

# The Menlo Briefs

Vol. I No.1

Newsletter of the Menlo Institute

Fall 2002

## The Software Factory — Beyond XP by Tom Meloche

Menlo strongly promotes the practices of Extreme programming (XP). XP seeks to solve the problems of failed software projects by working with, not against, the nature of developing software in a world of constant change. However, in our estimation XP isn't enough. We strongly promote using additional practices not part of XP to build a truly functional delivery team. We call our extended development process The Software Factory (TSF).

TSF is an agile software development process that, unlike XP, describes the complete delivery team as well as work environment. Where XP is primarily focused on just the development team,

TSF describes multiple teams required to close the loop between the project sponsors, users and developers. The Software Factory delivery team has four distinct specialties: Voice of the Stakeholders, Voice of the Users, The Coach and The Development Team.

These specialties leverage different best practices from multiple sources including Extreme Programming, Six Sigma, Interaction Design and even The Unified Process. Only the development team hires programmers and trains them specifically in XP. The other three specialties have different highly skilled members trained in practices not de-

scribed in XP and not necessarily directly related to programming.

We are not abandoning XP, far from it. We have seen its practices work too well too often to give up on a good thing. Instead we are building on XP, to better capture the reality of developing software in a world of multiple users and multiple stakeholders. We believe The Software Factory is the next evolutionary step in software development and a welcome addition to the XP family. Learn more about The Software Factory in Menlo Institute's classes *Secrets of Software Success* and *Building a Software Factory*. □

## Six Sigma Software by Tom Meloche

Six Sigma is a disciplined process, developed at Motorola, to help business focus on delivering near-perfect products and services. Sigma is a statistical term that measures deviation.

In Six Sigma, a formal process of measuring "defects" is employed to

systematically eliminate them. The goal is to statistically approach "zero defects."

Menlo Institute's new course *Six Sigma Software* describes how Six Sigma may be applied directly to creating software. It teaches how to build a delivery team that moves

beyond "entitlement" and delivers an order of magnitude increase in developer productivity. This course, a special adaptation of *Secrets of Software Success*, has been created specifically for students who are already versed in Six Sigma practices. □

### Contents

<i>4 Days with Dr. Deming</i>	2
<i>Waste</i>	2
<i>Please Stop!</i>	3



### The Menlo Extended Development Process is The Software Factory



### Six Sigma: Statistically Approaching "Zero Defects"

## Waste by Tom Meloche

The MIT *Technology Magazine* August issue was dedicated to the topic *WHY SOFTWARE IS SO BAD*. The feature story contained the observation that “software projects often devote 80 percent of their budgets to repairing flaws they themselves produced.”

The observation is stunningly profound and ties well into a Menlo Institute suggestion of identifying and eliminating waste.

Lean Manufacturing for years has studied and categorized waste. To help people recognize waste Taiichi Ohno created the following waste

categories: Overproduction, Excessive Inventory, Waiting, Transportation, Processing, Unnecessary Motion, and Making Defective Products. Devoting 80 percent of your budget to repairing flaws you created would be categorized as the waste of Making Defective Products.

How much is waste costing your organization? Do you have ways to identify, track and eliminate waste? The waste categories themselves are worthy of further study. Manufacturing quality practitioners have understood for years that *excessive inventory* is waste and antithetical to

quality. However, in the software industry, it is common to find entire projects dedicated to the production of excessive inventory, typically in the name of producing reusable logic.

Every time we have encountered one of these projects it is always a failure. A manufacturing quality practitioner is not surprised by the result because they have studied waste and its impact on production.

If you do not want to be surprised with regard to your projects, consider studying the lessons of manufacturing, specifically study waste. □

**“Software Projects often devote 80 percent of their budgets to repairing flaws they themselves produced.” - MIT**

## Four Days with Doctor Deming by Tom Meloche

Variance is the enemy of quality. Quality cannot be inspected into a product; it is an attribute of that product. These simple ideas are part of the foundation of modern manufacturing quality practices as taught by Dr. Deming. Unfortunately, a wide study of software engineering literature produces little evidence that either variance or inspection are understood by software devel-

opment professionals. We have never seen the principles of common and special cause variation taught in the field of software engineering. That is about to change.

Dr. W. Edwards Deming's insights into manufacturing, that heralded the rise of modern Japan, are directly applicable to software. Software professionals can directly implement his fourteen

points for management and leveraging his insights may be the best way to solve the software industries' current crisis in quality and productivity.

To encourage the study of Dr. Deming, the Menlo Institute is presenting the first video of the series “Four Days with Dr. Deming” at the Menlo Institute on November 20th from 6pm to 8pm. Admission is free

and pizza will be served. Please register at : [www.menloinstitute.com](http://www.menloinstitute.com). □



W. Edwards Deming

## Please Stop! by Richard Sheridan

"We tried XP and it didn't work."

We heard this comment a few times in the past year. Every time the conversation is almost exactly the same.

"We've been doing XP for a few months, but it doesn't work for us. I think our organization has some unique challenges that prevent XP from working."

At this point, hoping to be helpful, we try to find where the problems may lie.

"So are you finding paired-programming to be a difficult adjustment?"

"Oh. We're not doing paired-programming. A few of our team members pair every now and then, but most haven't tried it. Our developers are uncomfortable pairing and feel it would slow them down."

"Really? That's interesting. How was it to move to an open and collaborative workspace?"

"Well, none of us have actually moved out of our cubes, but we had our facilities guy rearrange some of the cube

desktops so it would be easier to have two people sit at a single computer."

"Have you found that the daily stand-up meeting has helped improve communications?"

"Well, hardly anyone ever shows up for our daily stand-up meeting. Some people are on a different floor and no one takes the initiative to call the meeting since everyone looks busy all the time."

"So, do you know how many unit tests you have running?"

"We wrote a few unit tests at the beginning of the project, but it was hard and they run too slow so we stopped. We pretty much had to focus on getting the work done."

"How long are your iterations?"

"Our next release is scheduled in 6 weeks. We used to only release every six months."

"But how long are your iterations?"

"Well, let's see, we started the project at the

beginning of last month and we've scheduled a release in six weeks. So I guess our iteration is about 10 weeks. Is that too short?"

"No, actually we recommend two week iterations at the longest."

"Oh. I think our projects are too complicated for releases that often."

"I see. Who participates in the planning games at your company?"

"Oh, we've done really well here. The programmers sat around a big table at the beginning of the project and wrote all the story cards we thought we'd need for the project. Then we estimated them. We were able to reject a lot of stories that were just too hard or we thought could wait until a future release. We ran the remaining stories by the "customer" and they thought it looked like a good plan."

"Does the customer come by and check on progress every now and then?"

"We don't have anything to show them quite yet. We're mostly doing infrastructure right now."

"So, let me summarize. You're not pairing, you're not in an open and collaborative environment, you're not writing or running

automated unit tests, the programmers are writing the story cards and you have only one iteration. So what part of XP are you doing?"

"Oh, we like how XP says don't write documentation. We don't document anything anymore."

"May I be blunt with you?"

"Uhhhh, sure. I'd love to get feedback on how we're doing from someone whose done this before."

"Please stop. Please stop saying you are doing XP or have ever done XP. XP is a collection of practices that must be taken together. You're not doing XP. You've never done XP. You're doing everyone a disservice by saying you are doing it when you're not. If you'd like we'd be happy to help you setup a real XP environment."

"Oh. I guess you're right. Sorry." □



If your office doesn't look like this, then you're probably not doing XP.