Robert L. Glass

Evolving a New Theory of Project Success

Regardless of how "troubled" a project may be, what is learned after crossing the finish line can be a practitioner's overwhelming success.

"Ask the Man Who Owns One"
—Advertising slogan for the Packard automobile

A lot has been written over the years about software project failure. You'd think by now there was nothing new to say on the topic. But I have recently read a study that sheds important new light on failure. I'd like to share it with you.

You already know what's old about this topic. Popular-press articles about spectacular failed projects. Hand-wringing articles about how software projects are always over budget, behind schedule, and don't work. Books about projects that fail, including Ed Yourdon's *Death March*, and my own *Software Runaways*. The articles and books may or may not be sympathetic to the software project personnel (most authors see software practitioners as "bumbling"). But the prevalent theme in most literature is usually some sort of variation on "ain't it awful." Software project failure is frequent and lamentable, these writings say.

So it may surprise you, as it did me, that one author found something new to say about software project failure. This is one of those "why didn't I think of that?" studies.

The study is by a newcomer to software literature. Kurt R. Linberg is a software practitioner who is also working on a Ph.D. in computing. His study has not yet been published, but it will be in Elsevier's *Journal of Systems and Software*, early in 2000.

What did Linberg say that was new? He decided to confront the software developers on a failed project and ask for their perceptions on what had happened. And what he found—to his amazement, to my fascination, and to the surprise of many who will read his work—is that the project participants declared this failed project to be one of the most successful they had ever worked on.

I can imagine the overwhelming mix of reactions to this description of Linberg's work. You may be thinking, "No wonder we have failed projects— software practitioners can't tell success from failure." You may be thinking, "I knew it was high time we did a better job of defining success and failure." Or you may be thinking, "What a classic case of employee/employer relationship breakdown; what the employers see as failure, the employees see as success." But none of these reactions quite capture Linberg discovery. The uniqueness of both the questions Linberg asked, and the answers he acquired are what make his study so fascinating.

Let's set the stage here for an elaboration of Linberg's findings.

The project was the develop-

LAURA CORNELL

This was no poster child for project failure, but it was a project hardly anyone in upper management could brag about.

ment of software for a medical instrument, one used by medical professionals to perform procedures.

The practitioners were experienced and application-knowledgeable, averaging two years of post-graduate education, with 64% having more than five years of software development experience and a background, on average, of 14 prior projects.

What went wrong on the project? It was over budget by 419%. It was over schedule by 193% (27 months vs. 14 estimated). It was over size estimates by 130% for its software component and 800% for its firmware component. By all the usual measuring sticks, one could consider this to be a troubled project. It is important to note that there was no catastrophic failure here; the project was eventually completed, did what it was supposed to do, and possessed the required "no post-release software defects." In other words, this was no poster child for project failure, but it was a project hardly anyone in upper management could brag about.

It is important for me to state here I consider this project closer to the typical software project than the runaway successes or the abject failures that we are so used to reading about. Missing cost and schedule targets but building software that works may correspond to failure on two of the three criteria used for the "software crisis" definition, but these two are (in my opinion, perhaps controversial) the least important criteria. (We will return to my thoughts after we examine what the practitioners on this particular project told Linberg.)

What Linberg did next is what leads us to the surprise in his findings. He asked each of the eight project participants to describe the most successful software project they had ever worked on. Five of the eight said their most successful project was, in fact, this so-called troubled one on which they were currently working. Furthermore, the remaining three project participants named the present project as the second-most successful project on which they had ever worked. In other words, the project participants saw their current project, which we have already identified as "troubled" and two-thirds of the way to being a "software crisis" project, as a success. What on earth was going on here?

It's important to examine why the project participants saw the project as a success. First of all, they said the product worked the way it was supposed to work. Second, developing it had been a technical challenge. And third, their team was small and high-performing.

One team member said "it was the most successful because of what we accomplished." Another said, "I enjoyed the people I worked with," and that "it was the best-managed [project] I've ever worked on." Why was it the best managed project? Because the team was given the freedom to develop a "good design." Because there was no "scope creep" as the project evolved. Because "I never felt pressure from schedule."

Linberg also asked the team participants about the least successful project they had ever worked on. The participants recalled projects where they were urged by management to "achieve dates no matter what." "We all said it was not possible" to achieve schedule, said another, but management "kept on adding people."

Even on this "most successful" but troubled project, there were problems. The participants were particularly upset that management tried to thwart their need to obtain expert help on such matters as the capabilities of hardware and software platforms. They spoke of clandestine meetings between themselves and the experts, conducted not with the support of, but rather in spite of, their management. Management thought the experts' time was too valuable to be interrupted for those purposes. The experts, as it turned out, disagreed. They even worked weekends to provide help to the team outside the scrutiny of the managers in question.

It's in the next set of Linberg's findings, I think, that is the heart of the matter. He asked the participants for their perceptions about why their project was late. They were quite definite in their answers. The schedule expectations, they said, were unrealistic. There was a lack of resources (remember the problem with obtaining expert advice). There was a poor understanding of the project's scope at the outset, especially regarding the firmware. The project started late. In other words, the project was troubled, these participants believed, not because of things that happened along the way, but because of what happened at the outset—poor schedule estimates, poor understanding of the problem to be solved, poor understanding of the resources needed.

Now I'd like to return to the personal aside that I mentioned earlier. Software's problems, these practitioners say, are about expectations established at the outset of a project much more than they are about trouble that happens along the way. I think this is a profound finding, and one that I wholeheartedly support. The important finding is that the software field needs not better ways of building software—as computer scientists and software engineer academics so frequently advocate—but better ways of approaching the building of software. More accurate and honest estimates. More understanding of the complexity of scope. More expert help.

We have already seen a profound difference of opinion between management and team members about what constitutes success—and failure. Near the end of Linberg's article is a fascinating table outlining these practitioners' own definitions of success and failure. First of all, these practitioners note, a project can be a success whether it is completed or canceled. Success, in their eyes, has to do with their learning experience on the project. If they learned something that can be applied to a future project, they say, then a project is a success. It's nice, they emphasize, if the cost and schedule performance are comparable to normal industry achievement (note that they specifically avoid linking this to project estimates), but success has to do with what the participants learned along the way.

Now it would be easy to conclude this column by wringing one's hands about those "cowboy" programmers who refuse to conform to management expectations and continue to do and see things in their own ways. In my opinion, however, that would be the wrong conclusion to draw from Linberg's findings. These practitioners, I assert, tell us something we all need to hear. Linberg himself says it best: "A new theory of software project success may be necessary." This new theory may need to include realistic expectations, placing importance on a quality product, and organizational congruency.

I believe, as I announced at the inception of this column a couple of years ago, in software's practice and software's practitioners. When in doubt, it is best to go to practice to identify questions and to find answers about how practice might be improved. (Hence the Packard slogan at the beginning of this column.) Linberg does an exceptional job of doing just that. I hope we have more researchers who will follow in his footsteps. **C**

Robert L. Glass (rglass@indiana.edu) is the publisher/editor of *The Software Practitioner* newsletter and editor of Elsevier's *Journal of Systems and Software.*