What Software Reality Is Really About

James Bach, Satisfice

About three years ago, in my first column for this department, I wrote “So long as our true practices are shrouded by a false view of our methods, we will be frustrated in our efforts to close the gap between our current experience and that grander success we keep reading about.”

I was talking about methodology gaps—the difference between what we do and what we claim to do. I set for myself the mission of writing and editing articles to explore and analyze the way things truly happen: the “reality” of software projects. Now I’ve come to my final column as editor of this department. Where am I in my mission? Not very far along, of course.

One challenge has been the vastness of the subject. My perspective is quality assurance, configuration management, and project management. I’m not competent to say much beyond those specialties. Another challenge is that I’m not on software projects any more, at least not for more than a few days at a time.

Still, looking back over the columns I and my cohorts published, we’ve covered some interesting ground. Enough ground to make a stab at a grand summation. So, here is my take on what software reality is really about.

SOFTWARE REALITY

For starters, it’s more about people working together than it is about defined processes.

A common argument in favor of defining software development processes is that the alternative is cowboy chaos. Large projects of all stripes require people to behave in coordinated ways. Maybe so. But there’s more to defining processes and coordinating people than assigning someone to dream up a checklist and get it blessed in a staff meeting. Whatever your defined processes, if you don’t know how to work together, I can all but guarantee that your processes are not being followed. Furthermore, if your team can work together, you may find that the coordination you need can be achieved without thick process manuals.

Everything really interesting that happens in software projects eventually comes down to people. It’s a fact of life. Deal with it. I’ve never been on a project where methods, metrics, processes, or equipment actually dictated the course of the effort. It sometimes appears that way, I know, until you look behind the processes and discover that someone breathing human like you and me, in some office or cubicle, is behind it all. Someday I hope to visit a CMM Level 5 shop and meet the talented personalities who make it go.

If the important insights about software projects begin with people, they continue with the thinking we people do. Software reality is about science, understanding, inquiry, skill, learning, and a quality I call “enoughness.”

Science

It’s more about science than it is about computer science.

Computer science is helpful in software projects. But what about science itself? Richard Feynman once defined science as the belief in the ignorance of experts. We could use a dose of that kind of science as organizations like the IEEE move closer to producing bodies of knowledge and supporting licensing programs for software engineers. This work seems to be supported not by qualitative research into the actual practices of successful software organizations, but rather by the passions of people who write certain textbooks or run huge stuffy companies. Hey guys, open it up.

Understanding

It’s more about understanding than it is about documentation.

It’s easy to say “We should document that” or ask “Where’s the documentation?” It’s much harder to create worthwhile documents. In some organizations, such as medical device manufacturers, project documentation is a necessary part of the product they sell. That’s fine. In other organizations, documentation is a tool to help people understand how things work. In the latter case, by focus-
you're any good at it? How would you

correct? I objectify it is the notion of skill.

It's relatively easy to talk about methods,
and you'll get an eyeful about methods.

It's more about inquiry than it is about
metrics. At one time I worked a lot with
metrics. I like them. But metrics alone aren't
enough. In order to use metrics wisely, you
either need a complete understanding of
exactly what controls your project and
how those controls work (nobody
has that), or you need the additional ingredi-
ents of humility and inquiry.

If you have an inquiring attitude, then
metrics join all your other observations to
help make sense of your situation. Seeing
a pattern in my bug-find-rate data, for
example, is the starting point for asking
questions such as “Is that a pattern I
should be concerned about?” and “What
could have caused that pattern?” When-
ever someone pushes metrics collection as
a strategy, and doesn't also suggest a strat-
egy of gentle inquiry, watch out.

It's more about skill than it is about
methods. Crack any software engineering book
and you’ll get an eyeful about methods. It's
relatively easy to talk about methods, especially
if we can label them. More difficult to objectify is the notion of skill.

Yet skill is the core issue. Any nontrivial method, performed without skill, may
cause more harm than good. So, your
team does object-oriented analysis.

It's more about being good than it is about being good.

When I hear someone tell me about
some great practice, some wonderful way
to do things, one of the first things I won-
der is what he did before he discovered
that practice and how he learned to per-
form the practice well. Everyone who’s
doing good work often began by doing poor
work. Everyone planning to do better
work needs to find some path to get there. I
find that the process of studying, experi-
menting, and negotiating with other mem-
bers of the team is more important than
having some prefabricated plan that tells
you what practices you should follow.

It's more about good enough than it is
about right and wrong.

Whenever you catch yourself thinking
"X is a best practice," consider this alter-
native: "If you don't do something like
X, then you run the risk of problems like
Y and Z." Any statement about the

It's more about enoughness than it is
about right and wrong.

Whenever you catch yourself thinking
"X is a best practice," consider this alter-
native: "If you don't do something like
X, then you run the risk of problems like
Y and Z." Any statement about the
goodness of a practice can be translated
into risk management terms. When you
do that, the binary idea of right and

enoughness

sence. With this tool I can cut through the
confusion and be confident in what I know. That tool is my peer
network.

As an example, I belong to a commu-

nity called Consultants Camp. We meet
once a week a year to discuss specific software
testing practices. Although these communities operate very differently, what
they have in common is that they change
slowly, allowing members of the com-

munity to get to know each other very
well, and they provide an opportunity to
exchange experience and collaborate on
independent projects. I also go to a lot of
conferences, and I’ve found fertile
ground there to make new connections
with people who are happy to ferret
out the errors in my work.

Once I learned how to ask colleagues
to review my work, and how to learn
from their opinions about it (that’s a
whole other column), I gained access to
a fantastic databank of wisdom. Not
only does this help my work, but it also
has the effect of building a genuine consen-
sus about how to think about the way
software projects do work and should
work. That consensus ultimately crosses
company boundaries and ripples out-
ward through the mechanism of these
personal relationships.

I suspect this is how our industry and
profession will evolve over the decades
to come. Certainly, we will be affected
by technological advances and pressure
from legal and consumer interests, but
our basic ideas about what are better or
worse practices are strongly influenced
by people we perceive as knowing how
to make software.

So, who writes the books? Who sets
the standards? Who crafts the laws? Who
will shape the paradigms of software
engineering in the future?

If your answer is “we will” instead of
“they will” (however it is you define we
or they), then I would urge you to look
up from your project, your technology,
and your company, and join the great
conversation of software engineering.