

## **Dealing With Dilemmas In Software Engineering**

John Sambrook  
Common Sense Systems, Inc.  
[john.sambrook@common-sense.com](mailto:john.sambrook@common-sense.com)

### **Abstract**

Software engineering is a complex and demanding activity. The degree to which engineering teams are able to recognize dilemmatic situations and resolve them quickly and without compromising important needs has a large impact on the performance of the team and the quality of the developed software. For this reason, providing engineering teams with education and training in dealing with dilemmas and establishing accountability for not compromising on important needs is a logical and practical means to improve their performance. In this article we explore the nature of dilemmas and provide actionable advice that individuals and groups can use to improve their ability to recognize and resolve dilemmas in an effective manner.

### **1. Introduction**

Engineers and managers are confronted with significant dilemmas as they work to create their deliverables on time, on scope and on budget.

Common dilemmas in software engineering include (but are by no means limited to):

1. How much “safety” should be included in the schedule?

Engineering teams want to be able to honor the project promises they make. This usually drives them to want to increase the amount of safety built into the schedule. But other groups within the company also want to earn revenue as quickly as possible, which drives a desire to reduce the amount of safety built into the schedule.

2. What capabilities will be included in the software?

In order to be embraced by new and/or infrequent users of the software there is a strong desire to keep the software simple. But the desire to also appeal to “power users” and/or to match the competition feature-for-feature typically adds complexity to the developed product.

3. Should we make this software component general or specific?

On the one hand, engineering teams want to make software components general, so they can be adapted to new uses in the future and so they can accommodate unforeseen demands. But on the other hand, engineering teams also want to make software components specific to the task at hand, so they are not doing work that may never be needed.

4. How much test code should we write?

In order to deliver a high-quality product engineering teams often invest a great deal of time and energy in the development of test code and testing frameworks. However, engineering teams often have their hands full with implementing the required features on time. This limits the resources available to develop test code and testing frameworks.

In addition to these very common and well-known problems, individual engineers often routinely deal with situations they may not recognize as dilemmas at all:

1. Should I check the arguments to this method or not?

Checking pre-conditions and post-conditions is a common way to ensure that out of range errors are captured near their origin. This technique often reduces the time spent diagnosing software defects. For this reason some developers become strong advocates of including code of this kind.

However, adding code of this kind takes time and effort and also incurs run-time time and space penalties. Developers can also be uncertain as to when a condition should provoke an assert as opposed to signaling an expected error condition. This can discourage developers from adding this kind of checking.

2. Should I rewrite this code or not?

Developers often have to decide whether a particular section of code – or perhaps a whole module or group of modules – should be rewritten.

On the one hand, redesigning code provides an opportunity to improve it. When code is modified over time it almost always becomes more obscure and less maintainable than it was originally. There is also the situation where changing requirements argue for rewriting the code.

On the other hand, developers have to wrestle with the question of how many defects will be introduced in the new code and whether the new code will be a net improvement. Developers also have to consider whether they have the resources to rewrite the code. They are often fully loaded and this tends to argue against undertaking significant rewriting efforts.

3. Should I try to find the root cause or not?

People unfamiliar with software development are often surprised to learn that software developers don't always have a crystal-clear understanding of how all of the code in a given project works. Modern projects often have hundreds of thousands of lines of complex code, so it is no surprise that people often have only a vague understanding of how some section of the code works.

As a result, when a developer makes a specific change s/he may be surprised to find out that the code suddenly appears to work. If the reason for this is not clear, the developer is confronted with a dilemma.

On the one hand, the developer wants to understand why the change seemed to fix the problem. This brings the benefit of improved understanding of the code base and the opportunity to check for unanticipated side-effects.

On the other hand, developers that are pressed for time or that do not expect to do ongoing work in some area of the code may conclude that it is not a good use of their time to continue the investigation as to why the code is now working.

These dilemmas and many others are extremely common for developers. There is also likely to be significant variation within the development team as to the criteria by which these decisions are made.

It should come as no surprise that many software developers simply love to write software. While this is wonderful, it may also be the case that rewriting a particular component serves the needs of the software developer but not the needs of the project. In this case, recognizing this as a dilemmatic situation – in an environment where people are skilled at dealing with dilemmas – is more likely to lead to an outcome where both the needs of the developer and the needs of the project are fully satisfied.

There is also a long history – documented in the literature – of people searching for a “silver bullet” solution to the problems inherent in software development. So far, no such solution has been found. Software development remains a complex and challenging domain.

This brings us to conclude that:

1. Dilemmas are ubiquitous in software engineering.
2. No silver bullet solution is likely to be found in the near future.

These realities – along with people’s natural tendency to resist change – often conspire to cause engineering teams to avoid taking a hard look at seemingly basic issues such as how they recognize and resolve dilemmas.

This is a risky position to take. It can block potential avenues of improvement – and software engineering teams must improve. Competition is fiercer than ever. Offshore software development is seen more and more as a viable option for many companies, and software engineering can now be purchased on services such as eBay.

In this article, we provide information on the nature of dilemmas and then provide actionable advice that teams can use to improve their ability to recognize and resolve dilemmas in an effective manner.

## **2. The Nature of Dilemmas**

Most people have an intuitive understanding of dilemmas. Given that dilemmas are all around us, this is not surprising. Relatively few people, however, can give a precise description of the elements of a dilemma (its composition or essential nature), or how to detect the existence of dilemmas in everyday business situations.

In this section we will take a close look at dilemmas. The material here draws on work done by Goldratt and others in developing the “Theory of Constraints” (TOC).

### **2.1. Overview**

Dilemmas are interconnected beliefs about a given situation. Dilemmas do not have a physical manifestation of their own. Team members with similar beliefs will generally perceive the same set of existing dilemmas.

Lacking a physical manifestation, dilemmas cannot be sensed directly. We cannot see, touch, taste, feel or hear dilemmas. For this reason, we must be prepared to recognize the presence of a dilemma based on its impact on the environment around it.

The beliefs that make up a dilemma can be characterized as being actions, needs or objectives. These entities are linked together by other beliefs (assumptions) about reality. Once these basic components are revealed (identified), we can work systematically on breaking the dilemma.

With this overview we are now prepared to consider more carefully the nature of dilemmas.

### **2.2. Actions**

When we sense a dilemma, it is usually because we have discovered that two actions seem to conflict with each other in some way.

Actions sometimes conflict by being opposite in effect:

*Charles and Mark are developers working on the same project. Charles is concerned that the software is hard to debug. Charles wants the group to agree to make more frequent use of assert statements in their code. Mark is concerned that adding asserts is going to make the code too slow. Mark wants the group to avoid using asserts and spend more time simply testing the code.*

In this example Charles' preferred action is to "Increase the use of asserts in the code base." Mark's preferred action is to "Reduce the use of asserts in the code base."

These actions are opposite in effect and therefore in conflict with each other. One can easily imagine how a new developer on the team could receive "mixed signals" as to when to use assert statements and when to avoid them.

Consider also the likely effect if this dilemma is allowed to exist for many years<sup>1</sup>. The result will be a code base that is schizophrenic. Because modules are usually created by one person and then only "tweaked" thereafter, some modules will be almost devoid of asserts, whereas other modules will make copious use of them.

Opposite-in-effect dilemmas are dilemmas where we don't know whether doing more or less of something is the right action.

Actions can also conflict with each other by being mutually exclusive:

*The final build of the project software has been completed. The software is almost completely validated when a serious defect is found. The team is forced to decide whether they should release the existing software as-is or stop work and create a new release that fixes the defect.*

In this example the actions are perceived as "We release the software as-is" or "We fix the defect and create a new release."

To summarize, an important part of any dilemma is that there are two<sup>2</sup> actions perceived to be in direct conflict with each other.

### **2.3. Needs**

The actions present in any dilemma are always related to needs of some kind. How do these needs arise?

People naturally try to conserve their own energy. They want to ensure that their "energy investments" will serve them in some way. Thus, when people take actions, it is almost always to meet or protect one or more needs important to them.

This reality applies to groups of individuals as well. In general, groups act to ensure that the needs of the group will be met or protected.

From this we can conclude that the actions inherent in every dilemma are connected to needs. More specifically, when a person or group is advocating that a certain action be taken or continued we can infer that they are doing so in order to meet or protect needs important to them.

---

<sup>1</sup> And this is extremely likely. The presence of a dilemma has to be inferred from its effect on the surrounding environment. But most people have not been trained to recognize dilemmatic situations and to deal with them effectively. Thus small problems such as this one can live on for years without ever being addressed head-on.

<sup>2</sup> In mutual-exclusivity dilemmas there can be any number of possible actions. For simplicity we address only the most common (two alternative) case in this article.

Think about our example from section 2.2:

*The final build of the project software has been completed. The software is almost completely validated when a serious defect is found. The team is forced to decide whether they should release the existing software as-is or stop work and create a new release that fixes the defect.*

One of the actions in this scenario is “We release the software as is.” What need is met or protected by this course of action? A reasonable answer would be “Position ourselves to earn revenue from sales of the product.”

The other action in this scenario was “We fix the defect and create a new release.” What need does this meet or protect? In this case, the need could be stated as “We provide high quality products to our customers.”

To summarize: Associated with every action is a need that the action is intended to meet or protect.

## 2.4. Common Objectives

In any true dilemma there is always a common objective that is met or protected when and only when all of the needs inherent in the dilemma are also met or protected.

Let’s say that we have a situation where one group – say, Marketing – wants to take some action in order to meet a need that is important to them. Another group – Engineering – wants to take a conflicting action in order to meet a different need that is important to them.

If the situation described represents a true dilemma for Engineering and Marketing, then neither side will just march ahead and implement their preferred action. Instead they will meet and try to find some acceptable compromise<sup>3</sup>.

What prevents each side from just marching ahead and implementing their preferred action? Why do they not just ignore the other group and do what serves them?

What blocks them from taking these destructive actions is the presence of a common objective. People or groups that are part of the same system always have a common objective – even if they fail to recognize it.

In a true dilemma there is always a common objective. A common objective is a goal that is dependent on the needs of both “sides” of the dilemma. That is, both of the need have to be met if the common objective is to be achieved.

For example, consider the following dilemma:

Mark and David are software engineers. They are discussing a proposed change to the system software they are developing.

Mark is proposing a complete rewrite of the graphics library. The existing library is old and has been patched fairly often. Mark is concerned that it might contain defect(s) that are discovered only after the product has shipped. Shipping a product with such a defect could force an expensive and embarrassing product recall.

David, on the other hand, believes that the library is good enough. He is opposed to rewriting it. His concern is that the project schedule is already tight

---

<sup>3</sup> This is a critical point. When the needs are legitimate and critical to the organization the idea that either one should be compromised is wrong. But this is the inevitable result when people don’t know how to avoid compromising.

and that taking on this unplanned activity is going to make it even more difficult to finish on time, on scope and on budget. Not shipping on time would mean lost revenue that would never be recovered.

What is the common objective in this example? What do both Mark and David want to achieve?

In this example, both David and Mark are trying to “Do what serves the company.” Mark is trying to serve the company by taking action to avoid a potential recall. David is trying to serve the company by taking action to create a saleable product.

So both Mark and David are trying to ensure that two different legitimate needs of the company are met. Unfortunately, the actions they believe need to be taken to meet these needs are in conflict<sup>4</sup>.

It is important to note that there is no conflict between the needs themselves. Both needs are legitimate and necessary. It is the actions that are in conflict. If Mark or David could find an alternative action to meet their need that does not conflict with the other action, the dilemma would cease to exist.

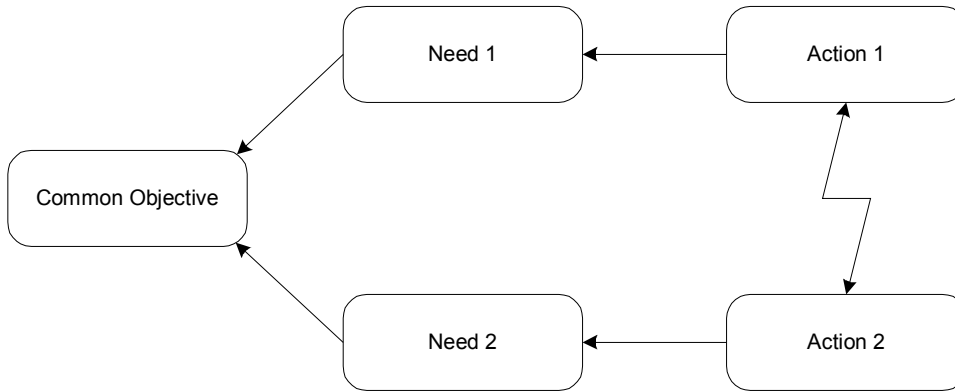
At this point we have proposed that dilemmas consist of actions, needs, and a common objective. In the following section we present a diagrammatic representation of a dilemma.

---

<sup>4</sup> In the Theory of Constraints (TOC) body of knowledge, dilemmas are called “conflicts” for exactly this reason. But this choice of terminology sometimes creates confusion. People new to the terminology often assume that a conflict is an argument of some kind. While TOC conflicts often cause people to argue, a TOC conflict is not an argument.

## 2.5. Diagramming Dilemmas

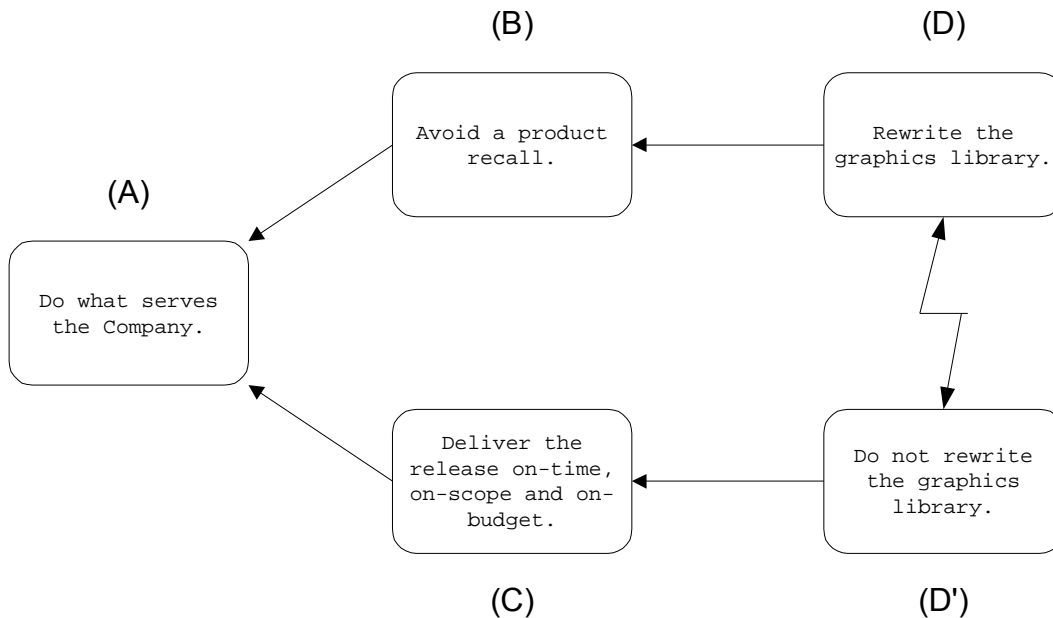
Figure 1 illustrates a graphical representation of a dilemma:



**Figure 1: General Structure of a Dilemma**

Of note in this diagram are the arrows connecting various entities. These arrows are arrows of “implied logical necessity.” That is, by drawing the arrows we are signaling our belief that the source entity must exist in order for the destination entity to also exist. The jagged arrow between the actions is meant to indicate that the actions are seen as being in conflict with each other.

Diagramming an actual dilemma includes describing the actions, needs and common objective specific to the dilemma. Figure 2 illustrates the dilemma Mark and David faced regarding the graphics library used by their project:



**Figure 2: The Graphics Library Dilemma**

In this figure the arrow connecting action (D) with need (B) indicates that at least some of the people captured by this dilemma believe that in order to avoid a product recall they must rewrite the graphics library. In like manner, the arrow connecting action (D) with need (C) indicates that at least some of the people believe that if they are to deliver on time, on scope and on budget, they must not rewrite the graphics library.

Each arrow on a dilemma diagram represents a belief. The straight arrows represent beliefs of the form “In order to have ... I must ....” The single jagged arrow (the “conflict arrow”) represents the belief that “Action ... is in direct conflict with action ....”

The dilemma exists – in the minds of the people captured by it – only as long as they believe that these beliefs are valid. In order for them to resolve the dilemma they will need to convince themselves that at least one of their beliefs (one of the arrows) is invalid.

If we are to show that a given belief is invalid, we need to explore the assumptions behind it and find at least one that is invalid or that could be made invalid by an additional action<sup>5</sup> that we will take.

For example, if we were to ask why rewriting the graphics library is necessary in order to avoid a recall we would likely expose<sup>6</sup> several assumptions:

1. Rewriting the library would actually improve it.
2. Our rewrite of the library would not contain major defects not present in the current library.
3. We can't test the library code well enough to ensure no recallable issues exist.
4. The only way to get the kind of graphics library we need is to write it ourselves.

Diagramming dilemmas and the importance of exposing and systematically exploring assumptions will be discussed in more detail when we consider what it means to resolve a dilemma in an effective manner.

## 2.6. Expressing Dilemmas in Written Form

Dilemmas can be expressed in words as well as diagrams. Consider again the dilemma represented by Figure 2. This dilemma could be written in this form:

*“Our objective is to do what serves the company.*

*In order to do what serves the company, we must avoid a product recall. In order to avoid a product recall, we must rewrite the graphics library.*

*On the other hand, in order to do what serves the company, we must also deliver on time, on scope and on budget. In order to deliver on time, on scope and on budget, we must not rewrite the graphics library.*

*The dilemma is clear. We need to both rewrite and not rewrite the graphics library and we can't do both.”*

---

<sup>5</sup> TOC refers to these actions as “injections.” In this article, we generally refer to them as “ideas for resolving the dilemma” or simply as “ideas.”

<sup>6</sup> In TOC this process is called “surfacing” assumptions. To surface an assumption is to make it an explicit part of the problem statement. In contrast, submerged assumptions are the assumptions that are not an explicit part of the problem statement and the ones on which, presumably, everyone agrees.

Expressing a dilemma in written form paves the way for verbalizing it – to say it aloud. Verbalizing a dilemma is an excellent way to find any minor errors in how the dilemma is stated. In general, it is necessary to “speak it and tweak it” a few times until the dilemma “sounds right” or “rings true” for the people captured by it.

### **3. Dealing With Dilemmas**

In order to deal with dilemmas effectively a software engineering team must be able to:

1. Recognize dilemmas.
2. Assess dilemmas.
3. Clarify dilemmas.
4. Resolve dilemmas without compromising.

We consider these issues in the following sections.

#### **3.1. Recognizing Dilemmas**

As stated previously dilemmas cannot be sensed directly. They can be likened to cosmological Black Holes – we can’t see them but we must infer that they are present based on what we can observe about the environment around them. In this section we discuss the “key indicators” that usually reveal the presence of a dilemma.

Training people to spot these key indicators improves their ability to recognize dilemmas in real life.

##### **3.1.1. “I Have a Problem”**

“Problems” are the most immediate and direct indicators of dilemmas. When someone says “I have a problem” what they are really saying is “I have a dilemma.” Problems are situations where we recognize that needs are threatened.

A small problem is a dilemma that doesn’t jeopardize important needs. A large problem is a dilemma that creates great risk to our most important needs.

So one way to recognize dilemmas is to be open to receiving evidence of problems.

##### **3.1.2. Stagnation**

When actions are in conflict, people often refrain from taking any action until the conflict is eliminated.

Consider the developer facing this dilemma: “In order to start my next task, I must release my work to the build. On the other hand, in order to avoid being punished<sup>7</sup> for breaking the build, I must not release my work to the build.” This dilemma results in stagnation for the developer until it is resolved.

Organizations can become stagnated as well. For example, a company may sense that it is in a dying market and yet not be able to bring itself to develop products for a new market. That organization will stagnate until the dilemma is resolved, one way or the other.

---

<sup>7</sup> Or some other disincentive, such as being embarrassed in front of ones peers, or becoming the scapegoat if the build doesn’t work anymore, etc.

### 3.1.3. Oscillation Between Actions

Sometimes a dilemma causes people to oscillate, or flip-flop, between the two actions of the dilemma.

For example, in the early stages of a software project the need to “Have a quality product” typically outweighs the need to “Ship on time.” And so in the early stages of a project people generally spend more time on actions that they believe will lead to a quality product – such as learning new design patterns or attending a seminar on a new development tool.

Over the course of the project, however, **the relative importance of the needs changes**. As the project moves towards its ship-date, having a quality product becomes less important than shipping on time.

And so, during the “crunch period” no one is going to any seminars or reading books on patterns. Instead, they are doing all they can to ensure the product will ship on time. Their preferred actions have changed.

When the product finally ships the original priorities will be restored. People will again be prioritizing actions that lead to a quality product above the need to ship on time. But again, as the time to ship the second product draws near, the preferred actions will change.

For this reason whenever we see people “oscillating” between different actions, we should suspect the presence of an underlying and unresolved dilemma.

### 3.1.4. “Compromise Speak”

Use of the word “balance” (or “trade-off”) is a strong indicator of an underlying dilemma.

For example, whenever someone uses the word “balance” as in “We need to balance the desire to add features against the need to make the ship date” it is virtually certain that (1) there is an underlying dilemma and (2) they have already concluded that the best way to solve it is by compromising. That is, they don’t believe there is a way to do both – to add all the features and to still make the ship date.

This is wrong. When the needs inherent in a dilemma are important the first course of action should not be to seek an acceptable compromise. The first course of action should be to recognize the situation as a dilemma and to begin the process (outlined later in this paper) for resolving it without compromising.

### 3.1.5. Arguments Between People or Groups

When people sense that their important needs are at risk they usually do all they can to ensure their needs will be met. Dilemmas are situations where needs are at risk. Therefore, when we recognize that people are arguing we should expect there is a dilemma causing them to argue.

### 3.1.6. Denigrating the Actions of Others

“Why is Joe constantly trying to use the Standard Template Library in every single piece of code he writes? Every time I turn around he is adding more code that uses it! Doesn’t he care about code-bloat? Doesn’t he care about compile times? Pretty soon it’s going to take us all night just to compile the code and another day to load it!”

From time to time people take actions that surprise us, inconvenience us, or perhaps cause us to question their competence. This can lead us to denigrate the actions of others.

But most often the real problem is that they are working to meet a need we either have not recognized or a need we consider unimportant.

Whenever you sense that someone is denigrating the actions of others it is a good idea to consider whether the criticism is actually warranted or whether the presence of an underlying dilemma would explain the behavior.

In this case it could be that Joe is trying to meet an important need – such as improving the quality of the code by using very powerful and well-designed libraries. But Joe also needs to be aware of the issues raised by his colleague – code-bloat and compile-time are also issues that have to be considered.

Will Joe and his colleague resolve this situation in a way that ensures that all needs will be fully met? Or will they suffer in silence and/or try to find some sort of “acceptable compromise?”

### 3.2. Assessing Dilemmas

Not all dilemmas are serious dilemmas. Dealing with a dilemma takes time and effort and for many of the dilemmas we encounter we can either break<sup>8</sup> them entirely in our heads or we can simply accept the consequences of not breaking them.

That being said, we should be assessing the dilemmas we encounter and judging whether they are jeopardizing important needs. In order to do this we need to be able to build dilemmas quickly in our heads and contemplate the needs that are at risk of being compromised.

This begs the question of “How do we know if a given need is important or not?”

Engineering teams that have invested the effort to decompose high-level objectives into sufficiency-based logic trees<sup>9</sup> will have a very clear understanding of what they need and exactly how their needs are related to each other.

But many engineering teams do not do this. The use of these techniques is relatively new and their value is not recognized by the majority of engineering teams at this time. However, a truly experienced team will have an intuitive sense of what needs are truly important and what needs are not.

In the final analysis, in order to know whether a given need is important or not one must understand the cause and effect relationships that exist within the problem domain. Only then is a person prepared to answer the question “What will be the likely effects of not meeting this need?”

Finally, we must also be prepared for situations in which something is stated as a legitimate need when it is not. People sometimes try to have a personal need met by casting it as a legitimate need of the project.

---

<sup>8</sup> To “break” a dilemma is to resolve it without compromising either need.

<sup>9</sup> A description of these logic trees and how they are created is outside of the scope of this article. Details are available on request from the author.

### 3.3. Clarifying Dilemmas

Dilemmas must be clarified before real work can begin on resolving them. To clarify a dilemma is to identify the actions, needs, and common objective; then to express them either as a diagram or as precisely worded text.

Clarifying a dilemma is important for several reasons:

1. It allows people to focus on the whole problem instead of just the actions that are in conflict. Being able to see the whole problem instead of just a part of it allows people to search for solutions in a larger problem space.
2. It helps to “buy time” to actually think about the problem. As we ask questions like “What need is this action intended to protect?” we are clarifying our understanding of the problem space.
3. It helps people recognize that their concerns have been accurately heard. Once they know that their needs have been heard, they are able to move on and actively contribute to the solution of the problem itself.
4. It changes the problem from “me vs. you” or “us vs. them” into “all of us against the problem.” When people recognize that they have a common objective they are more able to focus on the problem itself.

It has been said that “Define a problem precisely and you are half-way to solving it.” Nowhere is this truer than when we are talking about a serious problem that has been dividing a software engineering team into factions for months or years.

### 3.4. Resolving Dilemmas Without Compromising

Compromising on important needs is always easier than ensuring they will be fully met:

1. It’s always easier to do a quick hack than to implement a well-designed solution.
2. It’s always easier to assume the code will work than to prove that it does.
3. It’s always easier to do what serves us than to do what will serve the customer.

And so if the goal of an individual or an organization is to follow the path of least resistance then constantly compromising on important needs seems quite logical.

But logically, if we are always compromising on important needs, can we ever be really successful?

In reality, if we want outstanding performance from any system we must ensure that important needs of the system are not constantly being compromised.

Ensuring that the important needs of the system are not compromised can be broken down into a two step process:

1. Identify the dilemmas and determine how they should be resolved.
2. Implement the actions necessary to resolve the dilemma in reality.

A great many of the dilemmas encountered in real life can be completely resolved quite easily. For these dilemmas, implementing the actions necessary to resolve the dilemma in real life is not difficult.

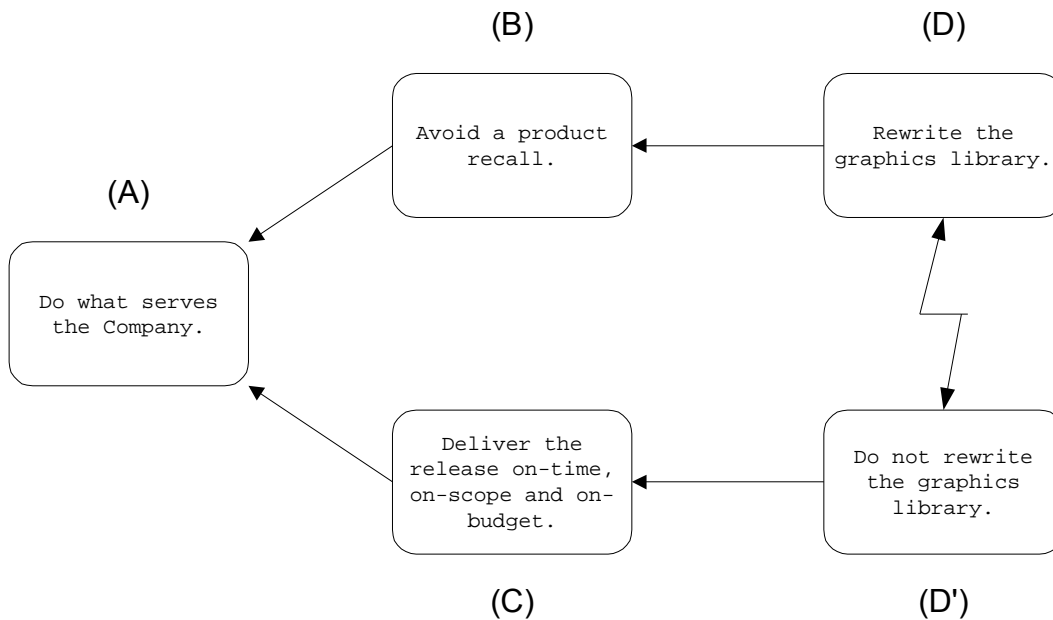
For other dilemmas, however, implementing the actions is difficult. There is nothing surprising about this. We should not allow the fact that doing the right thing is difficult become a justification for doing the wrong thing because it is easy.

Resolving a dilemma that has been diagrammed and clarified is almost a mechanical process. The process is largely one of looking at each arrow (including the conflict arrow) and surfacing (exposing) the assumptions we are making about that arrow. Three to five assumptions per arrow should be surfaced if at all possible.

This will generate quite a few assumptions. These assumptions can be checked to see if any might be challenged. In a list of ten or twenty there will always be at least a few weak assumptions. There may also be some that were valid at one point in time, but which are no longer true.

Weak or invalid assumptions represent an opportunity to break the dilemma – to find a way to resolve it without having to compromise on either need. When this is done the result can be a dramatic “evaporating” of the entire dilemma.

For example consider Mark and David’s graphic library dilemma:



They came up with the following assumptions by asking the question, “In order to avoid a product recall, we must rewrite the graphics library. Why?”

1. Rewriting the library would actually improve it.
2. Our rewrite of the library would not contain major defects not present in the current library.
3. We can’t test the library code well enough to ensure no recallable issues exist.
4. The only way to get the kind of graphics library we need is to write it ourselves.

One possible solution would be to explore the option to outsource writing of the graphics library. Similarly, they could come up with other solutions by challenging their assumptions, at this arrow or another junction.

For instance, here are some assumptions they could surface by asking the question, “In order to serve the company, we must deliver the release on-time, on-scope, and on-budget. Why?”

1. There are never times when not releasing on-time, on-scope, and on-budget, will serve the company.
2. Delivering *this* release will serve the company.

And, so on. Challenging these and others will enable them to create a no-compromises solution to their problem. If they systematically explore their assumptions and find one or more they could challenge successfully, they will evaporate the dilemma.

The example cited is a common outcome when people have been trained to recognize and resolve dilemmas without compromising – and when they are sufficiently motivated to follow the process.

While we sometimes find “silver bullet” solutions to dilemmas it is more common that we find we need two or three ideas to completely resolve a dilemma. While it is satisfying to find a single-idea solution to a dilemma, we should not expect that all dilemmas can be resolved in this way. In the final analysis, what matters is that we avoid compromising on important needs.

It is also important to understand that the ideas for challenging the assumptions are just that – ideas. Ideas have to be implemented in order to actually resolve the dilemma in real life. We must never confuse ideas with solutions.

People often reject the idea that dilemmas can be resolved without compromising important needs. It can be hard to accept. Most of us have learned that there is no free lunch and this approach seems – on the surface – to go against that experience.

The reality is that creating no-compromise solutions does not amount to getting a free lunch:

1. Creating no-compromise solutions is harder than compromising. So we should not be surprised when people advocate compromising.
2. Because the work is mentally challenging and dilemmas are common, there are not enough hours in the day to solve them all. Personal and group energy should be conserved for use on the really important dilemmas.
3. When the ways in which a dilemma could be resolved are identified we still face the challenge of implementing them.

#### **4. More on Resolving Dilemmas**

For many people the most difficult assumptions will be the ones associated with the conflict arrow. But the most powerful solutions to dilemmas are often found by challenging the assumptions associated with this arrow.

For example, let’s assume we are considering a dilemma where the actions are to either “Ship on Friday” or “Not ship on Friday.”

People usually see this as a binary decision – either we ship on Friday or we do not.

But if we explore this in more detail, there are many assumptions behind it:

1. “Ship” means shipping to all customers.
2. We can only ship the product once; we can’t ship it today and then ship another iteration of it at a later date.
3. There is no “slack time” in the shipping process itself that we could exploit without impacting the customer.
4. Shipping on Saturday or Sunday automatically means that the customer receives the shipment one or two days later.
5. We have to physically ship the product – we can’t distribute it electronically.

6. The shipment date is real and not an internal milestone with no real bottom-line impact.

The point here is that even “very clearly binary” decisions always have assumptions associated with them.

When the assumptions associated with each arrow have been exposed the team is asked to identify the assumptions that will need to be overcome in order to resolve the dilemma.

The use of “outrageous” language is of great help in exposing assumptions.

For example, let’s assume that a couple of developers are discussing their belief that they need to rewrite the kernel of the real-time operating system their application uses. One “outrageous” assumption might be:

*Obviously, we have no choice but to rewrite the kernel ourselves because there are no companies that provide kernels that are anywhere near as good as our homegrown kernel.*

This assumption, of course, is just begging to be challenged. The use of outrageous language exploits the very human tendency to want to prove other people wrong.

It is sometimes the case that there is one assumption that, when overcome, completely resolves the dilemma. But it is more often the case that two or three assumptions have to be challenged together to completely resolve the dilemma.

At this point the direction in which to go to resolve the dilemma is clear. In most cases there will be significant work required in order to actually implement the solution. We must be careful not to confuse “ideas” with “solutions.”

Doing the process outlined above is not difficult – if people are willing to invest some time and effort. People do this work if we provide them with the necessary information and training, and then hold them responsible for using the process when it is needed. We discuss this further in the next section.

## 5. Summary

Most of us have been taught from an early age that the right way to deal with serious problems is to seek some acceptable compromise. However, constantly compromising important needs does not create high performance organizations. Rather, high performance organizations are created by refusing to compromise on the important needs of the organization and its people.

The reality is that we can avoid compromising when the needs are important. To do this people need to be able to recognize dilemmatic situations and then to deal with them effectively.

The following conditions must be established if people are to routinely recognize dilemmatic situations and resolve them effectively:

1. Awareness

The first requirement is that people have a basic understanding of what a dilemma is and that dilemmas can be resolved without compromising when important needs are threatened.

When people lack this understanding they go with what they know – which is to always seek some “acceptable” compromise.

2. Willingness to Act

Resolving dilemmas without compromising is more work than simply compromising. As a result there must be an incentive to induce people to do the necessary work to protect the needs inherent in the dilemma they are facing.

If the willingness to act is not established then people will work to conserve their own energy and will continue compromising the needs of the organization.

### 3. Competence

In order to carry out the process of recognizing and resolving dilemmas people need to be competent in the techniques.

Without competence even the best-intended efforts will not yield results.

All of these conditions can be created in organizations.

Education and training provides the necessary awareness and competence.

Providing a willingness to act is also possible. The best course of action is to ensure that people are held accountable for recognizing and resolving dilemmas and that this behavior is rewarded when it occurs. There is no substitute for setting the expectation with the organization that serious dilemmas will be identified and resolved according to the methods outlined here and then helping people meet that expectation.

## 6. Getting Experienced Help

This article is intended to provide you with an understanding of how dilemmas affect all organizations and to provide an explanation of what is going on “under the surface.” It is not intended as a complete course on dealing with dilemmas.

At Common Sense Systems we provide products and services that help individuals and organizations to significantly improve their problem-solving abilities.

Individuals wanting more information should contact:

Common Sense Systems, Inc.  
18915 142<sup>nd</sup> Ave. NE, Suite #145  
Woodinville, Washington 98072

Tel: (425) 806-8744

Fax: (425) 806-8407

Web: <http://www.common-sense.com>