

Published in the October 2004 issue of CrossTalk,
The Journal of Defense Software Engineering, US Department of Defense

Catastrophe Disentanglement™

Getting Software Projects Back on Track

E.M. Bennatan

If you are responsible for a late and over-budget software project you are by no means alone; software project overruns are all too common. But if serious problems have existed for quite a while and the situation is getting worse, not better, you may have a project catastrophe on your hands. At this point, there is no PMI, IEEE, SEI, or ISO rescue process to follow, because these organizations essentially offer preventive, rather than corrective solutions. This article describes a ten-step process to disentangle a software project catastrophe and get it back on track.

In Spencer Johnson's "Who Moved My Cheese" [6], the littlepeople keep coming back to where the cheese used to be even though it's not there anymore. It's a natural tendency to continue doing what we did before even when, to an outside observer, it no longer makes sense. This behavior is quite common when projects get into trouble. We keep plodding away at the project hoping that the problems will go away and the "cheese" will miraculously reappear. In all too many cases, it doesn't.

Just as the smart thing to do when a ball of twine seems hopelessly entangled is to stop whatever we are doing with it (otherwise the tangle gets worse), so it often is with a disastrous project; the longer we keep at it the worse it gets. At some point, we need to halt all activity and reassess what we are doing.

Disastrous software projects, or *catastrophes*, are projects that are completely out of control in one or more of the following aspects: schedule, budget, or quality. They are by no means rare; 44% of surveyed development organizations report that they have had software projects cancelled or abandoned due

to significant overruns, and 15% say that it has happened to more than 10% of their projects (see Figure 1).

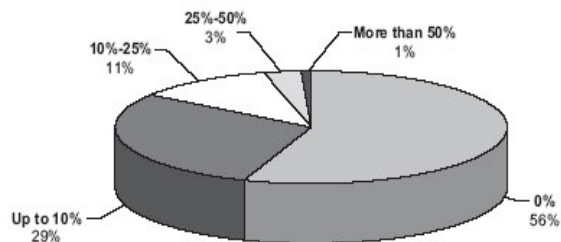


Figure 1 – Percentage of your organizations software projects that have been abandoned or cancelled due to significant cost or time overruns in the past three years (source [3])

But obviously, not every overrun or quality problem means a project is out of control; so at which point should we define a software project as a catastrophe? What are the criteria for taking the drastic step of halting all activities, and how do we go about reassessing the project? And, most importantly, how do we go about getting the project moving again? The answers to these questions are the essence of the concept of *catastrophe disentanglement*.

When is a Project a Catastrophe?

Organizations and projects vary to such an extent that there can be no universal criteria for branding a software project a catastrophe. The expectations from mission critical, life support, or banking software are significantly different than from most consumer or Internet-based software applications. But experience shows that in virtually all cases, projects are in deep trouble if serious problems have existed for quite a while and the situation is getting worse, not better. How is this reflected in terms of schedule, budget, and quality?

Schedule: Software projects rarely or never strictly follow their schedule; delays often grow and shrink like an accordion. It is also a sad reality that software project delays are an excessively common occurrence (see Figure 2¹). But we are not looking at just any delay; the issue here is to identify those projects where the delay is growing uncontrollably.

To determine if the delay is out of control, divide the total development schedule into 12 phases, and look at each of the last three. Has the delay steadily grown in each phase and is the total delay now greater than three phases (i.e. 25% of the total project schedule)?

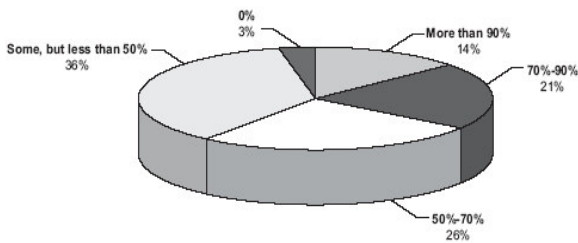


Figure 2 – Percentage of organization’s software projects completed within 10% of original estimated time in the past three years (source [3])

For example, for a one-year schedule, we would look at the last three months and ask:

1. Was the delay significant two months ago?

¹ To be statistically accurate, the results may have included some projects that were finished early, but we risked the speculation that such cases (if any) would only represent a small fraction of the results.

2. Was the delay even greater one month ago?
3. This month, has the delay grown again?
4. Has the delay growth been steady (that is, not two small delays and one major delay caused by an identifiable event)?
5. And lastly, is the total delay now greater than three months?

If the answer to all these questions is yes, it is probably a good idea to halt the project and reassess it.

Warning Signs to Watch For in a Project:

- It is late and getting later
- It is over budget and getting more so
- Performance is poor and getting poorer
- Criticism from customers/users is severe

Budget: A project is a budget catastrophe if its remaining projected cost far exceeds what the development organization is willing to pay for it. In software projects, major budget overruns are often the result of schedule overruns or of attempts to reduce schedule overruns (e.g. by adding staff). Points to consider are:

1. Does the project schedule appear to be a catastrophe? If so, project cost projections have little value at this time.
2. If the project schedule appears to be under control, then extrapolate budget overruns for the past three phases up to the end of the most current project schedule (assume that every future phase will continue to exceed the budget at a similar rate). Is this a cost your organization can bear?
3. Do you have current feedback from the project’s customers and users? Do you have updated market research data? Is the original cost/value analysis for this project still valid?

Quality: A software project is a quality catastrophe if (a) the list of serious quality problems has been substantial for three periods and is not decreasing, or if (b)

customers/users who have evaluated the software being developed are exceptionally critical of it.

The project problem list is a good indicator of how serious the problems are. The list is commonly divided into (a) critical, (b) serious, and (c) minor problems.

Points to consider are:

1. Is the critical problem list growing? Are problems being resolved? How fast are new problems being added?
2. The second level of serious quality problems can also indicate the gravity of the situation if the list is particularly long and not getting any shorter.
3. Another indicator to monitor is how well the quality problem lists are being maintained. Are problems being categorized correctly? Are problems being removed prematurely from the list? Are new problems being withheld from the list?

Severe quality problems (those that are either critical or most serious) are often difficult, if not impossible, to see in the early stages of a project. In fact, many severe quality problems emerge only towards the end of the project (and sometimes only after its release). Even the “last three phases” technique can be ineffective during the first half of the project because often problem lists have not yet been compiled or well maintained.

But project quality issues can be monitored from the outset if there is someone whose job it is to do so. This means assigning an independent software quality assurance (SQA) professional to every project team as soon as the project is launched. For small development teams, one SQA professional can be responsible for two or three projects, though large projects should have their own indigenous SQA team.

Customer and user feedback are the best source for evaluating the quality of a project. Unfortunately, it is sometimes difficult to get feedback before a project is very close to

release. For large projects, it is often worth investing in prototypes and prereleases, thus getting preliminary versions of the software into the hands of users for early evaluation and feedback. This investment is like an insurance policy; it reduces the risk of major product quality issues – but at a cost.

The Project is a Catastrophe – Now What?

The following ten steps describe the process for disentangling a failing software project and getting it back on track. Because these steps intrude on the responsibilities of the team members – most especially the project manager – the process should be confined to getting the project back on track and nothing more. Ultimately, the new project plan must gain the unreserved support of the development team members, and the details should be left up to them.

Ten Steps to Disentangle a Software Project Catastrophe

1. Stop
2. Assign an external unbiased evaluator
3. Evaluate true project status (what has been achieved and what has not)
4. Evaluate team capabilities
5. Define new minimal goals and requirements with senior (executive) management and customers
6. Determine if minimum goals can be achieved (if not abandon, find alternative to the project)
7. Rebuild the project team
8. Perform high-level risk analysis
9. Develop reasonable estimates
10. Install an early warning system

1. Stop: Once you have determined that a software project is unlikely to be completed with any reasonable degree of success, the next step is painful but clear: stop all activities immediately.

This is a difficult decision because it will always be open to harsh criticism from some circles. It is also a tough decision because, as we have seen, there is really no airtight algorithm for determining that a project is a catastrophe. Ultimately, the decision is a combination of data analysis and management experience.

Stopping a project should never leave a team idle. There is much to do preparing the project for assessment, including:

- Collecting and updating project documentation and data
- Preparing status reports for each team member and each team
- Bringing the project software to the nearest point (backward, but not forward) for demonstration. This means that except for minor exceptions, no new code should be written and no new features should be added or integrated (otherwise there is a risk that the demonstration will take much too long to prepare).
- Assisting the project evaluator

In addition, other activities should be prepared and held in reserve such as training and assistance to other projects.

2. Assign An Evaluator: Virtually all software projects in trouble have strong emotional and political hallmarks often producing passionate advocates and opponents. Therefore, the importance of using an *external* project evaluator cannot be overstated. This will increase the likelihood of getting an unbiased and unemotional evaluation.

Whom should you choose? Ideally, you should assign a reliable, pragmatic, experienced, and successful project manager who (i) understands the project technology, (ii) has good social skills, and (iii) can reprioritize other responsibilities to allow sufficient time for the evaluation.

For very large projects, use an evaluation team of two or more evaluators, but with a clearly designated chief evaluator.

3. Evaluate Project Status: The first challenge in evaluating a project is to determine its true status. Most failed software projects will have produced many status reports – some may even be quite positive – but they will not necessarily be objective or dispassionate.

In establishing an unbiased view of the project status:

- Reduce tension by involving the project team in your evaluation and by being completely open (no secrecy or mysterious behind-closed-doors discussions).
- Consider only observable facts (e.g. not “this feature used to work well but something has gone wrong”)
- Consider accomplishments, not effort.
- For almost completed tasks, apply the 90-50 rule (*It takes 50% of the time to do 90% of the work and another 50% to do the remaining 10%.*).
- Present your evaluation to the team before finalizing it and consider their responses (look for details and facts that you overlooked or misunderstood, while resisting undue pressure to amend your findings).

Choosing a Project Evaluator

- External (this might be the time to use a good consultant)
- Reliable, pragmatic and experienced
- Understands the project technology
- Has good social skills
- Can devote sufficient time

4. Evaluate the Team: Evaluating a team is a sensitive activity that should be handled both resolutely and tactfully. This step is purely part of the evaluation process and does not, at this point, result in any restructuring of the team.

The questions to be considered are:

- Does the project team have the necessary skill set and experience to successfully deliver the project?
- Do the team leaders have the leadership and technical skills and the personality necessary to lead their team?
- Does the project manager have the required leadership and technical skills and the personality necessary to lead the project team and does he or she command the respect of the team members?
- Are there any internal team conflicts or tensions that could disrupt the project?
- What is the level of team spirit and moral? If low, then why? (Are there reasons beyond the failing of the project?)

5. Define Minimum Goals: The emphasis here is on the word minimum; the project should be reduced to the smallest size that achieves only the most essential goals. This resetting of goals and objectives can only be performed with the active involvement of senior (executive) management and the customer².

Divide all project requirements into three sets:

Set 1: Essential requirements without which the project will have no value.

Set 2: Important requirements that greatly improve the project but are not essential.

Set 3: Nice-to-have requirements that add to the project, but are not especially important.

Now, start by retaining the requirements from set 1, and initially eliminating sets 2, and 3. This will often create tremendous opposition, but remember – we are dealing with a project that was totally out of control and may otherwise be cancelled. Occasionally, some elements from set 2 can be added, but this should be rare. All remaining requirements

² The term ‘customer’ here refers to the entity that requested the project or that will use its product, or more generally, for whom the project is being developed.

(from sets 2 and 3) should be targeted for subsequent releases of the software.

A word of caution: Be prepared to forestall the ploy by some stakeholders to second-guess the whole evaluation process by their insistence on listing all (or most) requirements in set 1.

6. Can Minimum Goals Be Achieved? The main challenge here is to determine whether the requirements in set 1 can reasonably be achieved. The questions to be addressed are:

- Is set 1 a genuine and significant reduction of the project scope?
- Is there a single requirement in set 1 that adds an order of magnitude to the complexity of the project? If so, are members of management aware of this and will they reconsider its inclusion?³
- Are the new project goals now achievable? Is there now a reasonable chance that the team will be able to deliver the requirements in an acceptable time frame, within a reasonable budget, and with an acceptable quality level?
- How genuinely confident are the team members (and especially the project manager) in their ability to achieve the new set of goals?

If the minimum goals appear unachievable (and they are truly minimal), a recommendation to cancel the project may be the only remaining realistic course of action.

7. Rebuild the Team: Based on the evaluation of the team(see step 4) it may need to be restructured and even partly restaffed to handle the new set of goals.

³ Fred Brooks (of *The Mythical Man Month* fame) tells the story of a senior naval officer’s last minute requirement after many months of negotiating features, schedule, and cost for a new navy helicopter. “It must be able to fly across the Atlantic.” he stated. Only after laboriously explaining to him the enormous complexity that it added to the project was the officer willing to drop the requirement [4].

A halted software project can often mean a demotivated and demoralized team. But in all probability, if the project was in deep trouble before it was halted, then the low morale did not start with the decision to halt the project. However, the issue of team morale should be a major consideration in rebuilding the project team (this will be further discussed later).

In rebuilding the team consider the following points:

- Team Structure – is the project team structured optimally for the success of the project?
- Team Functions – are the necessary team functions staffed?
- Team Members – are there any that should be replaced?
- Team Leaders – are there any that should be replaced?
- Project Manager – is the project manager the right person to lead this project?

8. Risk Analysis: In all phases of a software development project, risk analysis is virtually an indispensable tool – this is particularly true of a failing project trying to get back on track. The process identifies risk events, mitigation steps, and contingency plans, and assigns tracking responsibilities⁴.

High-level risk analysis (i.e. anticipating the most serious potential problems) should be performed as part of the project evaluation process. The analysis will not only help evaluate the chances of success in restarting the project, it will also help restore a level of confidence within the project team.

9. New Estimates and Schedule: Based on the minimal goals and the rebuilt team, new reasonable high-level estimates and a new schedule need to be prepared and the cost-effectiveness of the renewed project plan should be established.

⁴ For an overview of basic software project risk analysis, see [2].

If the schedule is firm, ensure that budget, staffing level, and feature set are not also all fixed (or another catastrophe will ensue).

In many cases, it may be prudent to focus primarily on the schedule and feature set (the other parameters, such as budget and staffing levels, can initially be sidelined). This means that if the minimal feature set is firm, then calculate the project delivery date and vice versa. Remember that even a generous budget and an unrestricted staffing level may not be enough to resolve the problem of a fixed feature set with an uncompromising delivery date.

A note on cost effectiveness: In analyzing the cost of completing a software project, only future costs (not costs already expended) should be considered. The cost of project completion should then be compared to the value of the completed project.

10. Establish Clear Project Review-Milestones: Put in place an early warning system to ensure that the project does not slip back into catastrophe mode. Such a system should include:

- The introduction of an efficient and reliable project data collection and analysis system
- Clear project evaluation criteria for management
- A schedule of frequent project reviews with well-defined measurable milestones

After successful completion of the above ten project evaluation steps, and after determining that the renewed project plan is achievable and cost effective, the project can be restarted.

Case Study

A failing project is often like a hand in a cookie jar; to get some cookies out you first have to let some go. Such was the case at Motorola with the software for a wireless telephony⁵ control and maintenance center

⁵ ‘Telephony’ here refers to the provision of telephone-related services.

(CMC) that we delivered several years ago as part of a two hundred thousand subscriber project, to one of the emerging Eastern European countries (see [1]). The specially tailored CMC was a last minute add-on to the wireless telephony contract and was consequently not well defined.

The CMC was developed with a subcontractor team, based on an existing control system, and the first phase of the project was devoted to producing a voluminous set of requirements, none of which could be omitted (according to the customer). The schedule was dictated – 16 months, which was set as close as possible to the date the subscriber telephony system was to become operational. Needless to say, every month was critical.

Five months into the project, key dates were already being missed. Seven months into the project, doubts began arising among senior management about whether the project would be ready on time. Nine months into the project, senior management was trying to calculate how much the late delivery penalties would cost, and a frantic marketing team was looking for alternatives. At all junctures, the development team was adamant that they would deliver the project on time.

At the end of nine months, amid significant resistance from the development and marketing teams we brought the project almost to a complete halt (some tasks did continue). Two activities were then launched: (i) a total external review of the project, and in parallel, (ii) negotiations were reopened with the customer on the CMC requirements.

- The project status was evaluated and it was confirmed that the then-current rate of progress would lead to a major project overrun. The team was moving forward at a steady pace but there was no way that they could meet the delivery date, or any date close.
- Because the CMC was critical for the operation of the whole system, the customer was cooperative in reevaluating the

project's software features. Thus, a new set of minimal requirements was prepared.

- The project was rescheduled with two release dates: the first with the minimal feature set and the second with the remaining features.
- On the development team side, instead of using a single team for development, installation, and support, a cooperative effort was launched together with a local support team.
- Frequent progress reviews of the project were initiated by management with key members of the development team together with the customer.

As a result, a working CMC system was delivered on time and the full telephony system became operational as planned. The additional CMC features were provided as part of a later second release.

The Customer Perspective

Some software organizations' attitude toward customers is reminiscent of the librarian who disliked readers removing books from the library shelves because they disrupted the tidy placement of the books on the shelves. The librarian had confused means (the library) with goals (reading books). In software development, we also sometimes tend to confuse means (the project) with goals (customer satisfaction⁶).

There is justification for a project only as long as there are willing customers for its product. Hence, it is wise for both management and the project team to keep an ever-watchful eye on the customers: their needs, their expectations, and their opinion of the software being developed. After all, the continued development of a product that no longer has a willing customer (or user) is the ultimate project catastrophe.

⁶ Yes, profitability is usually a good goal, too.

Post Project Reviews

Getting a failed software project back on track is an admirable accomplishment, but an even greater one is not having it go off track in the first place. Therefore, part of the catastrophe disentanglement procedure is preventing future recurrences of similar catastrophes. This is achieved through a special review process held after the project has ended (successfully or otherwise).

The post project review is a process intended to facilitate an understanding of why a project evolved the way it did; what was done right, what was done wrong, and what can be done better next time⁷. The review is a structured process that is not intended to find the guilty or to lay any blame, and is best done with a trained facilitator.

The output of the review includes a list of operational, procedural, and organizational changes and actions to ensure that mistakes are not repeated and successes are. In fact, the US Army recommends that 50% of the review be devoted to discussions on how to do better in the future; the remaining time is devoted to what happened (25%), and why (25%) [7].

The Human Factor

The process of disentangling catastrophes is traumatic; not just for the project team, but for the organization itself. Clearly, halting a project does not add to the motivation of a project team. Similarly, declaring a project to be a catastrophe does not add to the prestige of a development organization – though the courage to make such a decision often deserves praise.

While a highly motivated team is certainly one of the primary factors for project success,

⁷ A useful overview of a generic after action review process, which can be easily adapted for software projects, is given by former US Army Chief of Staff, Gordon Sullivan, and Michael Harper in their text *Hope is Not a Method* [8].

the fear of demotivating a team, or tarnishing an organization's image, should never be a reason to allow a team to continue in the wrong direction. Catastrophe disentanglement should be viewed like corrective surgery; just as the body undergoes trauma in order to heal, so does the development organization.

One of the problems with the rather drastic measures of catastrophe disentanglement is that the knowledge that an organization will take such measures can inhibit the flow of accurate information (particularly bad news) to senior management. But successful corrective action, just like successful surgery, depends on the flow of truthful and accurate information, even (in fact especially) when the news is bad.

The Post Project Review
<ul style="list-style-type: none">• What happened? (25%)• Why did it happen? (25%)• How to do better in the future. (50%)• Who is to blame? (0%)

The ability to bring bad tidings and make unpopular decisions is a desirable, if not entirely common, part of an organization's culture. Former Intel CEO, Andy Grove says: "...if you are a middle manager you [may] face... the fear that when you bring bad tidings you will be punished, the fear that management will not want to hear the bad news from the periphery". Grove continues "Fear that might keep you from voicing your real thoughts is poison. Almost nothing could be more detrimental to the well-being of the company." [5].

Grove's point is that effective corrective action requires accurate information – a reality not unfamiliar to us as automobile drivers: we cannot effectively steer a vehicle on the road if we cannot get accurate data from our body's senses. Thus, an organization that wants to be able to effectively evaluate its activities with processes such as the one described here,

needs to promote the flow of accurate information by ensuring that:

- The process is open and fair (not secretive)
- The staff is briefed about the process and the reason it is being adopted
- The organization promotes a mistake-tolerant culture⁸. Blame and punishment need to be eliminated from the evaluation process (mistakes should be addressed in normal performance reviews alongside successes and achievements).

In Conclusion

Most software catastrophes were troubled projects that went on for too long. Part of the trauma of dealing with them is the realization that “This shouldn’t have happened.”, or “We should have seen it coming.”, and of course the call to action: “Something has to change around here.”

Returning to Johnson’s “Who Moved My Cheese”, the tale continues: *The little people were outraged, shocked, scared, and befuddled when the cheese disappeared. In their comfort, they didn’t notice the cheese supply had been dwindling, nor that it had become old and smelly. They had become complacent.*

How better to describe the failing of a software project. ■

References

1. Bennatan, E., *Wireless Local Loop in Hungary – A Case Study*, New Telecom Quarterly, 2nd Quarter 1997, <http://www.tfi.com/pubs/ntq/auth-BennatanElli.html>

2. Bennatan, E.M., *On Time Within Budget, Software Project Management Practices and Techniques*, Third Edition, John Wiley & Sons, 2000
3. Bennatan, E.M., *The State of Software Estimation: Has the Dragon Been Slain? (Part 1)*, Executive Update Vol. 3, No. 10, The Cutter Consortium, July 2002.
4. Brooks, Fredrick P., *The Mythical Man Month After 20 Years*, Keynote Talk, The 17th International Conference on Software Engineering, April 23-30 1995, Seattle, Washington
5. Grove, Andrew S., *Only the Paranoid Survive*, HarperCollins Business, 1996.
6. Johnson, Spencer, and Kenneth H. Blanchard, *Who Moved My Cheese? An Amazing Way to Deal with Change in Your Work and in Your Life*, Putnam Pub Group, 1998
7. Meliza, Larry L., *A Guide to Standardizing After Action Review (AAR) Aids*, Report number: A348953, US Army Research Institute, Field Unit, Orlando, Florida, 1998, <http://www.stormingmedia.us/34/3489/A348953.html>
8. Sullivan, Gordon R., and Michael V. Harper, *Hope Is Not a Method*, Times Business, Random House, 1996.
9. Farson, Richard E., and Ralph Keyes, *The Failure-Tolerant Leader*, The Harvard Business Review, August 1, 2002.

⁸ For an interesting discussion of a mistake-tolerant business culture, see Harvard Business School Professor Richard Farson and co-author Ralph Keyes’ interesting article “The Failure-Tolerant Leader” [9].