James Bach, Software Testing Laboratories

ENOUGH ABOUT PROCESS: WHAT WE NEED ARE HEROES

How to get people and technology to work together.

All software managers are faced with three P's when software is to be built: people, problem, and process. Should each of the three P's be given equal weight? Should one be elevated to a more important role than the others? Are managers focusing too much attention on the wrong P?

James Bach offers answers to these questions and argues in favor of software heroism. Bach has been a programmer, tester, test manager, or process consultant for 12 years. He has run projects at Apple Computer and Borland International, two companies that exemplify both healthy and unhealthy heroism. He is currently the chief scientist for Software Testing Laboratories, of Seattle. His book, Process Evolution in a Mad World, *will be published by McGraw-Hill in 1996.*

According to Bach, heroic software people find a way to solve problems, and successful software projects are a sequence of problems encountered and effectively solved.

— Roger Pressman

I'LL GET RIGHT TO THE POINT. At conferences and in journals, the extraordinary attention we give to software-development processes is misplaced. Far too much is written about processes and methods for developing software; far too little about the care and feeding of the minds that actually write that software. Process is useful, but it is not central to successful software projects. The central issue is the human processor — the hero who steps up and solves the problems that lie between a need expressed and a need fulfilled.

A HERO IS SOMEONE WHO TAKES INITIATIVE TO SOLVE AMBIGUOUS PROBLEMS.

Heroism is a concept that evokes strong images, so let me clarify. Heroism means going beyond the borders of the known world and returning with new knowledge or wealth. Because any venture beyond a stock solution involves both commitment and risk, the sustainable, healthy sort of heroism requires judgment to know how much commitment and risk is right for the situation. The movement toward process in our industry is an understandable reaction against *pathological* heroism: heroism for its own sake, in which overcommitment and uncontrolled risk-taking is the norm. Unfortunately, process advocates tend to treat all heroism as pathological.

To bring the classical definition of hero into the business and engineering milieu, I define a hero as someone who takes initiative to solve ambiguous problems. Heroism cannot be automated; it cannot be predefined as a set of specific actions or outcomes. Hence, there is no unambiguous way to teach or evaluate it. Despite its mysterious nature, we must face the need for heroism because in the world of software development all problems are ambiguous. For every formalism or pattern there is the ambiguous problem of knowing when and how to apply it. For every solved problem there is the metaproblem of adapting that solution to other, similar problems.

Heroism would only be a minor management concern — of no more importance than punctuality or style of dress — if not for the startling complexity and boundless ambition of our projects. The well-known linear and hierarchical project-management methods can't cope with the load, and we aren't satisfied to perform only small and uncomplicated projects. But there are ways to enhance the capability of our nonlinear assets — our people — and there are disciplines such as organizational learning, systems thinking, complexity theory, chaos theory, cybernetics, and psychology that we can employ to that end.

As managers and methodologists, we complain about hackers and cowboys, and they can be a problem, but rather than dehumanize the development process, we can address pathological heroism as we create conditions in which useful heroes are born and healthy heroes thrive.

HEROES ARE PEOPLE FIRST. The central issue is people, but it isn't the only one. Process surrounds people and emanates from them. By treating the dialogue of process and people as an adaptive system, I find that some chaotic projects become tractable, and strange behavior — such as people deliberately creating poor-quality software — becomes intelligible and even reasonable. My point is this:

- Software is a product of people, not

Editor:
Roger Pressman
R. Pressman & Associates
620 E. Slope Dr.
Orange, CT 06477
rsp0547@aol.com

some conceptualized process.

- People can be developed into effective leaders or, as I say, heroes.
- There are emerging disciplines that provide a context for conceiving of projects as adaptive systems, rather than collections of tasks and products.
- When we view projects in this way, we are better able to understand and influence the behavior of the people who make them run.

Let me step back from this central argument and provide a little context.

As far as we've progressed in the various technologies of information systems, our industry still has great difficulty consistently developing reliable software. We've tackled the problem from every direction, it seems, but projects and products continue to fail with alarming regularity. We shake our heads and mutter something about a "software crisis."

One approach touted as a general solution demands that we define and control the software-development process. The theory goes that, because all products are the result of some process, by controlling the quality of that process, we can more effectively build quality into the product. The Software Engineering Institute's Capability Maturity Model is probably the best known software-process architecture that endorses this strategy. The process-control approach is generally addressed to entire organizations, which means, first and foremost, top management.

Another, very different solution suggests that we focus on the people who actually create the software and manage the projects. The theory here is that each software product is, as a system, the unique result of a collaboration of human minds; hence it pays to work on the skills of collaboration, thinking, and software-development methods, and to provide an environment where those skills can be effectively applied. This is the so-called "peopleware" approach, a term popularized by the book *Peopleware: Productive Projects and Teams*, by Tom DeMarco and Timothy Lister (Dorset House, 1987). One hallmark of this approach is the application of psychological theory to software development, such as Gerald and Daniela Weinberg's use of Virginia Satir's interaction model in their books and seminars (*Quality Software Management*, vols. 1-3, Dorset House, 1991-94). Another aspect of this approach is its obvious reliance on heroes. The peopleware idea is generally addressed to lower level managers.

A third approach might be called the "cowboy" or "big magic" model. In this view, gifted people create software through apparently magical means, with no particular guidance or support. This approach also centers on heroes, but pathologically so. It doesn't do much to grow or nurture them. Rather, it tends to wear them out. This model is addressed more to the individual contributors themselves, and even encourages management to "lead by example" by performing as individual engineers.

THE TERM "PROCESS" IS MISLEADING: BEHAVIOR IS NOT A SEPARATE ENTITY FROM A PERSON.

CLASH OF PARADIGMS. To put these conflicting paradigms in perspective, we must look closer at the suspicious word "process." The rollicking forums on Compuserve and Usenet reveal patterns of misunderstanding that seem to be caused by the many faces of process. Watts Humphrey himself distinguishes between task- and entity-oriented processes; between processes, process models, and process architectures; and between universal, worldly, and atomic processes.

The term "process" has become overloaded. It allows us to talk about our behavior as if it were some physical entity separate from ourselves. This can be misleading. Consider the apparently simple question, "Do you have a process for X?" This question can be interpreted as

- "Does X happen?"
- "How do you know that X happens?"
- "Do you have an idea or program that guides X as you do it?"
- "Do you make a plan before you do X?"
- "Do you have an ideal way of doing X?"
- "Do you have a basis for understanding how you do X?"

No wonder it's so difficult to talk about process. In an ideal situation, what is idealized, planned, and implemented are one and the same thing. But in real life, the plan falls short of the ideal and the implementation falls short of the plan. Furthermore, the planned and ideal processes may not even be present.

I'd like to make two assertions here. First, bringing processes from description to prescription to emergence is a difficult and fundamentally ambiguous problem. Second, a project may be without a planned or ideal process and still be "using processes" in the sense that it is guided by internalized process prescriptions that are in turn guided by higher level process models embedded within experience, education, and insight.

If these are valid assertions, then processes and heroes may be discussed together, rather than in opposition. Moreover, heroism is the better part of process. *Ideals* are tools to make *plans* and both are tools that must be *implemented* by *educated* people to *perceive* the desired *actual* result.

How do process manifestations relate to the three approaches used to manage software projects? Consider this excerpt from CMM 1.1 regarding process control:

> *Even in undisciplined organizations, however, some individual software projects produce excellent results. When such projects succeed, it is generally through the heroic efforts of a dedicated team, rather than through the proven methods of an organization with a mature software process. In the absence of an organization-wide software process, repeating results depends entirely on having the same individuals available for the next project. Success that depends solely on the availability of specific individuals provides no basis for long-term productivity and quality improvement throughout an organization.*

In this paragraph, process means ideal descriptions that in a particular project will result in tailored plans to suit the needs of that project. The CMM asserts that ideals and plans are the *only* basis for consistent success. The peopleware model counters that defined process is only one element of support, and not the most important one. The cowboy model considers such processes to be an outright impediment.

I argue that the only basis for success of any kind is the "heroic efforts of a dedicated team." Plans and ideals spring from the heroic synthesis of research, models, experience, and insight. Plans and ideals come alive in the heroic act of implementing them at the right time and in the right proportion. Furthermore, such heroism is not the province of specific gifted individuals who must never leave the organization, but of teams of people with varying gifts and ambitions who have the support they need to do well.

How can these apparently disparate models be integrated? The answer lies in looking at projects as problem-solving systems rather than as collections of tasks. The project-as-system includes the cowboys and the processes, but places people at the center. Instead of focusing on tasks and products, it focuses on roles and solutions.

People accept roles more readily than tasks, and they are motivated to create products when they see that the products solve problems. Tasks and products can readily be mapped onto this model, but they are not the organizing principles. In the peopleware model, process improvement means a dialogue between process concepts and the heroes who create and interpret them. This leads to improvement both in the heroes themselves and in the expression of their processes.

HERO HELPER'S TOOLKIT. To help heroes thrive, and to thrive on heroes, our management inventory should include the following skills, tasks, and habits:

- Recognize, observe and analyze nonlinear systems (see *Complexity*, M. Mitchell Waldrop, Simon & Schuster, 1992); analyze and discuss qualitative risk; accept risk and learn from failure.
- Diagnose and rectify incongruence: inconsistency between thoughts, words, and deeds.
- Identify and track problems more carefully than tasks.
- Treat tasks and products as solutions, rather than as ends in themselves.
- Continuously analyze, assign, and clarify project roles.
- Collaborate with heroes, rather than dominating or ignoring them.
- Understand the limitations of static concepts, models, and plans.
- Before asking, "what is the process?" ask "who is the process?"

The world of business can show us the way: See *The Fifth Discipline* (Peter Senge, Doubleday, 1990) and *Liberation Management* (Tom Peters, Fawcett Columbine, 1992). ◆