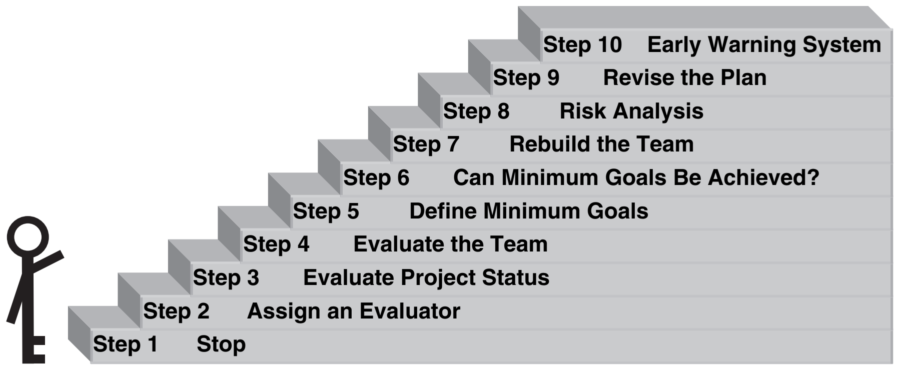
Bennatan_ch01.qxd 3/15/06 11:31 AM Page 1

1

An Introduction to Catastrophe Disentanglement


Step 10 Early Warning System
Step 9 Revise the Plan
Step 8 Risk Analysis
Step 7 Rebuild the Team
Step 6 Can Minimum Goals Be Achieved?
Step 5 Define Minimum Goals
Step 4 Evaluate the Team
Step 3 Evaluate Project Status
Step 2 Assign an Evaluator
Step 1 Stop

In Spencer Johnson's *Who Moved My Cheese* [9], the little people 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 software 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 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 1.1).

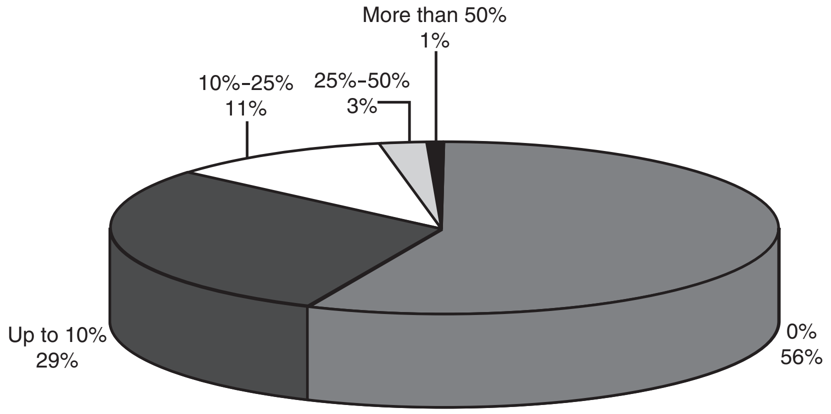


Figure 1.1 Percentage of surveyed organizations' software projects that have been abandoned or cancelled due to significant cost or time overruns in the past three years (source [12])

But obviously, not every overrun or quality problem means a project is out of control, so at what 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? 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*.

One of the best-known attempts to disentangle a multi-hundred-million-dollar catastrophe ended recently, more than a decade after it began. In August

2005, the plug was finally pulled on the infamous Denver airport baggage handling system, in a scene reminiscent of Hal's demise in the memorable Kubrick space odyssey movie.[1] This was a project that had gained notoriety for costing one million dollars a day for being late. One of the interesting questions about the Denver project is why didn't the repeated efforts to save it succeed?

Of all the problems that plagued the project (see [3], [4]), probably the most formidable was the project's unachievable goals. It is unlikely that anyone associated with the project could have brought about a significant change to the goals because the project's extravagant functionality had, in fact, become part of its main attraction. But the ability to define achievable goals is a cornerstone of any catastrophe disentanglement process, without which the process cannot succeed, and that is one of the main reasons the Denver system could not be disentangled.

As indicated by the above survey data, cases like the Denver project are not rare (although few are as extreme). Most development organizations know this even without seeing the survey data. This frustrating reality was expressed in a famous quote from Martin Cobb of the Canadian Treasury Board: "We know why projects fail, we know how to prevent their failure—so why do they still fail?" [10].

Cobb's quote highlights the conventional approach of software engineering. The objective of existing software engineering practices is to *prevent* the occurrence of software catastrophes—that is, to prevent the project from spiraling out of control. As such, the practices have an important role to play in software development. However, more than five decades of experience show that despite these methods, software catastrophes will continue to be around for a while.

When a software project is out of control, there is no PMI, IEEE, SEI, or ISO rescue process to follow because these organizations offer preventive, rather than corrective, solutions. But is such a project necessarily doomed? Will it inevitably collapse in failure? The following chapters will show that this is far from inevitable.

This book fills the void for corrective solutions in software engineering. It deals with projects that are already in serious trouble. In fact, this book is less concerned with how we got into trouble; it is more concerned with how we get out.

1.1 Overview of the Catastrophe Disentanglement Process

Before the first step in disentangling a project can be taken, we must first establish that the whole process is necessary. This means deciding that the project, as it is currently proceeding, has little chance of success without taking drastic measures.

[1] Hal was the wayward computer in Stanley Kubrick's movie *2001: A Space Odyssey*.

Many software organizations have difficulty making this decision, and some avoid it entirely. In fact, there is a general tendency to let troubled projects carry on way too long before appropriate action is taken [6]. Keil [7] uses the term "runaways" to describe software projects that continue to absorb valuable resources without ever reaching their objective. Keil's runaways are, in effect, undiagnosed catastrophes that went on unchecked for much too long. Indeed, the ability to save a project is usually dependent on how early in the schedule a catastrophe is diagnosed. Furthermore, organizations that permit a runaway project to continue are wasting valuable resources. This reality is well demonstrated in the following case.

1.1.1 A Case Study

The FINALIST case, described next, demonstrates how difficult it is to acknowledge that a project is in serious trouble, even when the problem is obvious to almost anyone looking in from the outside. It is an interesting case because it is by no means unique; it demonstrates just how easy it is to become committed to a failing path.

> After the year 2000 passed, and the software prophets of doom faded away, a Canadian software company found itself with almost no customers for one of its small business units. The unit's main expertise was in supporting Cobol programs (where many of the bug-2000 problems were expected to be), and suddenly there wasn't enough Cobol work to support it.
>
> So the company decided to rewrite one of its core products, FINALIST, a large financial analysis system, but it chose to write it again in Cobol in order to retain the company's unique expertise for solving bug-2000 problems (which it still thought would materialize). The new project, appropriately named FINALIST2, was given a 30-month schedule and a team of 14 developers, eight of whom were veteran Cobol programmers.
>
> At the beginning of the second year of the project, two Cobol programmers retired and, soon after, three more moved to another company. With only three veteran Cobol programmers left, the FINALIST2 project began to experience serious problems and schedule delays. The company's management repeatedly resisted calls to reevaluate the project and attempted to get it back on track by conducting frequent reviews, adding more people to the team, providing incentives, and eventually, by extending the schedule.
>
> Finally, 28 months into the project, a consultant was brought in, and his first recommendation was to halt the project immediately. This drastic advice was based on the conclusion that little or no meaningful progress was being made and the project, as it was defined, would probably never be completed. There were not enough experienced Cobol programmers around to do the work, and it was unlikely that new ones would be hired.

> Furthermore, it was unlikely that the new recruits would become sufficiently proficient in Cobol within any reasonable time frame.
>
> The final recommendation was to either restart the project in a modern programming language or to cancel it entirely.

One of the key points in this case is that management failed to notice that what was once a strength (Cobol) had ceased to be one—a classic example of "who moved my cheese." This failure was clearly fostered by a strong desire to preserve Cobol expertise within the company, but it was also the result of a natural reluctance to acknowledge a mistake (resistance to reevaluate the project). These two factors obscured the solution. And so management attempted to fix almost everything (process, team, schedule) except the problem itself.

This case illustrates the difficulties decision makers have in accepting the need for drastic measures and is reminiscent of a gambler who cannot get up and walk away. First, there is the natural tendency to put off making the difficult decision in hope that conventional methods will eventually get the project back on track. A second difficulty involves over-commitment to previous decisions, prompting the investment of more resources to avoid admitting mistakes (this is known as *escalation* [6]).

But troubled projects are never a surprise, and even those most committed to a failing path know that something is severely wrong. But how severe is "severely wrong"? How can we know that it is time for drastic measures? Ideally, there would be a decision algorithm (a kind of software breathalyzer) to which managers could subject their projects, and which would make the decision for them.

1.1.2 Deciding to Rescue a Project

There is no perfect breathalyzer for catastrophes. However, although it is difficult to make a completely objective decision about a project, there are methods that remove much of the subjectivity from the decision. These methods involve an in-depth evaluation of the project and require significant effort. Unlike status reports or regular progress reviews, they are not designed to be applied at regular intervals throughout the development cycle. The process prescribed by these methods is to be applied only when we suspect that a project may be in serious trouble, but we are unsure whether it requires life-saving surgery.

The procedure is based on the evaluation of three basic project areas:

- Schedule
- Budget
- Quality

The procedure examines whether serious problems have existed for quite a while in any of these project areas and whether the situation is getting worse, not better. Any one of these areas can trigger a catastrophe decision, but when this happens, it is not unusual for serious problems to exist in all three. Chapter 2, "When Is a Project a Catastrophe?," covers this subject in detail and also discusses the tricky question of what quality is (the definition will be based on the level of product defects and the degree to which customers or users are satisfied with the product).

Once the decision has been made that a project is indeed a catastrophe, the options become more clear: save it or lose it. This is the time for the ten-step disentanglement process.

1.1.3 The Disentanglement Process

The disentanglement process is designed to rescue a seriously troubled project, provided it can establish business or strategic justification for doing so. The process is built around two main figures: the *initiating manager* (who initiates the process and oversees its implementation) and the *project evaluator* (who leads and implements the disentanglement process). The initiating manager is an insider, a senior manager in the organization that owns the project. The project evaluator is an outsider, a seasoned professional, reliable, and impartial.

The catastrophe disentanglement process consists of the following ten steps:

1. **Stop:** Halt all project development activities and assign the team to support the disentanglement effort.
2. **Assign an evaluator:** Recruit an external professional to lead the disentanglement process.
3. **Evaluate project status:** Establish the true status of the project.
4. **Evaluate the team:** Identify team problems that may have contributed to the project's failure.
5. **Define minimum goals:** Reduce the project to the smallest size that achieves only the most essential goals.
6. **Determine whether minimum goals can be achieved:** Analyze the feasibility of the minimum goals and determine whether they can reasonably be expected to be achieved.
7. **Rebuild the team:** Based on the new project goals, rebuild a competent project team in preparation for re-starting the project.
8. **Perform risk analysis:** Consider the new goals and the rebuilt team, identify risks in the project, and prepare contingency plans to deal with them.
9. **Revise the plan:** Produce a new high-level project plan that includes a reasonable schedule based on professionally prepared estimates.
10. **Install an early warning system:** Put a system in place that will ensure that the project does not slip back into catastrophe mode.

There are three main reports generated by the project evaluator during the disentanglement process:

1. **Step 4: The team overview document**

   The document contains a summary of the project team evaluation. It is used as input to step 7 (“rebuild the team”). The overview includes the main sources of information, the list of interviews, the reasoning that led to any significant findings, and any problems or incompatibles that arose during the evaluation.

2. **Step 6: The midway report**

   The document is generated midway through the disentanglement process after establishing the feasibility of the minimized goals. This provides senior management and other key stakeholders with a formal update on the progress of the disentanglement process. The report documents all major decisions, evaluations, and conclusions that produced the new reduced-scope project. It also includes summaries of the discussion that led to agreement among the key stakeholders.

3. **At the end of the disentanglement process: The final report**

   Producing this report is the project evaluator’s last task. The report summarizes all information collected and generated, all decisions made, all major project documents produced, and lists all problems that were resolved or left unresolved. This report is produced even if the disentanglement process does not succeed or if the project is cancelled.

The sequence of the disentanglement steps is organized according to the logical flow described in Table 1.1. It is important to complete the steps in this sequence (though parts of the steps may overlap). The following points demonstrate why the sequence is important:

- There will not be enough information to propose new goals until the project has been evaluated (this includes both the project status and the team).
- There will not be enough information to rebuild the team until the new project goals have been established.
- There will not be enough information for a new plan (schedule and estimates) until the new project goals have been established, the team has been rebuilt, and the risks have been identified.

Table 1.1 Logical Flow of the Ten Disentanglement Steps

Phase	**Steps**
Launch the process	1, 2
Evaluate the status	3, 4
Introduce changes	5, 6, 7
Prepare to resume	8, 9, 10

Each one of these steps is described in detail in the following chapters. Their success is strongly dependent on the cooperation of all involved parties and the active involvement of the project team. But the main precondition for success is the support of the organization's senior management. As we shall see in the following chapters, without effective management support, the process will fail at almost every step.

The entire process should take no more than two weeks to complete (see the disentanglement timeline in Figure 13.1 of Chapter 13, "Epilogue: Putting the Final Pieces in Place"). This also represents the maximum amount of time that the project will remain halted.[2]

1.2 A Closer Look at the Data

We have seen in Figure 1.1 that software catastrophes are not rare, but the data in Figure 1.1 does not tell the whole story. Could these projects have been saved if a formal disentanglement process (like the one we described earlier) had been used in time? An indication can be found by looking at additional data related to the schedule, budget, and quality of the projects (these are the factors that would have triggered the process).

The data in Table 1.2 is based on a broad software development survey [8] that defined a schedule overrun of more than 50% as severe, a budget overrun of more than 50% as severe, and the quality problems of a product with critical post-release defects as being severe. These projects are considered failures even though they were permitted to run their course to completion (and many would submit that they should not have been permitted to do so).[3]

- **Schedule:** The data clearly shows that severe schedule overruns are far from rare. In a quarter of the surveyed software organizations, more than 10% of the projects had severe schedule overruns. In 13% of these organizations, the situation was much worse: more than a quarter of their projects had severe schedule overruns.
- **Budgets:** The data for software project budgets is just a shade better. In just less than a quarter of software organizations, more than 10% of the projects were severely over budget. In 8% of these organizations, more than a quarter of the projects were severely over budget.

[2] Unless, of course, the project cannot be saved and gets cancelled.

[3] The most common argument is that organizations that permit overruns of more than 50%, or the release of products with severe quality problems, have little chance of survival.

- **Quality:** The data for quality does not tell a good story. More than a third of software organizations (35%) had severe quality problems in more than 10% of their products after their release. Of these, 15% reported severe quality problems in more than a quarter of their products after their release.

Table 1.2 The Proportion in Software Development Organizations of Software Projects with Severe Problems (Source: The Cutter Consortium, 2005)

Severe Schedule Problems		**Severe Budget Problems**		**Severe Quality Problems**	
More than 10% of projects were very late	25%	More than 10% of projects were greatly over-budget	24%	More than 10% of projects had critical quality problems	35%
More than 25% of projects were very late	13%	More than 25% of projects were greatly over-budget	8%	More than 25% of projects had critical quality problems	15%

After a severely troubled project is completed or cancelled, there is, of course, no way of telling whether the outcome could have been different. However, we speculate that many of these severely troubled projects could have been rescued. At the very least, such projects would have greatly benefited from an early warning system. The survey findings are from projects that were more than 50% over-schedule or over budget or had critically severe quality problems. The warning system, which is further discussed in Chapter 12, "Step 10—Create an Early Warning System," is designed to trigger an alarm whenever such conditions begin to develop.

Interestingly, the survey found that 50% of software development organizations do implement some type of project rescue process. These companies reported that they handle early indications of project failure by initiating a formal project reevaluation process resulting in possible changes to goals, plans, and the development team. These are precisely the elements of the catastrophe disentanglement process presented in this book. Furthermore, according to another survey finding shown in Figure 1.2, 45% of organizations almost always succeed in getting troubled projects back on track. We can speculate that they overlap the 50% that conduct rescue processes.[4] These, then, are organizations that have independently developed catastrophe disentanglement processes, and have apparently applied them to great effect.

[4] Without the need to speculate further, we can state that the 45% is a low figure; there are certainly more successful project rescues within the 30% who responded that "some are saved and some are not." So the overlap may look even better.

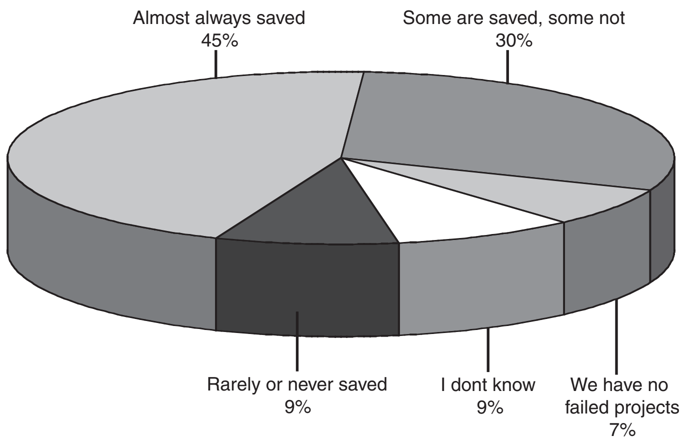


Figure 1.2 How frequently are troubled projects saved?

The survey also looked at the cost of catastrophes. When asked to assess the impact of software project failures on their organization, the most common response lamented the waste of funds, time, and other resources. Others reported a decline in the organization's motivation and prestige and the loss of customers and business opportunities. Clearly, the high cost of project catastrophes goes way beyond the cost of the failed project itself. With an effective disentanglement process, it is practically certain that much of these costs could have been avoided.

1.3 Tips Before You Proceed

There is no magic or mystery in the ten steps of the catastrophe disentanglement process. They are all familiar steps that many software developers will have used at various times in their careers. The strength of the process is in combining the steps together, implementing them as a single aggregate with each step building on the previous ones, and doing so within a short, fixed schedule.

Before proceeding with the implementation of the process, here is a summary of several of the tips that are provided throughout the discussion of the ten disentanglement steps. These tips and others are elaborated at length in the upcoming chapters.

1. **Work in parallel.**

   Though the ten steps of the process need to be completed in sequence, they do not need to start in sequence. In fact, parts of some of the steps can be implemented in parallel to others (this can save time). For example, step 8 (risk analysis) can begin as soon as step 3 (evaluate project status) is complete, and much of steps 5 and 6 (define and evaluate the feasibility of minimum goals) can be implemented in parallel.

2. **Expect resistance**.

   Expect resistance to change; it is natural. The best way to deal with resistance is by enlisting the support of allies from among the project stakeholders and the project team. In cases where resistance is particularly high, enlist the support of senior management.

3. **Be sensitive to the team and to the stakeholders.**

   Be sensitive to the emotional concerns of the project team and stakeholders. Team members will have legitimate job security and career concerns. Some of the stakeholders will have personal interests in the project that will not necessarily be financial or business related. Before proceeding, become familiar with the key stakeholders and team members.

4. **Keep within the schedule.**

   The process can easily slip well beyond the allocated two weeks; this will happen one day at a time, and at some point the delay is no longer manageable. Treat any delay (even of a half day) as a problem that needs to be immediately corrected.

   Overcome delays by working whenever possible at a high level (leaving details to be filled in by the team after the project resumes), by using a project evaluation *team,* and implementing the disentanglement steps in parallel. If delays are caused by a lack of cooperation, enlist the immediate help of the organization's senior management.

5. **Do not proceed without senior management support.**

   The disentanglement process cannot succeed without firm visible support from senior management. The process requires significant cooperation from all involved parties, and this will not be assured without such senior management support.

   The process also involves activities that will generate resistance from the project stakeholders and the development team. In some cases, it will be difficult or even impossible to overcome the resistance without the support of senior management.

6. **Encourage all involved parties to review the disentanglement process.**

   The disentanglement process is more likely to succeed if all involved parties understand how it works and why each step is being implemented. Thus, while the description in the following chapters is directed toward the project evaluator and the initiating manager, all parties involved in the effort to rescue the project will benefit by understanding the process.

7. **Document decisions and findings.**

   All key decisions and all major findings should be documented. This will save time, should the decisions need to be reevaluated or explained. The decisions and findings document should be maintained by the project evaluator and submitted to the initiating manager at the end of the process.

8. **Be open and accessible.**

   Many of the concerns and much of the reluctance to cooperate can be overcome by conducting the disentanglement in an open and candid manner. This means no clandestine decisions and no behind-closed-doors meetings except in rare occasions when topics of a purely personal nature are discussed (they should be kept to a minimum).

9. **Listen to arguments.**

   Be prepared to listen to arguments before decisions are finalized, provided they are of a professional nature (exclude political and personal interest viewpoints). After decisions have been made, be prepared to re-open them only if significant new information becomes available that was not previously considered. Be resolute about preventing undue delays in finalizing discussions and decisions.

10. **Not all problems discussed will occur.**

    The following chapters provide guidelines for resolving many problems that may arise during the disentanglement process. This can be alarming. Remember, not all problems will actually occur—in fact, most will not. The guidelines are like a first aid kit; just because you carry anti-venom serum doesn't mean that you will be bitten by a snake.

11. **The key to success is a good evaluator**.

    Of all the factors that affect the disentanglement process, the two that most contribute to its success are senior management support (discussed earlier) and a good project evaluator. Start the search for evaluator candidates even before a final decision has been made to proceed with the disentanglement process.

12. **Read through the entire process**.

    Read through the entire process before proceeding. Many of the steps are inter-dependent. You can better implement each step if you understand the steps that follow.

One final point: There are no shortcuts. The disentanglement process is designed to be implemented in its entirety. Each step relates to the evaluation or resolution of a problem that, if left unsolved, is likely to disrupt the entire disentanglement process. Furthermore, several of the steps are inter-dependent. In fact, the final step (install an early warning system), which ensures that the project does not slip back into catastrophe mode, is dependent on the preceding nine steps.

1.4 Summary

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. But obviously, not every overrun or quality problem means a project is out of control, so at what 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*.

Before the first step in the disentanglement process can be taken, we must first establish that the whole process is, indeed, necessary. This means deciding that the project, as it is currently proceeding, has little chance of success without taking drastic measures.

There are methods that can help remove much of the subjectivity from this decision. The idea is not to define an algorithm and subject projects to it every week, but rather to provide a procedure to be applied only when we suspect that a project may be in serious trouble and we are unsure if it requires drastic life-saving surgery.

The procedure is based on the evaluation of three basic project areas: schedule, budget, and quality. The procedure examines whether serious problems have existed for quite a while in any of these project areas and whether the situation is getting worse, not better.

The disentanglement process is built around two main figures: the *initiating manager*, who initiates the process and overseas it as it is being implemented, and the *project evaluator*, who leads and implements the disentanglement process.

The ten steps of the catastrophe disentanglement process are

1. Stop.
2. Assign an evaluator.
3. Evaluate project status.
4. Evaluate the team.
5. Define minimum goals.
6. Determine whether minimum goals can be achieved.
7. Rebuild the team.
8. Perform risk analysis.
9. Revise the plan.
10. Install an early warning system.

The ten steps should be completed in sequence, and the entire process should take no more than two weeks to complete.

The following list summarizes several tips for the successful implementation of the disentanglement process:

1. Work on the steps in parallel.
2. Expect resistance from stakeholders and project team members.
3. Be sensitive to the team and to the stakeholders. Before proceeding, become familiar with the key stakeholders and team members.
4. Keep within the two-week disentanglement process schedule.
5. Do not proceed without senior management support. The process cannot succeed without it.
6. Encourage all involved parties to review the disentanglement process. The process is more likely to succeed if all involved parties understand how it works.
7. All key decisions and all major findings should be documented.
8. Be open and accessible. This will reduce concerns and any reluctance to cooperate.
9. Be prepared to listen to arguments before decisions are finalized.
10. Remember that not all problems discussed here will actually occur—in fact, most will not.
11. The key to success is a good evaluator. Start the search early.
12. Read through the entire process before proceeding. Many of the steps are inter-dependent.

There are no shortcuts in the disentanglement process. The process is designed to be implemented in its entirety. Each step relates to the evaluation or resolution of a problem that, if left unsolved, is likely to disrupt the entire disentanglement process.