1. FEATURE: Why Projects Fail - and What You Can Do About It

"I just don't understand it," the project manager said dejectedly. We had just had a meeting with a client for whom we were building a large software project. "We put our best people on the job; we authorized overtime; we even promised bonuses if the project went well. And now this."

The "this" to which my then-boss referred was the client's rejection of our efforts. While they assumed that we hadn't given the project our best efforts, we knew that we had. Yet we had failed - and worse, we had no assurance that we would do any better on the next project.

If this scenario sounds familiar, you're not alone. Repeated studies have shown that the failure rate for custom software projects is above 70%. This is an astonishing number and is tolerated only because software is so vital to the running of modern organizations. Why, though, do these failures occur? If we can't blame it on not working hard enough, then what causes us to fail?

Over the many years since the failure I've related here, I've had a chance to study this question. I've come to the conclusion that what fails us is not a lack of efforts, but is our efforts themselves. What did Shakespeare say? "The fault, dear Brutus, is not in our stars, but in ourselves." Or perhaps, in our methodologies.

Who Needs a Methodology?

A methodology is a defined series of steps leading to the accomplishment of a goal. Most development methodologies produce an unwanted goal: failure. Yet it's important to realize that, though this result is not desired, it is the natural consequence of these methodology. To change the results, we must change our methodology.

What methodologies are being used today? Perhaps the most popular methodology is the no-methodology methodology (NMM). Adherents of NMM are sometimes called "cowboy coders" and their motto seems adapted from the movie, Blazing Saddles: "Methodology? We don't need no stinkin' methodology."

As popular as NMM may be, there are problems. It doesn't scale well. Division of labor doesn't exist. I might like to have different people with different skills working on different parts of a project, but this requires that we understand the whole well enough to break it into its parts. NMMers only know the whole when it's finished.

The "Best Developers" Methodology

Another popular methodology is the "Best Developers" one. This is especially popular with pointy-haired bosses. Their pitch is simple enough: "Hire the best coders and get out of their way." Their motto might be "Our Developers Can Beat Up Your Developers."

What if other disciplines were to adopt this "methodology"? Would we hear the head of the Federal Aviation Agency discussing their new plan to "hire the best pilots - and get out of their way?" The FAA chief wouldn't be the only one "getting out of their way." The "Best Developers" methodology is based on the faulty premise that project failure is due to inadequate programmers. My experience has been that most coders are both competent and committed to success. Failures aren't due to lack of talent or

commitment. Something else is at work.

The Proprietary Methodology

Another popular methodology is the proprietary methodology. This methodology works tremendously well, but it's secret! We could tell you what it is - but then we'd have to kill you. Proprietary methodologies, by their nature, are closed and secretive. They lack the fundamental requirement of all serious approaches: peer review.

This may be the worst possible methodology. Clients are locked into a single company with no assurance that the company will be responsive to their future needs - or even (as we saw in the meltdown of dot.coms) that it will be around at all. For individual programmers, working in a proprietary development shop is a career dead-end. The investment the programmer makes in learning the secret methodology is useless outside that small world.

Searching for a Methodology

I should confess that my interest in these topics is not merely academic. Having been involved in my fair share of failures - including some spectacular ones - I set out to discover what was causing best efforts to turn into worst results. What I found was that lacking a well-defined methodology, developers lacked a road map to ensure that they could get from the first, heady days of a new project to a successful end. That lack of direction, I found, meant that often a project could be doomed before the first line of code was written.

During my investigation, I became heavily involved in an open standard quest for a workable methodology. That methodology is Fusebox, and I will discuss it in this article. I find Fusebox works very well for the many developers I work with. However, I don't tout it as the only or the ultimate methodology. There are many methodologies available. Which one is best is that one that you will use - whose philosophy aligns with your own. The methodology needed by a coder working completely alone can be very different from one that involves a team of developers - particularly if some of those developers are remote workers.

In order to get the full utility from a standard however, it must be widely adopted. Fusebox is a development methodology that arose from the ColdFusion community, though it is not confined to that language. Fusebox begins with the assumption that there are no bad guys in software development, although there is an awful lot of failure, and to remove that failure, we must understand its causes.

To observe the amount of rhetoric devoted to arguing over the "right" platform or language or IDE or compiler, you might think that the technical challenges of software development are the real cause of our 70% failure rate. That's odd, because software projects almost never fail for technical reasons.

Getting to Failure's Root Cause

Consider a less-than-successful project you've been involved in. Did it run into trouble because the programmers couldn't handle the rigors of recursion or matrix math or multi-dimensional arrays? No?

To find out why projects fail, we need only ask our clients. They'll gladly tell us: "You don't give us what we want!" Against

such an indictment, arguments over compilers and architectures are both irrelevant and foolish.

Fusebox begins by noticing that we typically involve the client at two points in the process. In the beginning of a job, we sit through often painful meetings with clients or we may ask clients to answer questionnaires. We mean well during this requirements gathering phase, but the problem is that the beginning of a job is too soon for a client to give us a lot of useful feedback.

The second time we involve clients is at the end of a project, when we go through acceptance testing. Now, we get real client feedback, but now, it's too late: The time and money for the project are all used up.

Tools for Requirements Gathering

A great deal of the problem can be laid at the feet of the language we use. We speak of "requirements gathering" as if "requirements" were scattered about, waiting for us to pick them up and drop them into a basket. Fusebox says that clients can only tell us what they want when they see it! And with that realization, it becomes clear that our job as developers is to give clients something to see, so that they can tell us what they want - not in legalistic specification documents, but in simple, plain language.

We use two primary tools for this. Wireframes offer developers a way to rapidly knock together a clickable skeleton. There are no graphics and no attempt to make the application functional. Instead, each page the client will eventually see is represented by a simple page that tells (1) what the responsibilities of the page are and (2) what other pages are linked to this one.

Prototypes follow wireframes. Each prototype page has a small threaded messaging system that is automatically affixed to HTML pages. This system allows developers and clients to communicate with each other. The results of this interaction are saved into a central repository. Prototyping continues until both developer and client are satisfied that the prototype reflects exactly what the finished application will look like. Only then is the prototype "frozen" and coding can begin. In this way, we can find out what the client wants before we write the code.

Who Are We?

Because we write code, we tend to think of ourselves as coders. But our clients see us very differently. They want us to be their guides through the complicated and dangerous process of deploying a successful project. Our challenge is to see ourselves not as mere coders, but as full developers.

To this end, methodologies are important. Good ones provide a framework around which we can predictably produce successful software deployments that benefit ourselves, our companies, and our clients. If you would like to learn more about Fusebox, please visit http://www.fusebox.org.

# # # #

About the author: Hal Helms trains, writes, and consults with developers on the Fusebox methodology. He is a Team Macromedia member and a well-known ColdFusion speaker and author. He began programming with Smalltalk, studying at famed Xerox PARC. He can be reached at hal.helms@teamallaire.com. His web site is http://www.halhelms.com.