

# In-the-SPIN

Newsletter of the **Boston@SPIN**

Issue 55 Fall 2005

Editor: **Judi Brodman**

## From the Editor

**By: Judi Brodman, In-the-SPIN Editor**



Happy New Year to all our In-the-SPIN readers!!! It is hard for me to believe that we are already well into our SPIN year once more.

Our speakers thus far have been terrific, thanks to Donna Johnson, our Program Chair. When asked, they have graciously provided an article for me to include in the Newsletter. This issue contains articles by Jerry Weinberg and Watts Humphrey – our first two speakers this year. Both articles will definitely make you stop and think for a few minutes!

In addition to these two articles, this issue also contains a very interesting and informative article by one of our SPIN members on research that he conducted to see if culture has an impact on software productivity. Your guess is? See if you guessed correctly – read his article on page 2.

Also included in this issue are the results of the September, October, November and December Roundtables (RT). The RTs continue to be very popular and the sharing of experiences and expertise continues. If you haven't attended one, you really should. Also read the note included at the beginning of that section by our new *Roundtable Chair*, Chuck Anastasia.

If you have any comments or questions about our *In-the-SPIN* newsletter, please let me know - [brodman@logos-intl.com](mailto:brodman@logos-intl.com). Or, if you would like to contribute to the Newsletter, please send me a draft of your article and let's go from there.

Happy, healthy, and prosperous 2006 to all our readers!

## From the Chair

**By: Rick Brenner, Boston SPIN Chair**

As a local professional society, Boston SPIN is both reasonably typical and very unusual. It's reasonably typical in that our speakers are top notch, and our refreshments get the job done. Our meetings are informative and fun, and the MITRE facility is great too. We expect these things from the organizations we belong to, and Boston SPIN delivers.

But Boston SPIN is also very unusual -- it's one of the very few professional organizations that charges no membership fees. You show up, you participate and it's all free. We've always operated that way, and I hope and expect that we always will.

But it isn't magic. We depend on our sponsors for financial support, and they've all been great. Even more important, we depend on volunteers. Our Steering Committee and our committee chairs and other volunteers keep this SPIN running. It's an effort even when things go right, and it's even more when they don't. But our volunteers give of themselves every month to make our SPIN happen.

We acknowledge our sponsors and volunteers each month, and I'm sure they appreciate it. I have an idea, though, that I hope you'll help implement. If you know some Boston SPIN volunteers, or someone from one of our sponsoring organizations, please thank them personally, either at the meeting, or by email, or by carrier pigeon. You'll make them feel good, and you'll feel good too.

Oh, one more idea. *You* can become a volunteer. Just introduce yourself to any volunteer at a meeting, or send me a note at [rbrenner@ChacoCanyon.com](mailto:rbrenner@ChacoCanyon.com). And then before you know it, somebody will thank *you* for being a SPIN volunteer.

## A Special 'Thank You'

Speaking of "thank you", our thanks go out to Anna Glebova and Vlad Slezin who are no longer able to be our greeters due to late working hours. Anna will continue as our Web Wizard, and she will also continue analyzing and reporting on the feedback form data collected at every meeting.

We'll miss them both as greeters. Their warm smiles have been an important part of the face of Boston SPIN for so long, especially for new members.

But I'm sure we'll see them from time to time at meetings. And when we do, I hope we'll all give them the appreciations they so well deserve for their years of service to Boston SPIN.

**Boston@SPIN** Established  
Software Process Improvement Network January 1993

### IN THIS ISSUE . . .

From the Editor.....	1
From the Chair.....	1
A Special 'Thank You'.....	1
Contributor Spotlight.....	2
Speaker's Corner.....	3
Roundtables.....	6
Dear SPIN Doctor.....	10
SPIN Information.....	11
Upcoming Meetings.....	12
Quotes.....	12

# Contributor Spotlight

## Organizational Culture is Important in Software Productivity

Bruce Taylor, owner and principal of WorkingInUnison, an Organizational Development consulting firm

### Introduction



If you wanted to dramatically improve the productivity of a software development team, what would you do? Well, you might start by providing training to improve their skills; then you might improve the tools they work with; and finally you might improve their software process. Is that all? Can you do anything else to make them more productive?

I've been convinced for a long time that there is more you can do beyond the obvious technical improvements: you can improve the culture in which the team works. But it's very difficult to show that this "warm and fuzzy" improvement can have an effect on measured productivity, and this difficulty leads managers to ignore cultural improvements because it's nearly impossible to demonstrate a return on investment.

### The Study

I decided to investigate a very simple, limited version of the question: "Does the culture of the team members, the team manager, and the company affect the productivity of individuals or the team?" Specifically, I made the following hypothesis: "Personal and team productivity are affected most by the culture of the team, somewhat by the attitudes of the team's manager, and weakly by the culture of the company."

For the study, I operationalized "productivity" as three components:

1. efficiency: how much software is produced per unit time
  2. quality: how many errors does the product contain?
  3. schedule: how faithfully was the schedule followed?
- and I operationalized "culture" as:

1. respect: are people treated with respect?
2. honesty: is information shared accurately and freely?
3. support: do people support each other when needed?

I decided to measure productivity at the personal, team, and corporate level, and to measure culture within the team, for the team's manager, and within the company.

### The Survey

I created an on-line survey with three sections. The first section collected demographic information about the respondent and is summarized in Table 1 below.

The second section investigated productivity by asking these questions for self, team, and company. All questions were answered using a five-point Likert scale ranging from "Strongly Agree" (4) to "Strongly Disagree" (0)

1. "(I/my team/my company) work(s) very efficiently."
2. "(I/my team/my company) produces a very high quality product."
3. "(I/my team/my company) rarely miss schedule deadlines."

The final section asked about culture of the team, the manager, and the company:

1. "My (team/manager/company) always treats me with respect."
2. "My (team/manager/company) always supports me when I need help."
3. "My (team/manager/company) is always honest with me."

A word of caution is in order here. Because important terms like "efficiently" and "respect" were not explicitly defined in the survey, there may be a difference of interpretation among the respondents. However, the consistent sentence structure and the use of intensifiers like "always" and "very" minimize this variation.

### The Sample

I solicited participants in the comp.lang.\* Usenet groups which are mainly visited by active software developers: 155 people responded to the survey, but only 116 surveys were completed and these complete surveys comprise the sample. This sample carries the usual biases of self-selected samples: these are representative of programmers who enjoy answering surveys and have access to the Usenet groups. Table 1 summarizes the demographics of the sample.

47% of the sample comes from North America; another 43% come from Europe; and the remaining 11% are globally distributed except for the Far East., which is not represented. Because the responses are predominantly from North America and Europe, the results of this study should not be used to predict results in, for example, the Japanese or Chinese programming community.

### Regression Analysis

To analyze the dependencies further, I analyzed individual and team productivity against corporate culture and a variety of other variables. Table 2 shows the regression analysis for individual productivity (PERSPROD, adjusted  $R^2=0.11$ ) and team productivity (TEAMPROD, adjusted  $R^2=.39$ ).

The most striking features of the PERSPROD regression analysis are that only 11% of personal productivity is explained by the variables listed, and almost all of that variance is explained by the EXPERIENCE variable. In short, cultural issues seem to have very little to do with personal productivity but experience is very important in explaining it.

But the story for team productivity (TEAMPROD) is quite different. Notice that 39% of the team productivity is explained by the variables in the analysis, and that two of the variables, PROJULT and TEAMSIZ dominate the variance. Team productivity is moderately sensitive to other variables, but almost all the variance comes from project culture and from team size.

How long have you been working in software development?	
1-10 years	46%
11-20 years	26%
21-30 years	18%
31 years or more	9%
How many total employees does your company have?	
1-20 employees	22%
21-100 employees	23%
101-500 employees	18%
501-1000 employees	7%
1001 or more employees	30%
How many total people are in your immediate team?	
1-5 people	50%
6-10 people	27%
11-15 people	10%
15-20 people	1%
21 or more	12%
What is your level in the organization?	
Individual Contributor	66%
Team Leader	19%
First Level Manager	4%
Middle Manager	1%
Director	3%
Vice President	1%
Executive	6%

**Table 1: Sample Demographics**

## Interpretation

The message of this simple analysis is clear: personal productivity doesn't depend very much on cultural factors, because it is almost completely determined by the programmer's experience. But team productivity depends greatly on project culture and team size. In every case of correlation analysis, you have to be careful to distinguish correlation from causality. In this study, we should ask, "Is a team productive because it is happy, or happy because it is productive?" The data don't answer this question, but I suspect strongly that the two interact strongly and the issue of causality isn't really very important. So if you want to improve the productivity of an individual programmer, you would concentrate on training, tools and other technical factors. But to improve the productivity of a team, the most important thing you can do is to reduce the team size, and to improve the culture of the team (respect, support, and honesty).

Personal Productivity (PERSPROD)	
<i>Factors</i>	<i>Coefficients</i>
PROJCULT	0.17
MNGRCULT	0.06
COMPCULT	0.02
EXPERIENCE	0.37
COMPSIZE	-0.09
TEAMSIZ	0.14
Team Productivity (TEAMPROD)	
<i>Factors</i>	<i>Coefficients</i>
PROJCULT	0.40
MNGRCULT	0.05
COMPCULT	0.18
EXPERIENCE	0.19
COMPSIZE	0.08
TEAMSIZ	-0.48

**Table 2: Regression Analysis of Productivity**

## Speaker's Corner

The following article is the basis for the Preface in "Quality Software Management Volume 2" which Jerry Weinberg so graciously provided to our SPIN readers.

"When you can measure what you are speaking about and can express it in numbers, you know something about it. But when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind." -- Lord Kelvin

In the four decades I've spent in the software business, I've learned that there are three fundamental abilities you need to do a quality job of managing software engineering:

1. the ability to understand complex situations so you can plan a project and then observe and act so as to keep the project going according to plan, or modify the plan.
2. the ability to observe what's happening and to understand the significance of your observations in terms of effective adaptive actions
3. the ability to act appropriately in difficult interpersonal situations, even though you may be confused, or angry, or so afraid you want to run away and hide

All three abilities are essential for quality software management, but I didn't want to write a large, imposing book. Therefore, like any quality manager of software, I decomposed the project into three smaller projects, each one addressing one of these three fundamental abilities. Volume I, Systems Thinking deals with the first ability--the ability to understand complex situations. Volume III, Congruent Ac-

tion will deal with the third ability--the ability to act appropriately even in the presence of strong feelings. In this second volume--First-Order Measurement--I invite your attention to the ability to observe what's happening and to understand the significance of your observations.

I spent much time anguishing over the correct title for this volume before settling on First-Order Measurement. For one thing, when a book is published with the word "measurement" in the title, the author seems obliged to quote Lord Kelvin. I have kept to that tradition, but now feel compelled to analyze what Lord Kelvin meant, and did not mean, by his oft-quoted statement.

When you can measure what you are speaking about, Kelvin said, you know something. If you are measuring just for the fun of knowing things, then it doesn't much matter what you measure. But if you are trying to accomplish something, then arbitrary measurements are perhaps not what you need to know. Engineering is about accomplishing things, so engineering measurements are not arbitrary.

Lord Kelvin was a physicist. Although I wanted to work with computers since I was a boy, I was trained as a physicist because there was no computer training in my school days. Today I consider myself a software engineer, and though I still love physics, I distinguish between knowing in physics and knowing in engineering:

Physics knowing is understanding the universal laws of nature.

Engineering knowing is knowing how to do something.

I use the term "third-order" measurement for the kind of measurement that supports the physicist's search for universal laws. An example of third-order measurement would be an experiment that tested the First Law of Thermodynamics that says, in effect, that it's impossible to build a perpetual motion machine.

Engineers are not trying to know universal laws, but special laws relating to building specific things. Therefore, they are interested in what I call "first-order" and "second-order" measurement. An example of such measurement would be measurements to build and tune a high-efficiency machine.

The term "first-order measurement" is used in engineering to describe a measurement just adequate to the task of getting things built. First-order measurements are used in "back of the envelope" (or, in England, "back of the cigarette box") calculations. They support the "rule of thumb" and the rough sketch, as well as the "quick and dirty" or "seat of the pants" estimate. These are the measurements that allow us to build a machine in the first place.

The term "second-order measurement" refers to the kind of refined measurement used to optimize systems--to make them cheaper or faster. These are the measurements that are used to tune a machine to its most efficient performance. If you examine the other software engineering books with "measurement" or "metrics" in their title, you will find that they primarily deal with second-order measurement, and in some cases, even third-order measurement.

Although second- and third-order measurements are essential to the development of the software engineering profession, they do not really address the day-to-day problems of the typical software engineering manager. Perhaps the following parable will clarify why I make this claim:

Mr. and Mrs. Tweedle had 15-year-old twins, Dum and Dee, who were just learning to drive a car. Dee had driven the family car ten times, and she had a perfect safety record. Dum had also driven the car ten times, but he had been involved in three wrecks. Mrs. Tweedle told Mr. Tweedle that he had to have a talk with the boy, before someone was killed.

Mr. Tweedle opened the conversation by reviewing the three accidents, and then asked Dum, "What do you have to say for yourself?"

"Well, three accidents out of ten trips isn't such a bad percentage."

"I might agree with you," said Mr. Tweedle, "but your sister, Dee, has also made ten trips without so much as a stone chip on the windshield."

"That's true," said Dum, "but I get much better gas mileage than she does. And I don't get mud on the tires."

"Oh," said Mr. Tweedle. "I hadn't thought of those things. Well, just try to drive more carefully from now on, and keep up the good work on mileage and cleanliness. I'm going to have to talk to your sister about her driving habits."

Even to those of us who have been bamboozled by our own teenagers, Mr. Tweedle sounds like an idiotic parent. But before you judge him too harshly, remember that the story is only a parable. And what is it a parable for? You may recognize the three out of ten figure from software engineering. It's the fraction of large software projects that, among many of my clients, don't ever get finished. Capers Jones, Tom DeMarco, and Tom Gilb have all confirmed to me that this 30% figure is consistent with their experience.

Suppose you were the manager of software development in an organization that had done ten projects, three of which had failed. The software engineering managers from those failed projects can show you precise measurements proving that they produced more lines of code per dollar and fewer failures per line of code than the other seven projects. Would you go to the other managers and talk to them about their managing habits? Of course not.

As John von Neumann said, "There's no sense being precise about something when you don't even know what you're talking about." But that's exactly what many managers are doing when they involve their organization in measurement programs based exclusively on second-order measurement. Bill Silver, one of the gurus of software measurement, said, "It's sad, but true. Most software measurement programs fail."<sup>1</sup>

Silver's observation is confirmed by Capers Jones, whose experience is that 80% of software measurement programs are out of existence within two years. It's also confirmed by my own clients, who fit his description of the first and foremost reason for failure:

“Few companies have a quality culture that provides a positive environment for measurement.”

Let me put it more bluntly. A company that wrecks three out of ten major software projects is not ready for second-order measurement. Even worse, if such a company attempts to install a measurement program based primarily on second-order measurement, they will do more harm than good, and probably wind up with a million-dollar measurement mess. This is not “a positive environment for measurement.”

The data on software quality cultures indicates that at the present time, only a small percentage of organizations have organizations that would support a second-order measurement program.<sup>2</sup> Quality Software Management is designed to help the managers in those organizations improve the quality of their management, so that in turn they may improve their organizations.

The purpose of this volume--First-Order Measurement--is to help those organizations reach a point where they can meaningfully consider the use of second-order, or even third-order, measurement. If you work in an organization that churns out software products on time, within budget, that please your customers and continue to please them over their useful life--and you do this at least 99% of the time--then you won't need to read First-Order Measurement. In fact, if your organization meets these criteria, I would be delighted to refund whatever you paid for this book in exchange for lessons in quality management.

But if your organization doesn't meet these criteria, then I hope to show you how to create “a positive environment for measurement,” to avoid the costly mistakes that have led to failure of so many measurement programs, and how to measure things simply and efficiently--things that will help

your organization consistently produce the quality software you want.

\* \* \*

1. Silver, Bill, “The RPM3 Software Measurement Paradigm,” *Software Quality World* 1991:3,2, 11-15.

2. Humphrey, Watts, D. H. Kitson, and T. C. Kasse, “The State of Software Engineering Practice: A Preliminary Report,” Tech. Report CMU/SEI-89-TR-1, Software Engineering Institute: Carnegie Mellon Univ., 1989.

## Developing Great Software

*Watts S. Humphrey  
Software Engineering Institute  
Carnegie Mellon University*

While most people behave in reasonably predictable ways, software people are unique, both in our creative abilities and in the nature of the work we do. Software professionals are among the brightest people on earth. Most of us got into this field because we were excited by the thrill of working with a challenging and very special technology. However, the problem many of us face is that the environment in which we work rarely supports and motivates consistently high-quality

creative work. Much as in other professions, the performance of software people is governed by four things.

- our understanding of the job we have to do
- our knowledge of and skill at using the best known methods for that job
- our discipline to properly and consistently use our knowledge and skill
- the quality of the support system that motivates and controls our work

**People Principle Number 1: If we do not understand the job we have to do, we will not do it very well.**

This seems to be such an obvious point that it is hardly worth discussing. However, it is critically important and often overlooked. If the members of a development team are not intimately familiar with the job their product is to perform and the way the users of that product will use it, the project will almost certainly be troubled and the product will likely be a failure. If you can't get users or people with user experience on your development team, at least insist on ready access to people with such knowledge. To produce quality products, you must have a close and cooperative relationship with user experts.

**People Principle Number 2: When we know and use the best methods, we do the best work.**

While software developers usually get extensive training on tools and methods, they generally get little or no guidance on their personal practices. This is not true of any other sophisticated technical field. Chemical engineers are not born knowing how to conduct experiments, analyze the composition of materials, or follow sound laboratory practices. Doctors learn their profession through extensive training, by completing several years of internships and specialty studies, and by mastering the methods that their predecessors have found most effective.

If every scientist had to personally discover Bernoulli's principle, develop Maxwell's equations, or invent calculus, we would still be in the dark ages. The explosive growth of science and engineering didn't start until people defined their practices, measured their work, and communicated precisely. This allowed others to repeatably produce the same results. The essence of science and engineering is learning from the experiences of others. Until we build a body of professional software practices and teach new professionals to consistently and properly use these practices, programming will remain an unsatisfactory craft that produces defective, insecure, and even unsafe products.

**People Principle Number 3: When we know how to select and consistently use the best methods, we do extraordinary work.**

The SEI now has data on several thousand programmers who have taken the Personal Software Process (PSP)<sup>SM</sup> course as

<sup>SM</sup> Personal Software Process and PSP are service marks of Carnegie Mellon University.

well as data on over 100 Team Software Process (TSP)<sup>SM</sup> teams that have been launched<sup>1</sup>. Developers can learn and use highly-effective personal practices and when they do, they produce much better work than they ever did before. In a recent study of 20 TSP teams, their finished products had 100 times fewer defects than average industrial products and 16 times fewer defects than typical products produced by CMM<sup>®</sup> level 5 organizations [Davis]. Team productivity is also increased by an average of 78%.

With the PSP, professionals learn how to follow a defined process, how to modify that process when they need to, and how to measure and plan their personal work. By using their own data, developers can see what methods work best for them and they can make informed decisions about how to do their work. Then, when TSP teams are launched, they examine the job that they have to do and decide on the best strategy, process, and plan for the work. In doing so, they consider their personal skills, abilities, and interests; the work that must be done; and how they can best work together as a team. This generally ends up being the best way for this team to do this job.

#### **People Principle Number 4: When we are highly motivated, we do the best work.**

When people are discouraged, antagonized, or even just unhappy, they cannot do their best work. The key to getting superior work from creative people is to energize and motivate the entire team. But what motivates software people and how can one build and sustain this motivation? In an interesting study of software projects, Linberg compared management's typical views of project success with those of the team members [Linberg]. While managers typically think in terms of cost, schedule, and product success, the four factors team members considered most important were as follows.

- a personal sense of being involved and making a contribution
- frequent celebrations where the team and management complemented them on their achievements and milestones
- positive feedback from marketing and senior management
- the autonomy to do the job the way that they thought was best

#### **Conclusion**

These are the things that motivate us. While these same factors motivate people in almost all walks of life, they are particularly important for superior software work. In software, much of our work is intellectual, so for us motivation, personal discipline, and sound professional behavior are criti-

<sup>SM</sup> Team Software Process and TSP are service marks of Carnegie Mellon University.

<sup>1</sup> TSP team projects start with a launch where the members learn the project requirements, define their own processes, and develop and negotiate their plans with management.

<sup>®</sup> CMM is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

cally important. This is why contributing, being involved, being rewarded, getting positive feedback, and having autonomy are essential for consistently superior work.

Whether you manage software development or develop software yourself, remember and follow these four principles. To produce superior products, the developers must

- understand the job the product is to do
- know the best methods for doing the work
- consistently select and use these best methods
- be highly motivated to work on this team doing this job

#### **References**

Noopur Davis and Julia Mullaney, "Team Software Process (TSP) in Practice," *SEI Technical Report CMU/SEI-2003-TR-014*, September 2003.

Kurt R. Linberg, "Software Developer Perceptions about Software Project Failure: a Case Study," *The Journal of Systems and Software*, 49-(1999) 177-192.

## **Roundtables**

### **Roundtable Discussions are a Networking Opportunity**

**By: Chuck Anastasia, Roundtable Chair**



Where can I find professional networking opportunities to exchange work experiences, and find fresh ideas that apply to real issues in my job? How do I get to get acquainted with other software professionals outside my company

who can offer a different perspective on technical, management and career development issues? The roundtable sessions that precede our regular speaker at the monthly Boston SPIN meetings are a great opportunity to exchange ideas with other software professionals. Many SPIN members are aware of the high caliber speakers presented at each monthly meeting, but have never attended the smaller roundtable sessions at the beginning of each meeting.

Each month, SPIN offers small roundtable discussions on up to four topics. These sessions last 45 minutes, running from 6:00 to 6:45 pm providing time for a short break before the main meeting begins. Each session is coordinated by a volunteer facilitator who provides some information on the topic and encourages participants to contribute their ideas. Attendance at roundtable sessions is usually about 10 people. The discussions are lively and interactive. For insight into past roundtable sessions, look for the summaries in current or past issues of our newsletter, In-the-SPIN. Check the meeting announcements or the roundtable web page for a current schedule and description of Roundtable topics.

Boston SPIN has a Commercial-Free Policy to assure that the roundtables remain an open discussion and exchange of ideas

and practices related to the software process. It is not a vehicle for marketing commercial products or services. SPIN is always looking for volunteers to facilitate roundtable sessions. Please consider volunteering to facilitate if you have a topic you would like to explore with a small group of software professionals. Look at our roundtable web page for a list of requested topics or suggest a topic of your own. For more information, contact the roundtable coordinator listed on the Boston SPIN contacts web page.

**September 27, 2005**

## **Problem Definition and Resolution**

*Facilitator: Dolores McCarthy, Quality Manager;  
Computer Sciences Corporation*

The focus of the roundtable was to explore the development of a problem definition before attempting to resolve a problem. The roundtable began by soliciting a list of problems the six participants might experience at work. "The software doesn't work." "The software doesn't work as expected." "There were missing, incomplete, or contradictory requirements." "Help is not effective." "There were no release criteria." "They don't know what the software is for (marketing)." "Don't know what the path is to get to a desired result." The next step was to examine the problem statements one by one and see if they were sufficient to begin to solve the problems. Most agreed they were too vague. The group began to explore how to expand the statements to be more definitive and objective. Then analysis of the problem could begin. The group suggested asking questions to get clarification, more specific information, and the point of view of the person with the problem. For example, "The software doesn't work" could be expanded by asking, "What were you attempting to do?" "What menu were you using?" "What happened?" "What were you expecting the software to do?" "What time of day was it?" "What was the impact of the problem for you?" Answers to these and other exploratory questions can establish the number of people having problems with a particular software function, the time periods for problems, and how bad the problem is (impact) for people. This helps those who are going to analyze the problem have enough information to start finding the reason for the problem occurring and resolving it.

Someone stated that at their company a team was first formed to be responsible for a problem. The team had specific steps for solving problems and found the impact could be exponential. After success, the team was congratulated.

To expand the idea of forming problem statements, the facilitator asked the group to think about problems in their everyday life. They offered "Lost luggage," "I'm misunderstood," "Traffic jams," "Icy roads," "Empty nesters," "Stress," "Tree fell down in my yard," and "Crabgrass." There was not time to address all the items, so "Lost luggage" was used as an example. From an individual's point of view, the number of times an airline lost or delayed their luggage and the impact on them would affect the way the person resolves the prob-

lem. The group mentioned aggravation, inconvenience, having to buy new clothes, and time wasted waiting in the lost luggage area of the airport or waiting at home or the hotel for its return. This could affect their decision of which airline not to fly again or whether to take only carry-on luggage (problem resolution). From an airline's point of view, the numbers of lost luggage could impact their rating in the industry and affect revenue. Their problem statements could include, besides numbers of pieces of luggage lost or delayed, the terminals where it happened, times of year, times of day, and other information necessary to begin to solve or lessen the problem.

The facilitator provided a handout of references about problem solving and a Problem Statement worksheet to list

- 1) Problem (in measurable terms)
- 2) Impact
- 3) Goal
- 4) Criteria for success
- 5) Resources
- 6) Solution date

From Philip Crosby's book, *Quality Improvement Through Defect Prevention*, Philip Crosby Associates, Inc., Winter Park, Florida, 1985.

### **Resources:**

#### **Warning: Your Development Team Isn't Working on Development**

Sept 05, '05

Yochi Slonim

[http://www.sandhill.com/opinion/daily\\_blog.php?id=25](http://www.sandhill.com/opinion/daily_blog.php?id=25)

#### **A Tour of QTMS: Solving Problems**

#### **The 8D Problem Resolution Process...**

[http://www.reliasoft.com/enterprise/qtms\\_ov4.htm](http://www.reliasoft.com/enterprise/qtms_ov4.htm)

#### **Call Center Solutions**

August 1998 by Sperry Steven

"Selecting Problem Resolution Software to Meet Current and Future Support Needs"

[http://www.findarticles.com/p/articles/mi\\_qa3877/is\\_199808/ai\\_n8812123](http://www.findarticles.com/p/articles/mi_qa3877/is_199808/ai_n8812123)

#### *Running an Effective Help Desk*

Barbara Czegel, John Wiley & Sons © 1998

"Be thankful for problems. If they were less difficult, someone with less ability might have your Job."

*Creating a Lean Culture: Tools to Sustain Lean Conversions*, David Mann, Productivity Press, 2005

Chapter. 8: Solving Problems and Improving Processes - Rapidly

**October 18, 2005**

## **Managing Outsourced Teams from Male Dominated Cultures – challenges for the female managers**

*Facilitator: Jacqueline S. Luciano, CCP, PMP*

The following notes express topics discussed at the Roundtable.

Definitions of Male dominated cultures:

- Male dominated cultures could be in different regions in the US (e.g., mid-West vs. Northeast or Pacific Northwest)
- Male dominated cultures include some corporate cultures, regardless of the location of the corporation
- The type of business can also be of influence. For example, consulting services males dominated
- Some times attitudes towards females are not what we necessarily expect from someone coming from a particular country
  - People from cultures where mothers are revered, may be more accepting of female bosses (e.g., Latin America)

Outsourcing has inherent challenges:

- Exchange of information
  - Perhaps in-house projects should have the same level of documentation for in-house projects
- In the SQA area:
  - it becomes more difficult to do requirements tracing when the teams are working remotely
  - test automation not a solution
  - It is harder to determine people's body language if they are not with us

Outsourcing to different countries has challenges unrelated to attitudes toward women:

- language can be a challenge
- body language/ mannerisms are different
- time differences – hard to find a time when both parties are in the office when there is a 7 hours, 10.5 hours, or 17.5 hours difference – at least one of the parties is working outside of “normal” business hours
- what is acceptable behavior is different in other countries
  - places where want ads list that they are looking for “Male, 25 to 35 years old, ...” are legal
  - places where they don't give a second thought to a corporate computer with a wall paper of scantily clad women
  - places where kissing on the cheek is the way that you are introduced to somebody else, and where men kiss each other on the cheek
  - places where terms that are considered insulting and derogatory in the US are not considered offensive

Men and women are different:

- men are generally socialized to be competitive

- men learn to compete in sports
- more aggressive behavior is required to succeed
- should women learn sports to help them?
- women are generally socialized to be nurturing
  - women try to build teams
  - women need to strive for balance
- the same behaviors are perceived differently for men and women
  - what is perceived as assertive for men, is labeled pushy for women
  - what is perceived as emotional for women, is labeled as pride of ownership for men
  - what is perceived as consensus building for women, could be perceived as weakness or indecisiveness in men
- improvements have been made in the last 10 years to foster a more equitable environment for both men and women
- negotiation is one of the hardest challenges for women
- men and women have differences in their speech (reference: You Just Don't Understand: Women and Men in Conversation by Deborah Tannen)
- The key to success is learning to communicate with each other across gender and cultural lines:
  - we need to work on improving our communication
  - we need to learn to deal with each of the different personalities in our teams
  - we need to find a way to engage people

## **Software Reliability Prediction**

*Facilitator: Nihar Senapati, CRE, CQE, Six Sigma Black Belt (ASQ); Senior Reliability Engineer, Avici Systems Inc.*

The group in this roundtable was interested in understanding the methodologies for reliability prediction and how predictability could help them decide on the maturity of the software and releasing the software to the field/customer.

Some of the participants were conducting a reliability prediction process while some were embarking on setting up the infrastructure for this process. Some participants were interested in understanding the methodology better in order to move forward to make this process a determinant of their release schedule.

The scope of the prediction process was discussed. It was emphasized that an estimate of the defects in the software could be started in the ‘Requirements/Code’ phase. The defect estimate might not be representative of actual reliability of the software; however, it could be refined over time as the software organization moves from ‘Requirements/Code’ phase to ‘Test’ phase. Some of the models developed by Rome Lab's or Ida could be used in the development phase. However, as we move towards testing, actual defect data can help predict the defects over a time frame. Several prediction models were discussed including Jelinski-Moranda, Littlewood-Verrall, Musa Basic, Musa-Okumoto, Schneidewind and Yamada S-shaped. One key requirement for using any of the aforesaid models is to establish “severity” definitions (levels 1, 2, 3, 4) for the bugs. Participants discussed in detail which “severity”

level defect data (levels 1, 2, 3, 4) needed to be collected. It was decided that what was collected rested solely on the supplier's commitment and the customer's requirements. The scope of the project's features that forms the basis of the prediction process needs to be pre-determined. Testing will be conducted per the operational profile. Recording of the failures also needs to be noted within the test start and end points.

Some of the popular software tools for reliability prediction like CASRE, SMERF and SWEEP were discussed. CASRE is a popular tool used across the industry, developed at JPL, while SMERF is developed at NSWC. Both tools analyze the defect data through a specific input format and use the prediction models to generate reliability parameters such as reliability, defect estimate, etc.

A sample output of the CASRE tool was shown and its relevance to the release criteria was discussed. The participants also discussed the applicability of the prediction process to the SW-CMM Level 4 Key Process Area.

One of the focal points for discussion was how the residuals can be established in terms of gap between the actual defects and predicted defects at a certain time frame and how field Mean Time Between Failure (MTBF) could be calculated.

**November 17, 2005**

## **Things to Consider Before Automating Tests**

*Facilitator: Jean-Pierre Boissy*

The discussion began by suggesting that automating tests could translate into increased cash for the company. Automation could contribute to a faster time to market, faster identification of real world defects, and less post-ship maintenance costs. The roundtable felt that sometimes automating tests were the only way to emulate real user loads and gave the example of performance scalability. Another benefit was the ability to retest code "at will" as there would be instances when previous results would become invalid due to requirement and system changes. The conclusion was that today automation is not likely a "hard sell" to get monies from management.

Investment in automation needs to be justified by clearly showing what could be automated. Begin by making it clear that automated testing is not a panacea but that it is an effective means of testing specific areas like 'smoke', performance, regression, API, or boundary tests. The roundtable did not feel that unit testing should be included in the automation strategy as they were written by developers to check their own work. The conclusion was to have the first automation effort be a "smoke" test as a proof of concept.

Costs were broken down into five types: opportunity, purchasing, hiring, training, and support costs. Opportunity costs were the time that would otherwise be spent doing manual testing. Purchasing related to software test tools. Hiring and training people related to writing the automation, consultant costs, and the cost of buying supporting software and hard-

ware. Time investment included researching the areas of the product to be automated, tool selection, making buy or build decisions, designing tests, building prototypes, and testing the automation. Determining who would develop, run, and maintain the automation was addressed. The roundtable felt that resources could be best managed by starting with an automation leader or consultant, focusing on training the trainer, registering people in classes, reinforce shared learning, and approaching test automation as a programming project.

Automated tests requires specifications which map to test plan use cases, be measurable, have provisions for dealing with exceptions, and each test must begin from a known clean state. The roundtable cautioned that use case mapping would not be enough because this could miss the problems related to system functionality. The best approach would be to combine tests into system level tests. They agreed that specific tests should be measurable and also pointed out that certain test could not be automated and gave the example of testing race conditions.

The roundtable stated that products should be testable and developers can help by designing testable code from the onset of a project. They could build "hooks" into their code to facilitate automation making calls to the product and monitoring its functionality. Additionally developers can act as great mentors helping with sharing coding best practices and guiding the building of effective test cases.

**December 13, 2005**

## **Defect Triage Techniques**

*Facilitator: Erik Hemdal, Software Engineer;  
Comprehensive Power, Inc.*

There's no one best way to triage defects. Everything depends on the situation and the nature of the issues at hand.

What does it mean to "fix a bug"? There are many ways to resolve a defect, among them:

- Fix the bug
- Ignore it and hope it goes away
- Change the operation of the software (example: avoid a crash by automatically rebooting overnight).
- Modify the requirements to sidestep the bug.
- Decide not to fix the bug for any of various reasons.
- Drop, or defer, a feature if a defect means that it cannot be ready in time. Better to defer (or make "experimental") a feature, rather than release it when it is not truly ready.

Should you have a dedicated "bug-fix" team or schedule defects into new development work? Much depends on the nature of the software and the staff available.

How to decide what to fix? One idea is the "open-heart surgery" idea. Once you start working in one area of a software system, fix all the defects you can. This can save money, considering the generally high fixed cost of regression testing

(costs the same whether you fix one or 100 defects in the same code).

When do you drop a defect from consideration? We discussed the “three strikes” rule. If a defect has been deferred three times – that is skipped for three major releases – it is a drop candidate. Anything not fixed after three chances likely is an unimportant issue.

More and more teams are using daily builds and regression tests. We talked about the “red flag” status example. A team used a large flag to indicate success of the overnight build: green indicated success and red indicated failure. It got the word out to the development team quickly and dramatically.

We were generally in agreement that arbitrarily lowering the severity of a defect is a dangerous practice; it may help to also define a priority for repair. Although a high-severity defect may exist, this allows other high severity issues to be tracked ahead in the repair queue.

In any event, triage and priority decisions must be documented, in the defect tracking database if at all possible. Stakeholders must be aware of the triage decision -- and the reasons behind it -- for each issue.

Two kinds of triage:

- Ongoing triage as defects arrive during development.
- Cleanup triage of deferred issues, done as part of planning for a new development effort.

How to triage: One unique suggestion was the “Beta-test Fest”. The entire team participates in a serious test effort. Finding three or more defects entitles you to join the “triage party” after the test. At this gathering, participants discuss what will be repaired and what is planned for the next cycle of work. Project manager is also available to resolve differences and make final decisions, so there is a single point of accountability.

It’s often an option to discuss a defect in release notes if nothing else can be done about it. But “putting it in the release notes” is a very poor practice unless your stakeholders and customer are well aware. There are few things worse than being surprised by a serious defect in release notes.

## **Control mapping for cross purpose audit and performance measurement**

*Facilitator: Robin Basham, CEO and Owner of Phoenix Business and Systems Process, Inc.*

This was a hands-on live demonstration showing Project Management and Software development flow diagrams that included common application and IT controls.

This discussing began with a high level overview of some of the primary regulations currently impacting the need for adopting control frameworks. Among the U.S. and international laws and standards discussed were: SAS 70, SOX, OMB A-130, BASEL II, FISMA, Computer Fraud and Abuse Act, and Visa's PCI Data Security Standard.

One of the main insights offered was that no matter what the regulatory requirement, efforts by Project Managers and QA have ever greater potential for reuse than ever before.

Also discussed was the importance of aligning controls requirements to every aspect of process documentation. A demonstration was given for using Visio as a dynamic tool for demonstrating process. The importance of storing metadata for departmental application of processes, quality measures, and controls was emphasized. The applicability of Object Oriented Programming (OOP), Business Object Model (BOM), stencils, and schemas were discussed. As an example, demonstrating evidence of instance, change, and incident tracking processes were shown.

The high point of the evening was being joined by Capers Jones.

## **Dear SPIN Doctor**

Dear SPINners:

It’s a new year and I was thinking about new topics to get us going again. I’d like to discuss the benefits of using Peer Reviews or inspections and where they benefit the software process the most. If any of you have experience that you would like to share with our readers on peer reviews and/or inspections, please send your comments to me.

Send any questions you have on any software areas and, if the SPIN Doctor can’t answer them, she will find an expert who can! Review some of the old columns and see who the guest columnists were!

Remember, this column is for you; let’s make a difference! Send your comments and questions to “Dear SPIN doctor” at [brodman@logos-intl.com](mailto:brodman@logos-intl.com). Sign them or use a “pen-name” -- I respect your confidentiality!!



the SPIN Doctor

## SPIN Information

The Boston SPIN is a forum for the free and open exchange of software process improvement experiences and ideas. Meetings are usually held on third Tuesdays, September - June. Boston SPIN welcomes volunteers and sponsors. There is no charge to attend the meetings. Additional information about the Boston SPIN can be found at our Web home page: <http://www.boston-spin.org>.

For more information about our programs and events contact Donna Johnson, Program Chair, [johnson@logos-intl.com](mailto:johnson@logos-intl.com).

### Cancellations (including weather)



Starting at 3pm, we'll notify you via email to the SPIN distribution list, we'll post the notice on the SPIN web page, and we'll send the cancellation announcement to Channel 7 TV.

### SPIN Meeting Location

Boston SPIN meetings are held at The MITRE Corporation in Bedford where they are located at 202 Burlington Road (Route 62), Bedford. Directions can be found on our Web site: <http://www.boston-spin.org> or on The MITRE Corporation Web site: <http://www.mitre.org>.

Please be aware that MITRE has advised us that, due to increased security concerns, you will need a Picture ID for admission to the SPIN meetings. We encourage you to leave all carrying bags, backpacks, and briefcases behind (i.e., in your car). Otherwise, you should be prepared to have these opened and inspected upon arrival.

### Our Sponsors

Our appreciation goes to all our generous sponsors for supporting our SPIN activities!!



**The MITRE Corporation** who provides our meeting space.

#### Chaco Canyon Consulting

<http://www.ChacoCanyon.com>

Chaco Canyon Consulting works with people in problem-solving organizations who make complex products or sophisticated services that need state-of-the-art teamwork, and with organizations that want to achieve high performance by building stronger relationships among their people.

#### AccuRev, Inc.

<http://www.AccuRev.com>

AccuRev provides award-winning, stream-based software configuration management tools that provide both the flexibility and built-in best practices that developers need to manage today's complex, parallel, and distributed software development projects.

#### Microsoft Technology Centers (MTCs)

<http://www.microsoft.com/mtc>

### The MathWorks

<http://www.mathworks.com>

The MathWorks, founded in 1884, is the world's leading developer of technical computing software for engineers and scientists in industry, government, and education. With an extensive product set based on MATLAB and Simulink, The MathWorks provides software and services to solve challenging problems and accelerate innovation in automotive, aerospace, communications, financial services, biotechnology, electronics, instrumentation, process, and other industries.

The MathWorks employs more than 1,000 people worldwide, with headquarters in Natick, Massachusetts. The MathWorks is currently recruiting company-wide. For career listings visit [www.mathworks.com/company/jobs/](http://www.mathworks.com/company/jobs/).

### WorkingInUnison

<http://www.workinginunison.com>

If your organization is interested in sponsoring the Boston SPIN, please contact SPIN Steering Committee member Bruce Taylor at [info@workinginunison.com](mailto:info@workinginunison.com) or ask any Steering Committee member for a sponsorship brochure.

### Email Lists

To receive Boston SPIN specific notices, send an email to:



Jim Withall, Boston SPIN membership chair, at [withall@rcn.com](mailto:withall@rcn.com).

To receive Boston SPIN-Plus specific notices, send an email to: [spin-plus-request@boston-spin.org](mailto:spin-plus-request@boston-spin.org), include "subscribe" in the message body.

To post an announcement on the SPIN-Plus list: [spin-plus@boston-spin.org](mailto:spin-plus@boston-spin.org)

### Newsletter Call for Articles

The *In-the-SPIN Newsletter* is always in need of new and interesting articles dealing with process improvement, software development methodologies, project management and other related subjects that may be of interest to our readership.

Please send general correspondence or articles that you would like to have considered for publication to:

- Judi Brodman, Editor of In-the-SPIN at [brodman@logos-intl.com](mailto:brodman@logos-intl.com)

Back issues of the *In-the-SPIN Newsletter* can be found on the Boston SPIN Web site: <http://www.boston-spin.org/>.

## Upcoming Meetings

Date	Speaker	Topic
1/17/2006	Michael Mah	Excess Friction: How Fast Deadlines Can Slow You Down & Ruin Your Life
2/21/2005	TBA	TBA
3/15/2005 Joint SPIN/ASQ meeting	Steve Anthony	Six Sigma
4/18/2005	TBA	TBA
5/16/2005	Panel	Offshoring
6/20/2005	TBA	TBA

## Quotes



“Process is like fire. You can either warm your hands with it, or burn down your house.”

*IBM staff member*

