dds.

If Dds decreases, the code quality is improving. However, low Dds associated with high Yield values could imply high code quality, because a high Yield value indicates strong defect detection. On the other hand, low Dds associated with low Yield values—

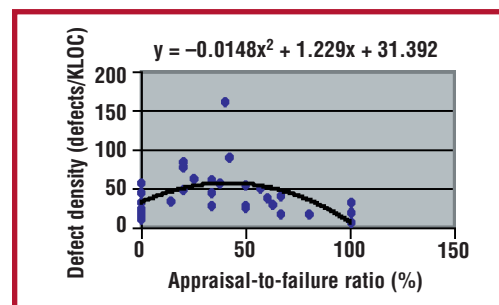
**Figure 2. Relationship between defect density and Yield.**

Table 3**Regression Equations for Drr**

Independent variable	Quadratic equation	R ²
A/FR	$Drr = -0.0006 \times A/FR^2 + 0.2280 \times A/FR + 2.7327$	0.2940
Yield	$Drr = -0.0033 \times Yield^2 - 0.1472 \times Yield + 12.4070$	0.2737
Abtk	$Drr = -1.8761 \times Abtk^2 - 10.5210 \times Abtk + 27.5430$	0.0221
DT/TDT	$Drr = -0.0321 \times DT/TDT^2 - 1.2469 \times DT/TDT + 5.4882$	0.0281
NTD/ND	$Dds = -0.0023 \times NTD/ND^2 - 0.3008 \times NTD/ND + 20.6260$	0.0449
TT/TDT	$Dds = 0.0051 \times TT/TDT^2 - 0.6243 \times TT/TDT + 24.0180$	0.0756

Table 4**Regression Equations for LOC/hour**

Independent variable	Quadratic equation	R ²
A/FR	$LOC/hour = -0.0005 \times A/FR^2 + 0.2895 \times A/FR + 16.609$	0.244
Yield	$LOC/hour = 0.0085 \times Yield^2 + 0.4715 \times Yield + 30.936$	0.3899
Abtk	$LOC/hour = 4.048 \times Abtk^2 - 23.985 \times Abtk + 63.938$	0.0655
DT/TDT	$LOC/hour = -0.0414 \times DT/TDT^2 + 1.3816 \times DT/TDT + 23.587$	0.0182
NTD/ND	$LOC/hour = 0.0037 \times NTD/ND^2 - 0.4322 \times NTD/ND + 40.055$	0.0430
TT/TDT	$LOC/hour = -0.0054 \times TT/TDT^2 + 0.236 \times TT/TDT + 30.340$	0.0027

implying poor code quality—might be a result of poor defect detection. So, our first observation is

When individuals use Dds to evaluate software quality, Yield might also be an important factor to consider.

Critical Factors Affecting Defect Removal Rate

Table 3 lists the quadratic relationships between dependent variable Drr and its independent variables. As it shows, Drr has a relatively strong relationship with A/FR and Yield.

Figures 3a and 3b depict the relationship between Drr and A/FR and between Drr and Yield, respectively. Drr has been used to assess engineers' performance and product quality.^{9,18} If Drr increases, engineers' defect detection ability and product quality also increase. (That is, high Drr generally implies that the defects have been relatively easy to

fix, which, in turn, implies that the individuals fixed the defects close to their points of origin in the software life cycle. So, high Drr implies that the product, by the time it's completed, is relatively defect free.) Increasing the A/FR value and increasing the Yield value positively affect Drr.

According to the relationship equation between Drr and A/FR, the Drr value increases with the increase in the A/FR value, and we can achieve the optimal Drr value with an A/FR $(-0.228 / (2 \times (-0.0006)))$ of 190%. We know that A/FR is the ratio of *design and code review time to compile and test time*. Also, we know that it measures the relative effort spent in early defect removal. Its objective is to detect defects in earlier phases and thus improve the Drr. However, once A/FR meets the objective, further increase in the A/FR will likely decrease or stabilize Drr. So, our second observation is

The A/FR value might be a useful guide

Figure 3. Relationship between (a) defect removal rate and appraisal-to-failure ratio, and (b) defect removal rate and Yield.

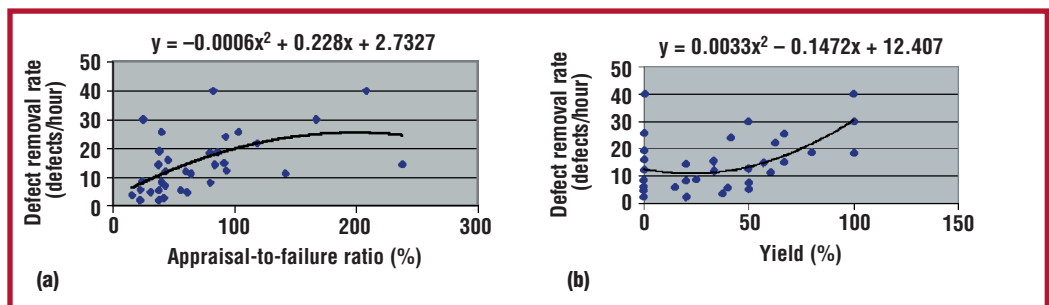


Table 5**Regression Equations between Critical Factors and Other Factors**

Quadratic equation	R ²
$ABtk = -0.00005 \times A/FR^2 - 0.0208 \times A/FR + 3.3126$	0.4316
$NTD/ND = 0.0023 \times A/FR^2 - 0.8302 \times A/FR + 79.566$	0.3781
$TT/TDT = 0.001 \times A/FR^2 - 0.3241 \times A/FR + 31.71$	0.3529
$ABtk = -0.00003 \times Yield^2 - 0.0092 \times Yield + 2.6081$	0.2196
$NTD/ND = 0.0026 \times Yield^2 - 0.1647 \times Yield + 59.59$	0.1750
$TT/TDT = 0.0018 \times Yield^2 - 0.2343 \times Yield + 17.802$	0.0776

for software developers in adjusting their review time to achieve a high Drr value during software development.

From the relationship between Drr and Yield, the Drr value slightly decreases with the increase in the Yield value until Yield $(-(-0.1472) / (2 \times 0.0033))$ is 22.3%, and then Drr substantially increases with the increase in the Yield value (Yield > 22.3%). So, our third observation is

A low Yield value might imply that review skills are poor or that the effort spent on review is not adequate. In this case, most of the defects captured during review might be syntax or simple errors. This might be interpreted to mean that Drr decreases when the Yield value is low. Therefore, Yield has little positive effect on Drr unless it has achieved a high level.

Critical Factors Affecting Productivity

Table 4 lists the quadratic relationships between dependent variable LOC/hour and its independent variables. As it shows, LOC/hour has relatively strong relationships with A/FR and Yield. Figures 4a and 4b depict the relationship between LOC/hour and A/FR and between LOC/hour and Yield, respectively.

LOC/hour increases when A/FR increases and decreases slightly with the increase in the Yield value until Yield $(-(-0.4715) / (2 \times 0.0085))$ is 27.7%. Then, it increases greatly with the increase in the Yield value (Yield > 27.7%). So, our fourth observation is

Reviews positively effect LOC/hour because the latter increases with the increase in A/FR and also with the increase in Yield.

Developers often resist performing reviews. That's not because reviews are seen as worthless. Indeed, there is increasing awareness of the benefits of performing software reviews in terms of software-defect quality. Despite that, developers tend to rush toward testing on the assumption that reviews would slow down the development process. Our findings definitively show that reviews also help increase productivity.

Other Noticeable Factors and Inferences

We know from observations 1 through 4 that the critical factors, such as Yield and A/FR, positively effect quality and productivity (Dds, Drr, and LOC/hour). We can theorize, and perhaps even back up with experience, and perhaps even back up with experience, why such an impact exists (this rationale is not self-evident from the findings). However, such arguments would be more credible if our data set could help explain our intuition. So, to do that, we further analyzed the relationships between the critical factors and other factors. (We mean the term *inference* as *reasoned conclusion*, not *statistical inference*.¹⁹)

Other Noticeable Factors

Table 5 lists the equation relationships between critical factors and other factors. The first three equations (Abtk, NTD/ND, and TT/TDT) are considered relatively strong.

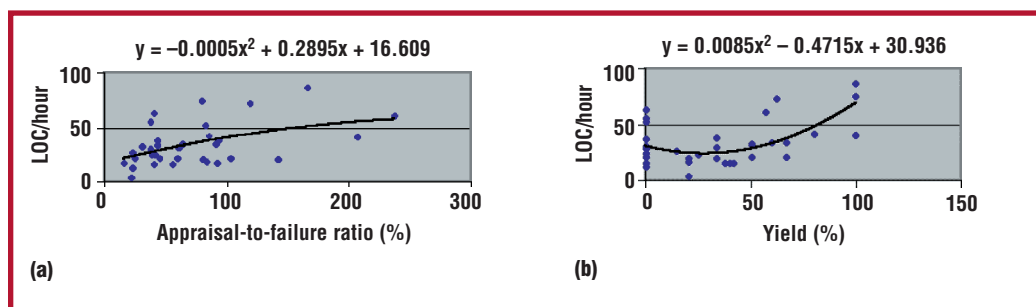


Figure 4. (a) Relationship between productivity and appraisal-to-failure ratio, and (b) between productivity and Yield.

Figure 5. Relationship between appraisal-to-failure ratio and average number of phases backtracked to fix a defect.

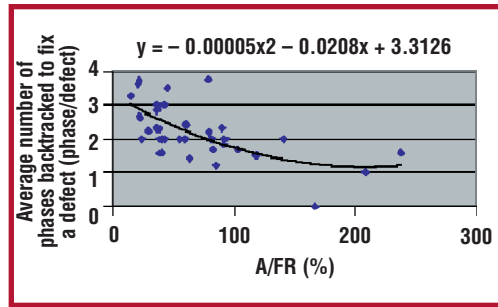


Figure 6. Relationship between the number of test defects as a percentage of total defects and appraisal-to-failure ratio.

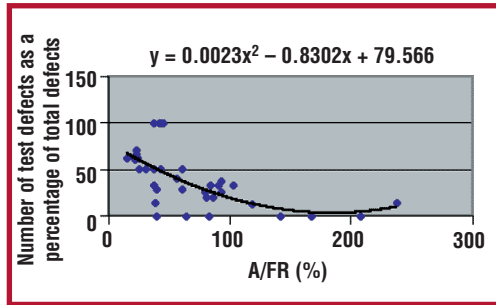
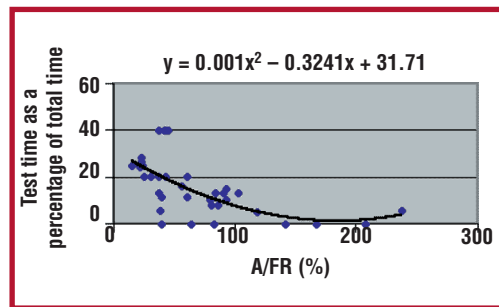


Figure 7. Relationship between the test time as a percentage of total time and appraisal-to-failure ratio.



Noticeable Relationships

Figure 5 relates to A/FR. Abtk drops as A/FR increases. So, considering Figures 3a and 4a with Figure 5, our first inference is

A decrease in Abtk might contribute to an increase in Drr and LOC/hour.

This is because a low Abtk can imply that software defects pertain to the current development phase or to recent development phases. When we have a low Abtk, we can relatively easily remove the defects, and the software project tends to move forward at an increased speed, giving rise to higher LOC/hour and lower cost.

Figure 6 depicts the relationship between A/FR and NTD/ND. A/FR values above 100% are associated with a relatively low percentage of test defects, and vice versa. So, our second inference is

High effort in software reviews might lead to fewer defects in the test phase.

Reducing test defects is an important software development objective because having many test defects implies poor software quality, and because test defects are relatively expensive to fix. Because determining product quality during development is generally difficult for engineers, the A/FR measure is a useful guide for personal practice. Although our finding generally hovers around 190%, knowing how high the A/FR ratio should be needs further study. However, we need to consider the appraisal cost and the cost of fixing defects during the test phase.

Another important personal software process attribute is the time spent during testing software. The less time needed in testing, the better the cycle time (and other indirect effects, such as customer satisfaction). Figure 7 presents the relationship of A/FR and TT/TDT. There is an optimal A/FR value corresponding to the TT/TDT value. So, our third inference is


Reviews reduce the time spent in the test phase because fewer defects creep in.

By examining inferences 1 through 3 and Figures 5 through 7, we see that the optimal Abtk, NTD/ND, and TT/TDT values always occur when the A/FR value is high. Also, the Drr and LOC/hour are high when A/FR is high. So, based on what we've observed, we've deemed that A/FR positively affects Drr and LOC/hour.

Individuals are increasingly recognizing personal excellence as a significant contributor to project success,^{1,2,20} and personal software processes are a medium to exhibit personal excellence. Humphrey's PSP framework instills excellence in personal software processes.²

Previous studies on PSP suggest its benefits—namely in software defect quality, size estimates, and planning.⁸⁻¹¹ Although the results of those studies are encouraging, they treated PSP as a black-box tool for personal process improvement. Generally, they did not identify the detailed factors underlying personal processes and affecting product quality and productivity.

Our experiment complements the previ-

ous studies, and our findings led us to conclude that A/FR and Yield are two critical factors affecting personal software processes. Because our findings are limited to one experiment involving subjects in a university setting, developers in an industrial environment should use them with caution. We expect readers to view this article as hopeful; we call for more of such studies in academia and industry. With additional studies, we can collectively build stronger theories underlying personal software processes. (In that respect, we are gathering additional PSP data aimed at a multiyear study.) 

13. B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
14. F. Mosteller and J.W. Turkey, *Data Analysis and Regression*, Addison-Wesley, Reading, Mass., 1977.
15. B. Ostle, *Statistics in Research*, Iowa State Univ. Press, Ames, Iowa, 1963.
16. B. Boehm and P. Papaccio, "Understanding and Controlling Software Costs," *IEEE Trans. Software Eng.*, Vol.10, 1988, pp.1462-1476.
17. A. Davis, *201 Principles of Software Development*, McGraw Hill, New York, 1995.
18. A. Naqvi, *Personal Progress Functions in the Software Process: A Replicated Study*, master's thesis, School of Computer Science, McGill Univ., Montreal, 2000.
19. L. Ott, W. Mendenhall, and R. Larson, *Statistics: A Tool for the Social Sciences*, Duxbury Press, Boston, 1978.
20. N. Whitten, *Managing Software Development Projects*, 2nd ed., John Wiley & Sons, New York, 1995.

References

1. F. Brooks, "The Mythical Man-Month," *Essays on Software Engineering*, anniversary ed., Addison-Wesley, Reading, Mass., 1995.
2. W.S. Humphrey, *A Discipline For Software Engineering*, Addison-Wesley, Reading, Mass., 1995.
3. M. Paulk et al., "Capability Maturity Model for Software (Version 1.1)," *IEEE Software*, Vol. 10, No. 4, July 1993, pp. 18-27.
4. T.P. Rout, "SPICE: A Framework for Software Process Assessment," *Software Process: Improvement and Practice*, Vol. 1, No. 1, 1995, pp. 57-66.
5. V.R. Basili, "The Experimental Paradigm in Software Engineering," *Proc. Int'l Workshop Experimental Software Eng. Issues*, Lecture Notes in Computer Science, Vol. 706, 1992, Springer-Verlag, Berlin, pp. 3-12.
6. R.E. Zultner, "TQM for Technical Teams," *Comm. ACM*, Vol. 36, No. 10, Oct. 1993, pp. 79-91.
7. N.H. Madhavji, "The Process Cycle," *IEE/BCS Software Eng. J.*, Vol. 6 No. 5, Sept. 1991, pp.234-242.
8. W.S. Humphrey, "Introducing the Personal Software Process," *Annals of Software Engineering*, Vol. 1, 1995, pp. 311-325.
9. K. Sherdil and N.H. Madhavji, "Human-Oriented Improvement in the Software Process," *Proc. Fifth European Workshop on Software Process Technology*, Lecture Notes in Computer Science, Vol. 1149, 1996, Springer-Verlag, Berlin, pp. 145-166.
10. K. El Emam, B. Shostak, and N.H. Madhavji, "Implementing Concepts from the Personal Software Process in an Industrial Setting," *Proc. Fourth Int'l Conf. Software Process*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 117-130.
11. P. Ferguson et al., "Results of Applying the Personal Software Process," *Computer*, Vol. 30, No. 5, May 1997, pp. 24-31.
12. L. Hou and J. Tomayko, "Applying The Personal Software Process in CS1: An Experiment," *SIGCSE Bulletin ACM Special Interest Group on Computer Science Education*, Vol. 30, No. 1, Mar. 1998, pp. 322-325.

About the Authors

Xioaming Zhong is a software designer at Gimmetry Systems. His research interests include software process and software quality issues. He received BS and MS degrees in computer science from Fuzhou University and McGill University, respectively. Contact him at 4200 de Courtrai, #21, Montreal, PQ, Canada, H3S 1C2; zxm@cs.mcgill.ca.

Nazim H. Madhavji holds a Chair in Information Science (Software Engineering) at the University of Otago. His research interests are in empirical studies of software development, from personal level to project and organizational levels; congruence between software products and processes with their environments; software quality and measurements; process design, customization and generalization; feedback; requirements; product lines; and software architectures. He is Chair of the IEEE TCSE Committee on Software Process. Contact him at madhavji@infoscience.otago.ac.nz.

Khaled El Emam is at the National Research Council in Ottawa. He is co-editor of ISO's project to develop an international standard defining the software measurement process, and leading the software engineering process area in the IEEE's project to define the Software Engineering Body of Knowledge. He is an adjunct professor at both the School of Computer Science at McGill University and the Department of Computer Science at the University of Quebec at Montreal. His current work focuses on object-oriented measurement (www.object-oriented.org). He received a Ph.D. from the Department of Electrical and Electronics Engineering, King's College, University of London.