

# most significant bits

News and Information For High-Tech Professionals

Published by Pocket Protector Press™

A division of Stout Systems



Fall 2006

## In This Issue

### Talking Technology too Early

When consultants ask the right questions, they often discover that the problem they were hired to solve is not the *real* problem.

### The Tablet PC Revolution

The Tablet PC under .NET is easier to program and more fun than other platforms—even easier and more fun than .NET alone!

### Recent News

Stout Systems welcomes new employees David Blake and Martin Shoemaker (Software Engineers) and Theresa Kraft (Project Manager).

### Job Openings

Check out current permanent and contract openings on our Web site [StoutSystems.com](http://StoutSystems.com).

### Current Candidates

A sampling of candidates we represent is available on [StoutSystems.com](http://StoutSystems.com).

### Subscribe To Our Newsletter

If you would like to receive this newsletter directly at your desk or in-box, send a note to [info@stoutsystems.com](mailto:info@stoutsystems.com) providing your subscription info (e-mail or ground mail address).

### Call us:

(734) 663-0877

## When the problem is not the problem

# Talking Technology too Early

by John Stout

I have said it before and it's well worth saying again: in the high tech business, your success isn't determined by which languages, technologies or methodologies you embrace or evangelize. Those things are subject to marketing hype and industry fads.

Your success is in how you identify and solve the real problems faced by your customers. Sometimes the problems presented by customers are only the consequences of the "real" problem.

You could sum up the issue as the problem the company is trying to solve isn't the problem. A corollary of that would be if that problem isn't resolving, it's probably not the real problem.

Here are some examples based on real life challenges:

1) A new customer was creating a Web site and had done significant front end design. Implementation was started but went quickly into a tailspin. The customer came to us for consulting to ensure the development was being done properly. But when we stepped in we realized it wasn't more expertise in development that they needed. We identified a missing step in the process: they really needed to do a technology proof-of-concept because it was never determined up front whether or not what they wanted to do could actually be done. The proof-of-concept demonstrated a realistic and less costly way to implement the site, and that led to a straightforward implementation.



2) Another customer came to us with an employee retention issue. They had gone through developer after developer on a project and the CTO was at his wits end trying to find someone who would be "loyal". The issue we found was not one of loyalty; the job really demanded a jack-of-all trades, that is, someone who could write code, anticipate technology trends, manage the project, and communicate effectively to non-techies. The previous developers were actually leaving in self-defense, overwhelmed by the real responsibilities. So we fixed that by finding someone with technical, management and presentation skills.

3) One other customer brought us a hiring requirement for a person with good development skills to handle their quality issues. In that case, it turned out that what they really need was to introduce a slightly more formal software development method and introduce a separate quality assurance cycle.

Software project problems are almost never,

*continued on page 4*

## The Tablet PC Revolution

by Martin L. Shoemaker

**C**ue up the Buffalo Springfield:  
*There's something happening here.  
What it is ain't exactly clear.*

The rest of that classic tune is ominous and foreboding, when I'm really feeling just the opposite right now. And the reason why I'm in such an upbeat mood is the Tablet PC. And like the song says, there's something happening here, and what it is ain't exactly clear – yet. But it sure is exciting!

When I got my first Tablet PC, I couldn't explain to people why I wanted it – indeed, why I put up with a two-month backlog to get it. “It's a laptop with a pen—but without as much memory or processor. Big deal,” they said. And these weren't just any people; these were fellow programmers and even extreme gadget freaks. But I just knew they were wrong. Somehow, this was more than a laptop with a pen.

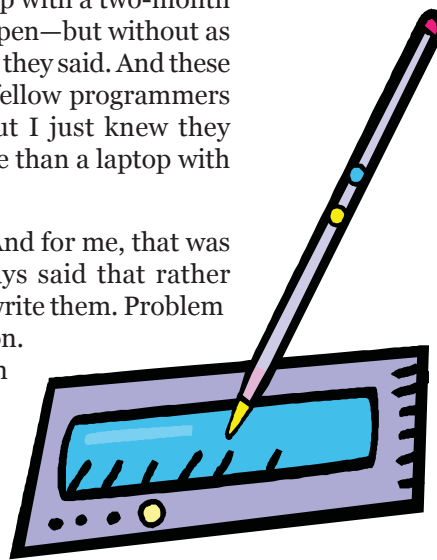
And then I started programming it. And for me, that was where the real fun began. I've always said that rather than play video games, I'd choose to write them. Problem solving isn't just my job—it's my passion. Given a computer, I would program even if nobody ever paid me a dime for it (don't tell my clients!). And programming .NET is even more fun. And programming the Tablet PC using .NET is F-U-N.

### The Microsoft.Ink Namespace

What makes the Tablet PC under .NET easier to program and more fun than other platforms (even easier and more fun than .NET alone) is that it adds one more namespace that's right up there with the core .NET namespaces in terms of power and flexibility. And that new namespace, Microsoft.Ink, provides new and powerful user interface metaphors. Right now, our common user interface metaphors are Console (keyboard) and Forms (keyboard and mouse). Well, you can use Ink for the Forms metaphor; but Ink also adds an electronic paper metaphor, or as I like to call it, the Smart Cocktail Napkin metaphor. It's the ease of use of paper combined with the ease of editing of electronic documents, plus you can add the ease of interpretation that lets the computer parse meaning from the drawing. It provides controls and classes for capturing

and rendering hand-drawn Ink, including attributes such as pen color and size and shape; and then it further allows you to interpret that Ink in terms of text and shapes and gestures. These elements form the basis of the Smart Cocktail Napkin user interface metaphor.

To get an idea of what the Microsoft.Ink user interface paradigm is like, you can download a demo for the Tablet PC called the Tablet PC Composition Tool (which runs on Microsoft Windows XP Professional Tablet PC Edition, as well as all versions of Windows Vista). The Composition Tool shows you musical staves and lets you write in notes, edit them, erase them, and so on. And then you can play those notes, with different digital instruments assigned to each part. Even within its limitations—such as no undo or clipboard functions—it's a compelling user interface paradigm for people who know music: they just start using it, without instruction. You write like paper, you erase and edit like a word processor, and the computer interprets and plays based on what you write.



And that last part is key. Without the interpretation, it's “only” electronic paper: easy to draw, easy to edit. That's pretty easy and fun by itself. I have to steal my Tablet PC back from my nieces, because it's the most fun coloring book they've ever had!

But with interpretation added, it's the Smart Cocktail Napkin: you get a good idea, you jot it down, and the computer acts on it.

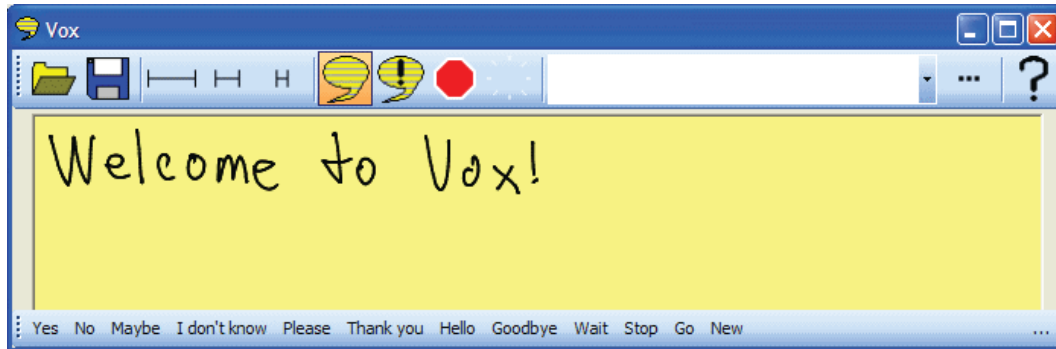
### Accidental Software?

My excitement reached a whole new level when I noticed something: I kept accidentally building really cool Tablet PC programs.

Now we all know better than that, right? No one accidentally builds anything in software, at least not anything useful or successful. As a UML instructor, I spend much of my time teaching how purposeful design leads to successful systems. And yet here I was, Mr. UML, accidentally building really cool Tablet PC programs. Unless we're in the Realm of Infinite Monkeys, code doesn't just happen. There has to be intent there. And I'm used to the process being: intent-thought-work-thought-work-thought-work-work-work... But these Tablet PC apps felt like accidents, like unexpected surprises out of nowhere, because the process was closer to intent-thought-work-done! Developing these apps felt like an accident or a discovery, like there were solutions lying around just waiting for me to find them and tie them together.

One example of an application that practically wrote itself

is Vox, a handwriting-to-speech Tablet PC tool. I was asked to give a ten-minute talk at the Ann Arbor .NET Developers Group, and I suggested text-to-speech as a brief topic. After my talk was accepted, I realized the topic was too brief: the core code to let a user type a message and have the computer speak it was less than five lines. So to beef it up for the talk, I switched to writing a Tablet PC application:



the user can write a message into the Microsoft.Ink.InkEdit control; and then the Tablet PC converts the handwriting to text, and speaks that text. But while this made for a more interesting talk, I still had less than five lines of code! The Tablet PC API did all the hard work, so I could concentrate on adding user interface features to let the user edit and reuse text. I spent less time writing the core code than the talk itself.

Another very simple yet very powerful Tablet PC application is a paint program, complete with saving and loading Ink images in multiple colors and pen shapes, exporting to multiple image formats, erasing, selecting, moving, resizing, and printing. Now I won't claim that this is a radically new and improved paint program; but what's impressive about it is that I can create it from scratch, writing and explaining the code as I go, all during the span of a 90 minute presentation!

Mark my words: the Tablet PC is more than just a laptop with a pen. It's something new, some combination of the right technologies at the right time to change computing in ways we haven't really foreseen yet. And I think a big key to the change will be the Smart Cocktail Napkin metaphor, which, when combined with the power and ease of use of .NET programming, just keeps letting me

do really complex, really powerful things in a way that is simultaneously:

- Easy for my user to use, because Microsoft wrote all the really hard pieces that go into a natural, intuitive user interface based around Ink;
- Easy for me to write, because the Tablet PC API and

.NET provide a tremendously rich base functionality;

- And easy for me to reuse, so my repertoire of powerful programs keeps growing. .NET makes code reuse easier than I've ever experienced before. This was always the promise of OO (and then later of components); but for the first time, I feel like the promise is really delivered. .NET makes it just as easy to design for reuse as to design in the first place, and almost as easy to move something into the library after it's designed for one app.

And I haven't yet scratched the surface of the other two technologies that add power to the Tablet PC: ubiquitous WiFi and ubiquitous speech recognition. By spec, the Tablet PC has to have both of these technologies. And once you know you can take those for granted, you can envision new applications that weren't worth discussing before.

So I'd like to close as I opened, with a little 60s protest music. This time, The Beatles:

*You say you want a revolution...*

It's here.

**Martin L. Shoemaker** is a .NET developer specializing in Tablet PC programming, as well as analysis and design with UML. He is also the author of Tablet UML, a Unified Modeling Language tool for the Tablet PC ([www.TabletUML.com](http://www.TabletUML.com)).

#### Core Vox Code

```
using SpeechLib;
...
/// <summary>
/// Voice for speaking.
/// </summary>
private static ISpVoice mVoice = new SpVoiceClass();
...
// Speak a string. The flag value 1 indicates asynchronous speech.
uint uiStream;
mVoice.Speak(message, 1, out uiStream);
```

*continued from page 1*

at the core, bits and bytes.

When a customer recognizes this, a software consulting organization can lose a project proposal if they start talking bits and bytes too early. They are talking technology before showing that they understand the customer's problems. No doubt the technology is part of the solution but it doesn't itself open the door to a solution.

A good way to illustrate our approach is in a brief statement that we use about our business. When someone wants a 30-second summary about what we do, here is one of things we say:

Stout Systems helps company officers, managers, and business owners who are aggravated because they have an IT project that is in trouble, or are under pressure because their IT or project goals are not being met, or are disappointed by the quality of high tech personnel referred to them from recruiting companies.

We don't mention specific technologies or processes, because those things are determined by the problems we find.

It's worth mentioning that sometimes customers don't know what it's costing them not to solve the real problem—so the budget they have allocated is insufficient. We always ask a lot of questions to get to the bottom of the actual problems that have to be fixed, and then we get at least an estimate of what it's costing them in real dollars not to fix it. That helps the customer evaluate whether or not there will be sufficient return on investment if money is invested in the solution.

**John W. Stout** is the founder and president of Stout Systems Development. He has twenty-five year's experience in the software industry. He is also sought after as a technology speaker, presenting sessions at developer conferences and user groups.



ADDRESS SERVICE REQUESTED

Presorted Standard  
US Postage Paid  
Permit #187  
Ann Arbor MI

**In This Issue:**

- Talking Technology too Early
- The Tablet PC Revolution