CHAPTER 1

The Changing Landscape of Software Development

We're trying to change the habits of an awful lot of people. That won't happen overnight but it will bloody well happen.

John Akers, CEO
IBM

Chapter Learning Objectives

After reading this chapter, you will be able to:

- **Explain the software development landscape**
- **Know the definition of software development project management strategy**
- **Understand the four quadrants of the software development landscape**
- **Know what project management approach is compatible with each quadrant**
- **Explain the relationship of complexity and uncertainty and the software development project management landscape**
- **Know how risk, team cohesion, communications, customer involvement, change, specification, and business value are affected by the complexity/uncertainty domain**
- **Explain the importance of balancing people, process, and technology in the organization**
- **Explain staff-driven, process-driven, and technology-driven environments**

The software project management landscape is ever-changing. It is defined by no less than five interdependent variables: the characteristics of the software project itself, the software development life cycle, the project management life cycle, the profile of the project team, and the technology that supports the whole. While this may seem overwhelming, it isn't. I'll explore the complexities of this multidimensional landscape with you and show you how to obtain and sustain an effective presence in this changing landscape.

Software development processes and modern project management processes are both about 50 years old. Both are adolescents. Both are trying to earn a seat at the corporate strategy table. Both are sure that they can contribute to the success of their enterprise. Unfortunately, both have a reputation for failing to live up to expectations. Both are struggling, and both face tremendous odds against making any positive impressions.

The equation that says you must strike a balance between people, process, and technology holds the clue as to where you should look. People are smart. Of that there is no doubt. How many times have you heard an executive say, "Just put five of our smart people together in a room, and they will solve any problem you can give them." That may be true, but I don't think anyone would bet the future of their enterprise on the continuing heroic efforts of the anointed few. Technology is racing ahead faster than any organization can absorb, so that can't be the problem. Process is the only thing left, and it is to process that you turn in this book. But it isn't just your normal everyday processes that have your attention. It is the integration of software development processes and project management processes that will demand your attention throughout this book. The result of that integration will be a type of discipline—effective software project management (ESPM). This book is about the concepts and principles of ESPM and its application to real software development problems.

Despite their brief history, software development and project management practitioner groups have never taken the pains to seriously integrate what they have learned with one another. Software developers use their systems development life cycle as a surrogate for project management. Traditional project managers are locked into the construction and engineering mindset that initially defined and continues to define the project management discipline. The impact of the construction and engineering practices on project management continues to be a roadblock to the further development of project management in the software development discipline. As a result, most software developers dismiss most project managers as incapable and irrelevant to meeting their needs. What is needed is to have traditional project managers think openly and creatively about how to effectively serve their customers and deliver business value as their prime directive.

That suggests a fresh approach to managing software development projects. I hope to do that in pages that follow. But right now that that doesn't mean creating new tools, templates, or processes. What we have now is sufficient. What we do not have is the awareness, skills, and creativity to integrate project management life cycles (PMLC) and software development life cycles (SDLC), and the courage to stay the course in implementation of the resulting integrations.

In this book, I take the position that the characteristics of the software development project drive your choice as to the project management tools, templates, and processes that should be used. This is not a recipe book to be blindly followed. Rather, it is a book that teaches you how to create a recipe. In other words, one of my objectives is to help you think like a great project manager.

What Is a Software Development Project?

Several types of software development projects are within the scope of this book. They range from repeatable projects that have been done many times before to projects that are cutting edge problem solving projects. Each presents its own special challenge to the developer. The example given below will be the staging area for exploring effective approaches to software development project management (SDPM).

DEFINITION: SOFTWARE DEVELOPMENT PROJECT

A software development project is a complex undertaking by two or more persons within the boundaries of time, budget, and staff resources that produces new or enhanced computer code that adds significant business value to a new or existing business process.

Although this is a restrictive definition, it does define the types of software development projects that are addressed in this book. The criteria for these projects are that they have the potential of adding significant business value and are not trivial undertakings. These development projects will have significant business value, be highly visible, be of moderate to high complexity, and were needed yesterday.

Examples of Two Software Development Projects

I've crafted a hypothetical case study that will be a referent as I apply the SDPM strategies presented in this book. I hope that this will help you further align yourself with using the models and approaches that this book addresses. I'll incorporate more details to the case study as needed. Any resemblance to past or present companies is strictly coincidental. The case study is purely hypothetical and written to illustrate the use of the concepts and principles in this book.

Introducing the Case Study

Pizza Delivered Quickly (PDQ) is a 40-store local chain of eat-in and home delivery pizza stores. Recently PDQ has lost 30 percent of sales revenue due mostly to a drop in their home delivery business. They attribute this solely to their major competitor who recently promoted a program that guarantees 30-minute delivery service from order entry to home delivery. PDQ advertises one-hour delivery. PDQ currently uses computers for in-store operations and the usual business functions but otherwise is not heavily dependent upon software systems to help them receive, process, and deliver their customers' orders. Pepe Ronee, their Manager of Information Systems, has been charged with developing a software application to identify "pizza factory" locations and create the software system needed to operate them. In commissioning this project, Dee Livery, their president, said to pull out all the stops. She further stated that the future of PDQ depends on this project. She wants the team to investigate an option to deliver the pizza unbaked and "ready for the oven" in 30 minutes or less or deliver it pre-baked in 45 minutes or less.

These pizza factories would not have any retail space. Their only function would be to receive orders, and prepare and deliver the pizzas. The factory location nearest the customer's location will receive the order from a central ordering facility, and process and deliver the order within 30 or 45 minutes of order entry, depending on whether the customer orders their pizza ready for the oven or already baked.

There are two software development projects identified here:

- The first is a software system to find pizza factory locations.
- The second is a software system to support factory operations.

Clearly the first is a very complex application. It will require heavy involvement by a number of PDQ managers. The goal can be clearly defined but even at that the solution will not be at all obvious. The second focuses on routine business functions and should be easily defined. Off-the-shelf commercial software may be a big part of the final solution to support factory operations.

These are obviously very different software development projects requiring very different approaches. The pizza factory location system will be a very sophisticated modeling tool. The requirements, functionality, and features are not at all obvious. Some of the solution can probably be envisioned, but clearly the whole solution is elusive at this early stage. Exactly how it will do modeling is not known at the outset. It will have to be discovered as the development project is underway. The operations system can utilize commercial off the shelf (COTS) order entry software, which will have to be enhanced at the front end to direct the order to the closest factory and provide driving directions for delivery and other fulfillment tasks on the back end. The requirements, functionality, and features of this system may be problematic.

As the case study unfolds in later chapters, you will see that this simple yet realistic case study is rich with learning opportunities. I expect to draw heavily on it for practical illustrations of the concepts and principles presented here.

What Is Software Development Project Management?

Now that you have a clear idea of what a software development project is, it's important to clearly define what software development project management is.

DEFINITION: SDPM

Software development project management is the discipline of assessing the characteristics of the software to be developed, choosing the best fit software development life cycle, and then choosing the appropriate project management approach to ensure meeting the customer needs for delivering business value as effectively and efficiently as possible.

At the risk of cluttering up your vocabulary, I have coined a phrase that reflects the thinking process that I follow to craft a management approach to software development. The definition that follows is unique to this book but important to add to your vocabulary. From now on, any use of the term *SDPM strategy* refers to the definition given here.

DEFINITION: SDPM STRATEGY

A SDPM strategy is an integration of a software development life cycle and a project management life cycle into a customer-facing approach that will produce maximum business value regardless of the obstacles that may arise.

I want you to think of SDPM as an emerging discipline. It is new, although the two components that define it are not new. What is new is the integration of those components to produce an effective SDPM environment. The SDPM strategy for making this happen will be developed in this book.

The title of this section poses a question that is not trivial and certainly not a rhetorical question. I know several project managers that would like to have a working definition of exactly what constitutes software development project management. And further to the point, they would like to know how to do it. This book is a first attempt, but certainly not the last, to answer both questions. My expectations for the effective management of software development projects lie not only in the answer to these questions but also in the answer to three questions that are more operationally focused:

- What are the characteristics of the software to be developed?
- What software development approach is appropriate for building the software?
- What project management approach is appropriate for managing the chosen software development process?

The questions are meant to be answered in the order listed. Each one is dependent on the answers to the previous questions. Furthermore, in execution all three are dependent upon one another. Compare this to the situation where both the software development approach and the project management approach are fixed. Given those two constraints, what do you want us to develop? Do you operate like a solution out looking for a problem? Wouldn't you rather let the characteristics of the problem drive your choices for solution approach? I would hope so. That is the focus of this book.

What Are the Characteristics of the Software to Be Developed?

When I think of the software development landscape, I think of it in very simple terms. I see it as a two-dimensional grid like the one shown in Figure 1-1.

The first dimension relates to the goal of the software development project. The goal is either clearly specified (therefore known) or it is not clearly specified (therefore not known). It's an all or nothing situation. The boundary between clear and not clear is more conceptual that actual. The same is true of the second dimension, which relates to the solution or how you expect to reach the goal. That also has two categories. The solution is either clearly specified (and therefore known) or it is not clearly specified (and therefore not known). If you intersect these two dimensions as shown in the figure, then you have defined a four-category classification of software development projects. This classification is simple but inclusive of every software development project. That is, every software development project that ever has been or ever will be must fall into one and only one of these four categories.

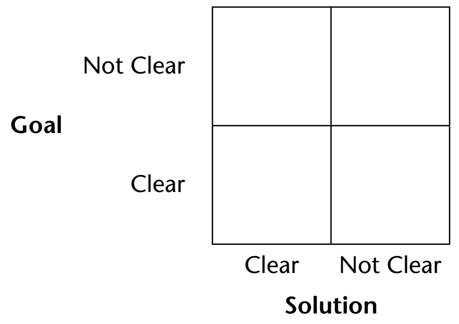


Figure 1-1: The software development landscape

Why is this important? First and foremost, the characteristics of the software to be developed will play an important role in determining the model that will be used. Each of these quadrants presents the development team with a number of decisions regarding how to go forward. The next sections briefly examine each quadrant and the salient aspects of clarity or lack thereof with respect to goal and solution.

Quadrant 1: Goal and Solution Are Clearly Specified

How could it be any better than to clearly know the goal and the solution? This is the best of all possible worlds, but it is also the least likely to occur in today's fast-paced, continuously changing business world. Software development projects that fall into this quadrant are familiar to the organization. Perhaps similar projects have been done several times before. There are no surprises. The client has clearly specified the goal and how to reach that goal. Little change is expected. A variety of approaches is in use for such software development projects. They are all of the design-build-test-implement variety or some variation of the linear concept implied by these approaches. Such projects also put the team on familiar technology grounds. The hardware, software, and telecommunications environments are familiar to the team. They have used them repeatedly and have developed a skilled and competent developer bench to handle such projects.

The limiting factors in these plan-driven approaches are that they are change-intolerant, are focused on delivering according to time and budget constraints, and rely more on compliance to plan than on delivering business value. The plan is sacred, and conformance to it is the hallmark of a successful project team.

Because of the times we live in, these approaches are rapidly becoming dinosaurs. At least the frequency of their application is diminishing rapidly. They are giving way to a whole new collection of approaches that are more customer-focused and deliver business value rather than adhere to a schedule and budget plan.

In addition to a clearly defined goal and solution, software development projects that correctly fall into this quadrant have several identifying characteristics as briefly identified here.

Low Complexity

Other than the fact that the project really is simple, this will often be attributable to the fact that the software development project rings of familiarity. It might be a straightforward application of established business rules and therefore take advantage of existing designs and coding. To the developer it might look like a cut-and-paste exercise. In such cases integration and testing will be the most challenging phases of the development project.

You can still find situations where the project is complex but still well-defined. However, these are rare.

Well-Understood Technology Infrastructure

A well-understood technology infrastructure is one that is stable and has been the foundation for many software development projects in the past. That means that the accompanying skills and competencies to work with the technology infrastructure are well-grounded in the development teams.

Low Risk

The total environment for development projects in this quadrant is that it is known. All that could happen to put the project at risk has occurred in the past, and you have well-tested and well-used mitigation strategies in place. Experience has rooted out all of the mistakes that could be made. The customer is confident that it has done a great job identifying requirements, functions, and features, and they are not likely to change. Except for acts of nature and other unavoidable events, the project is protected from avoidable events. You find few unanticipated risks in software development projects in this quadrant.

Experienced and Skilled Developer Teams

Past projects have been good training grounds for the teams. They have had opportunities to learn or to enhance their skills and competencies.

NOTE

I'll have much more to say about teams in the chapters that discuss the Launch Phase of each SDPM strategy. They are a critical success factor in all software development projects. As the characteristics of the software to be developed changes, so also does the profile of the team that can be most effective in developing that software.

Quadrant 2: Goal Is Clearly Specified but Solution Is Not

You have a host of incremental, iterative, and adaptive approaches to SDPM that can be used when the goal is clearly defined, but how to reach the goal—the solution—is not. As you give some thought to where your projects would fall in this landscape, consider the possibility that many if not most of them are really these types of projects. If that is the case, shouldn't you also be considering using an approach to managing these projects that accommodates the goal and solution characteristics of the project rather than trying to force fit some other approach that was designed for projects with much different characteristics?

I contend that the adaptive and iterative class of projects is continuously growing. I make it a practice at all “rubber chicken” dinner presentations to ask about the frequency with which the attendees encounter Quadrant 2 projects. With very small variance they say that at least 75 percent of all their projects are Quadrant 2 projects. Many of them try to adapt Quadrant 1 solutions to Quadrant 2 projects and meet with very little success. The results have ranged from mediocre success to outright failure. Quadrant 2 projects present a different challenge and need a different approach. For years I have advocated that the approach to the project must be driven by the characteristics of the project. To reverse the order is to court disaster. With the addition of the Quadrant 2 approaches discussed in Parts IV, V, and VI of this book, I cover the project landscape with a full complement of approaches for every conceivable type of project.

Quadrant 3: Goal and Solution Are Not Clearly Specified

Quadrant 3 extends to the remotest boundaries of project types. Quadrant 3 projects are those projects whose goal and solution cannot be clearly defined. What little planning is done just in time, and the project proceeds through several iterations until it converges on an acceptable goal and solution. If instead there isn’t any prospect of convergence, the customer might pull the plug and cancel the project at any time and look for alternative approaches.

Quadrant 4: Goal Is Not Clearly Specified but the Solution Is

The fourth category represents projects whose goals are not known but whose solutions are. This is an impossible situation. It would be equivalent to solutions out looking for problems. Nevertheless, we all have had experiences working with professional services organizations that practice such approaches. They advocate a one-size-fits-all approach, which has never shown to be very successful. I have always discouraged a one-size-fits-all approach with my clients. Most see the wisdom in adopting this position.

What Software Development Approach Is Appropriate for Building the Software?

The characteristics of the software development project will play an important role in determining the software development model to be used. Here I give a generic description of the model characteristics. In Chapter 2 I peel back the onion to the next level of detail and present the five classes of software development approaches. That sets the stage for a detailed discussion of the software development approaches in each of the five classes, which is the topic of Part II through Part VI of the book.

Quadrant 1: Goal and Solution Are Clearly Specified

Because all of the information that could be known about this development project is known and is considered stable, the appropriate development model is the one that gets to the end as quickly as possible. Based on the requirements, desired functionality, and specific features, a complete project plan can be developed. It specifies all of the work needed to meet the requirements, the schedule of that work, and the staff resources needed to deliver to the planned work. Quadrant 1 projects are clearly plan-driven projects. Their success is measured by compliance and delivery to that plan.

Quadrant 2: Goal Is Clearly Specified but Solution Is Not

As the solution moves from one that is clearly specified toward one that is not clearly specified, you move through a number of situations that require different handling. For example, suppose only some minor aspects of the solution are not known—features, perhaps; how would you proceed? An approach that includes as much of the solution as is known at the time should work quite well. That approach would allow the customer to examine, in the sense of a production prototype, what is in the solution in an attempt to discover what is not in the solution but should be. At the extreme, when very little is known about the solution, development projects are higher risk than those where a larger part of the solution is known. A solution is needed, and it is important that a solution be found. How would you proceed? What is needed is an approach that is designed to learn and discover the solution. Somehow that approach must start with what is known and reach out to what is not known. The anchor to this approach is that the goal is clearly specified.

Quadrant 3: Goal and Solution Are Not Clearly Specified

If goal clarity is not possible at the beginning of the project, the situation is much like a pure research and development project. Now how would you proceed? In this case you use an approach that clarifies the goal and contributes to the solution at the same time. The approach must embrace a number of concurrent probes that accomplish both. The concurrent probes might be the most likely ones that can accomplish goal clarification and the solution set at the same time. Depending on time, budget, and staff resources, these probes might be pursued sequentially or concurrently. Alternatively, the probes might eliminate and narrow the domain of feasible goal/solution pairs. Clearly Quadrant 3 projects are an entirely different class of projects and require a different approach to be successful.

Quadrant 4: Goal Is Not Clearly Specified but the Solution Is

Here is that nonsense quadrant again. You have the solution; now all you need is to find the problem. This is the stuff that academic articles are often made of. Post your solution and hope somebody responds with a problem that fits it. It has happened. Take the 3M Post-it Note saga, for example. The product sat on the shelf for several years before someone stumbled onto an application. The rest is history.

In summary, you have to answer the first of the three questions posed earlier: "What are the characteristics of the software to be developed?" Because the landscape has been defined in terms of four categories, it should be easy to identify the quadrant that the development project belongs to. If there is any doubt about the quadrant, err on the side of choosing a higher numbered quadrant. I'll have more to say on that strategy throughout the book.

What Project Management Approach Is Appropriate for Managing the Software Development Process?

Now that the first question has been answered and you know what quadrant the project lies in, you can answer the second question, "What project management approach is appropriate for managing software development projects in this quadrant?" As you move through the quadrants from clarity to lack of clarity, the project management processes you use must track with the needs of the project. As a general word of advice as you move through the quadrants, remember that "Lots is bad, less is better, and least is best." In other words, don't burden the project manager and team with needless planning and documentation that will just hinder their efforts. As my colleague Jim Highsmith has said to me conversationally: "The idea of enough structure, but not too much, drives agile managers to continually ask the question, 'How little structure can I get away with?' Too much structure stifles creativity. Too little structure breeds inefficiency." Quadrant 1 projects are plan-driven, process-heavy, and documentation-heavy. As you move to Quadrants 2 and 3 projects, heaviness gives way to lightness. Plan-driven gives way to value-driven, rigid process gives way to adaptive process, and documentation is largely replaced by tacit knowledge that is shared among the team members. These are some of the characteristics of the many approaches that fall in the *agile project management* taxonomy. Several approaches fall under the umbrella of agile. Each is discussed in detail in Part II through Part VI.

I've always felt that the project manager must see value in a project management process before she is willing to use it. Burdening the project manager with what they perceive as a lot of non-value-added work is counterproductive and to be avoided. This becomes more significant as you move from Quadrant 1 to 2 to 3. Furthermore, project managers will resist, and you will get a token effort at compliance. My overall philosophy is that the less non-value-added time and work that you encumber your project managers with the better off you will be. Replacing non-value-added work with value-added work increases the likelihood of project success. Time is a precious (and scarce) resource for every software development project. You need to resist the temptation to add work that doesn't directly contribute to the final deliverables. Up to a point the project manager should determine what is a value add to their project processes and documentation. Make it their responsibility to decide what to use and when to use it. This is the mark of a successful manager of project managers—that they make it possible for the project manager to be successful and then stay out of their way.

Project management methodologies include a number of tools, templates, and processes and the rules for their use. The process of integrating those tools, templates, and processes into software development processes is actually quite straightforward to define. It isn't quite that simple as far as implementation is concerned and that is what motivated me to write this book. In this book, you will learn how to do that integration effectively and how to deal with the various demons that raise their heads during that implementation.

The Complexity/Uncertainty Domain of SDPM

Each quadrant of the software development project landscape has different profiles when it comes to risk, team, communications, customer involvement, specification, change, business value, and documentation. In this section, you examine the changing profile of each domain as you move from quadrant to quadrant.

Complexity and uncertainty are positively correlated with one another. As software development projects become more complex, they become more uncertain. That follows from at least four other relationships, as commented on in the next four sections.

In the Quadrant 1 models you know where you are going, and you know precisely how you are going to get there. It's all in the requirements, functionality, and features. Your plan reflects all of the work, the schedule, and the resources that will get you there. No complexity here. As soon as you move away from a clearly specified solution and are in Quadrant 2, the world is no longer as kind to you as it was while you were in Quadrant 1. The minute you have uncertainty anywhere in the project complexity goes up. You have to devise a plan to fill in the missing pieces. There will be some added risk—you might not find the missing piece, or when you do, you find that it doesn't fit in with what you already have built—go back two steps, undo some previous work, and do the required rework. The plan changes. The schedule changes. A lot of the effort spent earlier on developing a detailed plan has gone to waste. By circumstance it has become non-value-added work. If you had only known.

As less and less of the solution is known, the realities of non-value-added work become more and more a factor. Time has been wasted. Quadrant 2 models are better equipped to handle this uncertainty and the complexity that results from it. The models are built on the assumption that the solution has to be discovered. Planning becomes less of a one-time task done at the outset to a just-in-time task done as late as possible. You have less and less reliance on a plan and more reliance on the tacit knowledge of the team. That doesn't reduce the complexity, but it does accommodate it. So even though complexity increases as you move from Quadrant 1 to 2 to 3, you have a way to deal with it for the betterment of your customer and your sanity as a project manager.

Requirements

As project complexity increases, the likelihood of nailing requirements decreases. This follows logically from the fact that the human brain can retain in memory only about seven pieces of information. The dimensions of complexity are likely to far exceed that constraint. In a complex software product the extent of the number of requirements, functionality, and features can be staggering. Some will conflict with each other. Some will be redundant. Some will be missing. Many of these might not become obvious until well into the design, development, and even integration-testing tasks.

Flexibility

As project complexity increases, so does the need for process flexibility. Increased complexity brings with it the need to be creative and adaptive. Neither is comfortable in the company of rigid processes. Quadrant 2 projects are easily compromised by being deluged with process, procedure, documentation, and meetings. Many of these are unrelated to a results-driven approach. They are the relics of plan-driven approaches. Along with the need for increased flexibility in Quadrant 2 and 3 projects is the need for increased adaptability. Companies that are undergoing a change of approach that recognizes the need to support not just Quadrant 1 projects but also Quadrant 2 projects are faced with a significant and different cultural and business change. For one, the business rules and rules of the project engagement will radically change. Expect resistance.

Flexibility here refers to the project management process. If you are using a one-size-fits-all approach, you have no flexibility. The process is the process is the process. Not a very comforting situation if the process gets in the way of commonsense behaviors and compromises your ability to deliver value to your customer. Wouldn't you rather be following a strategy that allows you to adapt to the changing situations?

Quadrant 1 development projects generally follow a traditional project management methodology. The plan is developed along with a schedule of deliverables and other milestone events. A formal change management process is part of the game plan. Progress against the planned schedule is tracked, and corrective actions are put in place to restore control over schedule and budget. A nice neat package isn't it? All is well until the process gets in the way of product development. For example, if the business situation and priorities change and result in a flurry of scope change requests to accommodate the new business climate, an inordinate amount of time is then be spent processing change requests at the expense of value-added work. The schedule slips beyond the point of recovery. The project plan, having changed several times, becomes a contrived mess. Whatever integrity there was in the initial plan and schedule is now lost among the changes.

Quadrant 2 is altogether different. Project management is really nothing more than organized commonsense. So when the process you are using gets in the way, you adapt. The process is changed to maintain focus on doing what makes sense to protect the creation of business value. Unlike Quadrant 1 processes, Quadrant 2 processes expect and embrace change as a way to a better solution and as a way to maximize business value within time and budget constraints. That means choosing and continually changing the SDPM strategy to increase the business value that will result from the project. Realize that to some extent scope is a variable in these types of SDPM strategies.

Quadrant 3 projects are even more dependent upon flexible approaches. Learning and discovery takes place throughout the project, and the team and customer must adjust how they are approaching the project on a moment's notice.

Adaptability

The less certain you are of project requirements, functionality, and features, the more need you will have to be adaptable with respect to process and procedure. Adaptability is directly related to the extent to which the team members are empowered to act. The ability of the team to adapt increases as empowerment becomes more pervasive. Remember to make it possible for the team members to be productive and stay out of their way. Don't encumber the team members with the need to get sign-offs that have nothing to do with delivering business value. Pick them carefully and trust them to act in the best interest if the customer.

Change

As complexity increases so does the frequency and need to receive and process change requests. A plan-driven software development project is not designed to effectively respond to change. Change upsets the order of things as some or all of the project plan is affected. Resource schedules are compromised. The more that change has to be dealt with, the more time is spent processing and evaluating the changes. That time is lost to the project. It should have been spent on value-added work. Instead it was spent processing change requests.

You spend so much time developing your project plan for your Quadrant 1 project that the last thing you want is to have to change it. But that is the reality in Quadrant 1 projects. Scope change always seems to add more work. Did you ever receive a scope change request from your customer that asked you to take something out? Not too likely. The reality is that the customer discovers something else they should have asked for in the solution. They didn't realize that or know that at the time. That leads to more work, not less. The call to action is clear—choose Quadrant 1 models when specifications are as stable as can be. The architects of the Quadrant 2 and 3 models knew this and so designed approaches that expected change and were ready to accommodate it. You'll see that in more detail in Parts 2 through 6 of the book.

Risk Versus the Complexity/Uncertainty Domain

Risk increases as you move from Quadrant 1 to 2 to 3. In Quadrant 1 you clearly know the goal and the solution and can build a definitive plan for getting there. The exposure to risks associated with product failure is low. The focus can then shift to process failure. A list of candidate risk drivers would have been compiled over past similar projects. Their likelihood, impact, and the appropriate mitigations is known and documented. Like a good athlete, you have anticipated what might happen and know how to act if it does.

As the software development project takes on the characteristics of Quadrant 2, two forces come into play. First, the SDPM strategy becomes more flexible and lighter. The process burden lessens as more attention is placed on delivering business value than on conformance to a plan. At the same time, the product risk increases, as illustrated in Figure 1-2. Risk increases in relation to the extent to which the solution is not known. On balance that means more effort should be placed on risk management as the software development project moves through Quadrant 2 and looks more like a Quadrant 3 project. You will have less experience with these risks because they are specific to the product being developed. In Quadrant 3, risk is the highest because you are in a research and development environment. Process risk is almost nonexistent because the ultimate in flexibility has been reached in this quadrant, but product risk is extremely high. You will have numerous product failures because of the highly speculative nature of Quadrant 3 projects, but that is okay. Those failures are expected to occur. Each product failure gets you that much closer to a functional solution, if such solution can be found within the operative time and budget constraints. At worst those failures eliminate one or more paths of investigation and so narrow the range of possible solutions.

Team Cohesiveness Versus the Complexity/ Uncertainty Domain

In Quadrant 1 the successful team doesn't really have to be a team at all. You assemble a group of specialists and assign each to their respective tasks at the appropriate times. Period. The plan is sacred, and the plan guides them through their task. It tells them what they need to do, when they need to do it, and how they know they have finished their task. They are a group of specialists. They each know their discipline and are brought to the team to apply their discipline to a set of specific tasks. When they have met their obligation, they often leave the team to return later if needed. Period.

The situation quickly changes as the project is a Quadrant 2 or 3 project. First of all, you have a gradual shift of the team makeup from a team of specialists to a team of generalists. The team takes on more of the characteristics of a self-directed team. They become self-sufficient and self-directing as the project moves from a Quadrant 2 to a Quadrant 3 project. Quadrant 1 teams are not co-located. They don't have to be. Quadrant 2 and 3 teams are co-located. Research has shown that co-location adds significantly to the successful completion of the project. Figure 1-3 reflects this shift from a loosely formed team to one that is tightly coupled.

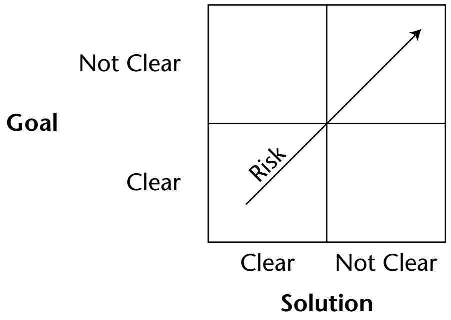


Figure 1-2: The Risk domain

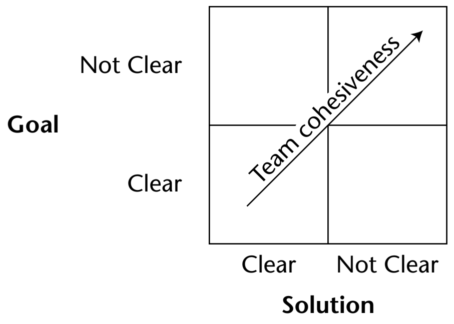


Figure 1-3: The Team Cohesiveness domain

Communications Versus the Complexity/Uncertainty Domain

Lack of timely and clear people-to-people communications has been shown to be the single most frequent reason for project failure. I include both written and verbal communications media in making that statement. Figure 1-4 reflects my thinking.

As you move in the direction of increased complexity and heightened uncertainty, communication requirements increase and change. When complexity and uncertainty are low, the predominant form of communications is written. Status reports, change requests, meeting minutes, issues reporting, problem resolution, project plan updates, and other written reports are commonplace. As uncertainty and complexity increase, written communications give way to verbal communication. The burden of plan-driven approaches is lightened, and the communications requirements of value-driven approaches take over.

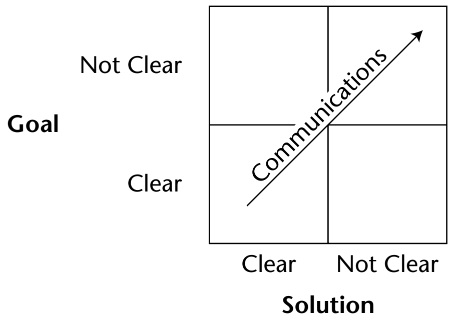


Figure 1-4: The Communications domain

Value-driven communications approaches are the derivatives of meaningful customer involvement where discussions generate status updates and plans going forward. Because projects that are high in complexity and uncertainty depend on frequent change, they have a low tolerance of written communications. In these project situations, the preparation, distribution, reading, and responding to written communications is viewed as non-value-added work. It is to be avoided and the energy spent on value-added work.

Customer Involvement Versus the Complexity/ Uncertainty Domain

Consider for a moment a project from your experience where you were most certain of the goal and the solution. You would be willing to bet your first-born that you had nailed requirements and that they would not change. Yes, that type of project might just be a pipe dream, but give me the benefit of the doubt. For such a project you might ask: Why do I need to have my customer involved except for the ceremonial sign-offs at milestone events? A fair question and ideally you wouldn't need their involvement. How about a project at the other extreme where the goal is very illusive and no solution would seem to be in sight? In such cases the complete involvement of the customer, as a team member perhaps, would be indispensable. What I have painted here are the extreme cases in Quadrant 1 and Quadrant 3.

Quadrant 1 projects are team-driven projects. Customer involvement is usually limited to answering clarification questions as they arise and giving sign-offs and approvals at the appropriate stages of the project life cycle. It would be accurate to say that customer involvement in Quadrant 1 projects is reactive and passive. But all that changes as you move into Quadrant 2 projects. The customer must now take a more active role in Quadrant 2 projects than was their role in Quadrant 1 projects. For Quadrant 3 projects, meaningful customer involvement is essential. In fact, the customer should take on a proactive role. The project goes nowhere without that level of commitment from the customer. Figure 1-5 reflects the gradual shift from passive to very active across the project domain.

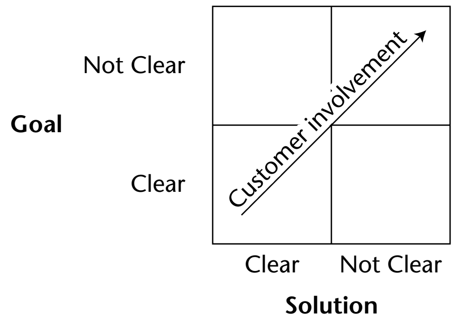


Figure 1-5: The Customer Involvement domain

Finding the solution to a software development project is not an individual effort. In Quadrant 1, the project team under the leadership of the project manager is charged with finding the missing parts of the solution. In some cases the customer is passively involved, but for the most part the team solves the problem. The willingness of the customer to even get passively involved depends on how you have dealt with them so far in the project. If you bothered to include them in the planning of the project, they might have some sympathy and help you out. But don't count on it. Beginning with Quadrant 2 and extending through Quadrant 3, you find more and more reliance on meaningful customer involvement. In your effort to maintain a customer-focus and deliver business value, you are dealing with a business problem not a technology problem. You have to find a business solution. Who is better equipped to help than the customer? After all, you are dealing with their part of the business. Shouldn't they be the best source of help and partnership in finding the solution? This involvement is so critical that without it you have no chance of being successful with Quadrant 3 projects.

Meaningful customer involvement can be a daunting task for at least the three reasons cited in the subsections that follow.

The Customer's Comfort Zone

The customer has been trained ever since the 1950s to take up a passive role. That training went well, and now you have to retrain them. In many instances their role was more ceremonial than formal. They didn't understand what they were approving but had no recourse but to sign. The sign-off at milestone events was often a formality because the customer didn't understand the techie-talk, was afraid not to sign off because of the threat of further delays, and didn't know enough about development to know when to ask questions and when to push back. Now you are asking them to step into a new role and become meaningfully engaged in the software development life cycle. Many are not poised to take up that responsibility. That responsibility is ratcheted up a notch as the project moves further into Quadrant 2 toward Quadrant 3, with less and less known about the solution. The project team is faced with a critical success factor of gaining meaningful customer involvement throughout the SDLC and PMLC. In Quadrant 3 their involvement is even more proactive and engaging. Quadrant 3 projects require that the customer take a co-leadership role with the project manager to keep the project moving forward and adjusted in the direction of increasing business value.

At the same time, the customer's comfort zone is growing. He or she has become smarter. It is not unusual to find a customer who was once more technically involved. They go to conferences where presentations often include technical aspects. They know how to push back. They know what it takes to build software solutions. They've built some themselves using spreadsheet packages and other applications tools. That has two sides. They can be supportive, or they can be obstacles to progress.

Ownership by the Customer

Establishing ownership by the customer of the project product and process is critical. I often ensure that there is that ownership by organizing the project team around co-managers—one from the provider side and one from the customer side. These two individuals are equally responsible for the success of the project. That places a vested interest squarely on the shoulders of the customer manager. This sounds really good, but it is not easily done. I can hear my customers saying, "This is a technology project, and I don't know anything about technology. How can I act in a managerial capacity?" The answer is simple, and it goes something like this: "True, you don't have a grasp of the technology involved, but that is a minor point. Your real value to this endeavor is to keep the business focus constantly in front of the team. You can bring that dimension to the team far better than any one of the technical people on the team. You will be an indispensable partner in every decision situation faced in this project." This ownership is so important that I have postponed starting customer engagements because the customer can't send a spokesperson to the planning meeting. When they do, you have to be careful that they don't send you a weak representative who wasn't busy at the time or who doesn't really understand the business context of the project. Maybe there's a reason that person wasn't busy.

Customer Sign-Off

This is often the most anxiety-filled task that you ever ask of your customer. Some customers think that they are signing their lives away when they approve a document or a deliverable. You are going to have to dispel that perception. This world is one of constant change, high-speed, and high risk. Given that, how could anyone reasonably expect that what works today will work tomorrow? Today's needs might not even come up on the radar screen next week. No matter how certain you are that you have nailed the requirements, you wouldn't expect them to remain static for the length of the project. It simply won't happen. That means that you had better anticipate change as a way of life in SDPM.

Specification Versus the Complexity/Uncertainty Domain

What does this mean? Simply put, it advises you that the choice of SDPM strategy should be based on an understanding of the confidence you have that the specifications have been completely and clearly defined and documented and that scope change requests will not arise from any shortcomings in the specifications documents. As that specification certainty diminishes, your best choices lie in the iterative strategies that populate Quadrant 2—those that allow the solution to become more specific and complete as the project commences or that allow you to discover the solution as the project commences. Finally, if you have very little confidence that you have clearly and completely documented the specifications, then your SDPM strategy takes on the flavor of the research and development strategies that populate Quadrant 3. Figure 1-6 reflects this shift in understanding about specification clarity and completeness.

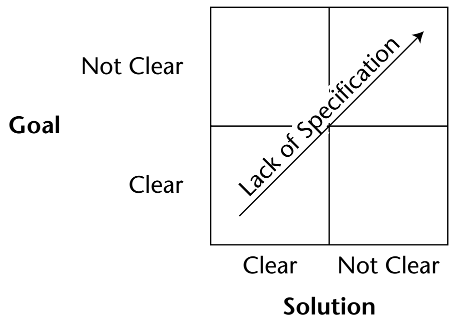


Figure 1-6: The Lack of Specificity domain

The SDPM strategies that require a high level of specification certainty tend to be change intolerant. Consider the situation where a significant change request comes early in the project life cycle. That could render much of the planning work obsolete. A large part of it will have to be done over. That contributes to the non-value-added work time of the SDPM strategy you have chosen. If changes like that are to be expected, an SDPM strategy that is more tolerant and supportive of change should be chosen. The non-value-added work could have been greatly diminished or removed altogether.

If you look inside the specifications document, you can find more detailed information that might help you decide on the best software development model. Specifications are composed of requirements, functions, and features. These array themselves in a hierarchical structure much like that shown in Figure 1-7.

Uncertainty at the requirements level has more impact on choice of software development approach than does uncertainty at the functionality level, which has more impact than that at the features level. Despite all of these efforts, you still have changes on any of those fronts that could have significant impact on our best efforts. That's life.

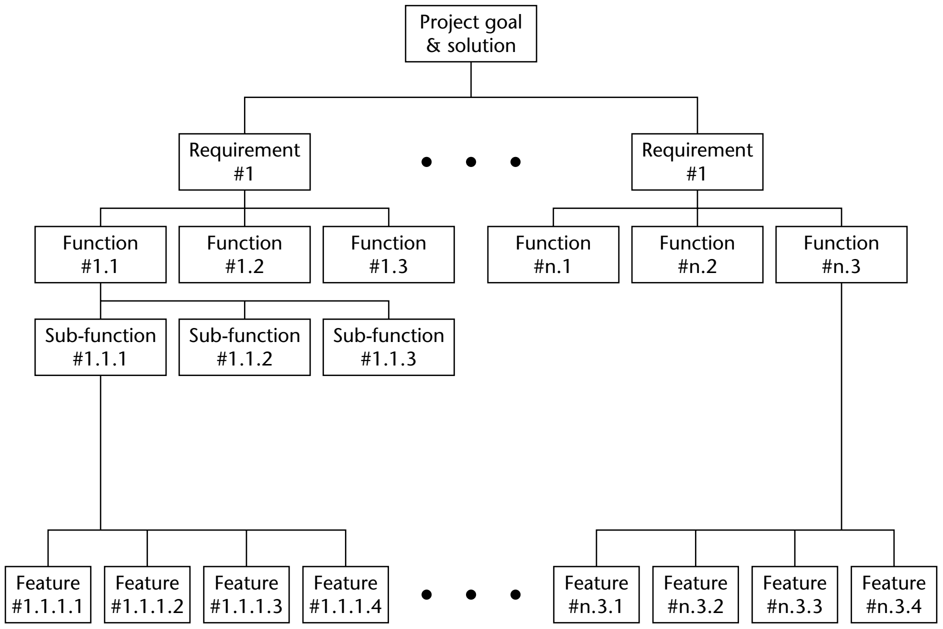


Figure 1-7: The requirements, functionality, and features breakdown structure

Change Versus the Complexity/Uncertainty Domain

The less you know about requirements, functionality, and features, the more you have to expect change. In Quadrant 1 you know everything there is to know about requirements, functionality, and features for this development project. The assumption, then, is that there will be little or no internal forces for change during the development project. Externally, however, that is not the case. Actions of competitors, market forces, and technological advances can cause change, but that is present in every project and can only be expected. The best the enterprise can do is maintain a position of flexibility in the face of such unpredictable but certain events. Figure 1-8 reflects the frequency of change as projects move across the landscape.

Quadrant 2 is a different story altogether. Any change in this quadrant comes about through the normal learning process that takes place in any software development project. When the customer has the opportunity to examine and experiment with a partial solution, he or she will invariably come back to the developers with suggestions for other requirements, functionality, and features that should be part of the solution. These suggestions can be put into one of two categories: either they are "wants" or they are "needs."

"Wants" might be little more than the result of a steak appetite on a baloney budget. It is up to the project manager to help the customer defend their want as a true need and hence get it integrated into the then solution. If they fail to do that, their suggestion should be relegated to a wish list. Wish lists are seldom revisited. If, on the other hand, they demonstrate its value and hence transfer it to a true need, it is up to the project manager to accommodate that new requirement, functionality, or feature into the solution set. It might have to be prioritized in the list of all needs.

In Quadrant 3 you have a further reliance on change to affect a good business-valued product. In fact, Quadrant 3 projects require change in order to have any chance at finding a successful solution. Change is the only vehicle that will lead to a solution.

Business Value Versus the Complexity/Uncertainty Domain

This domain would seem to be trivial. After all, aren't all projects designed to deliver business value. These projects were commissioned based on the business value they would return to the enterprise. This is all true. However, traditional project approaches focus on meeting the plan-driven parameters: time, cost, scope. When originally proposed the business climate was such that the proposed solution was the best that could be had. In a static world that condition would hold. Unfortunately, the business world is not static, and the needs of the customer aren't either. Bottom line, what will deliver business value is a moving target. Quadrant 1 development projects aren't equipped with the right stuff to deliver business value.

It follows then that Quadrant 1 projects deliver the least business value and that business value increases as you move from Quadrant 1 to Quadrant 2 to Quadrant 3. Figure 1-9 illustrates that point quite clearly. At the same time, however, as you move from quadrant to quadrant, risk increases and that means that higher-valued projects need to be commissioned as you move across the quadrants. Remember that the expected business value of a project is the product of (1-risk) and value. Risk here is expressed as the probability of failure and the probability of success is therefore (1-risk).

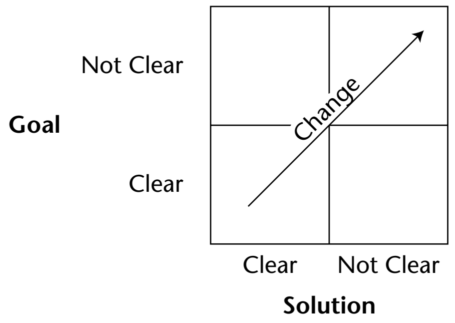


Figure 1-8: The Change domain

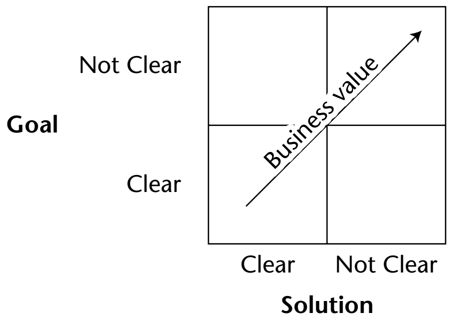


Figure 1-9: The Business Value domain

What does this mean? Simple—whatever SDPM strategy you adopt for the project, it must be one that allows redirection as business conditions change. The more uncertainty present in the development project, the more you need to be able to redirect to take advantage of changing conditions and opportunities.

As projects move through Quadrants 1 to 2 to 3, they become more customer-facing. The focus changes from conformance to plan to delivery of business value. The Quadrant 1 models focus on conformance to plan. If they also happen to deliver maximum business value it would be more the result of an accident than the result of a clairvoyant project plan. The focus on delivery of business value is apparent in all of the Quadrant 2 and 3 models. It is designed into the models.

Balancing Staff, Process, Technology

In this book, I adopt the model shown in Figure 1-10. Staff, or rather the skill and competency profile of the project team, drives the choice software development process and project management process to be employed, and together staff and process drive the choice of technology infrastructure to be employed. This is critical to forming the environment in which the project work will be undertaken. These three factors together form the SDPM strategy.

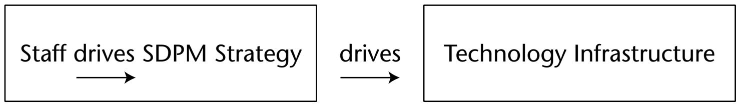


Figure 1-10: Achieving balance in the SDPM environment

This balance is achieved by first assessing the available staff resources as compared to the skill and competencies needed for the project. The chosen project team then determines the SDPM strategy that best meets the needs of the project and aligns with the team's capacity to deliver. Knowing that, the team can now select the technology infrastructure that best supports the team's capacity to deliver using the chosen SDPM strategy. That technology infrastructure includes software choices (programming languages and other support software, hardware, data communications hardware, and so on).

The balance achieved by these choices is represented by the triangle shown in Figure 1-11. It shows the three coordinates (staff, process, and technology). Those coordinates are constrained to the inside of the triangle because there is a linear constraint on the metric that measures each coordinate. In this example the sum total of the assessed values of each coordinate is 200.

The notation requires some explanation. The three letters (S, P, and T) denote the following: S is for staff, P denotes SDPM strategy, and T denotes technology. The ordering of the three letters is meaningful. The proximity of each vertex to the data point determines the ordering. For example, in Figure 1-11 the current state is closest to the Process vertex, next closest to the Technology vertex, and furthest from the Staff vertex. That results in the labeling PTS. All of the data points that have that property fall in the zone labeled PTS.

By the time this book is published, a beta version of the assessment tool will be available. The assessment tool consists of twenty questions, each with three possible answers. A question is answered by distributing ten points across the three possible responses with the highest point value given to the response that most represents the situation being assessed. The questions are asked twice—once for describing the current state of the environment and once for describing the ideal state of the environment. Figure 1-11 displays a possible result.

Knowing the current state and the ideal, or desired, end state, you can develop a plan that will migrate the organization to its desired end state. Figure 1-12 summarizes what some of those migration strategies might look like

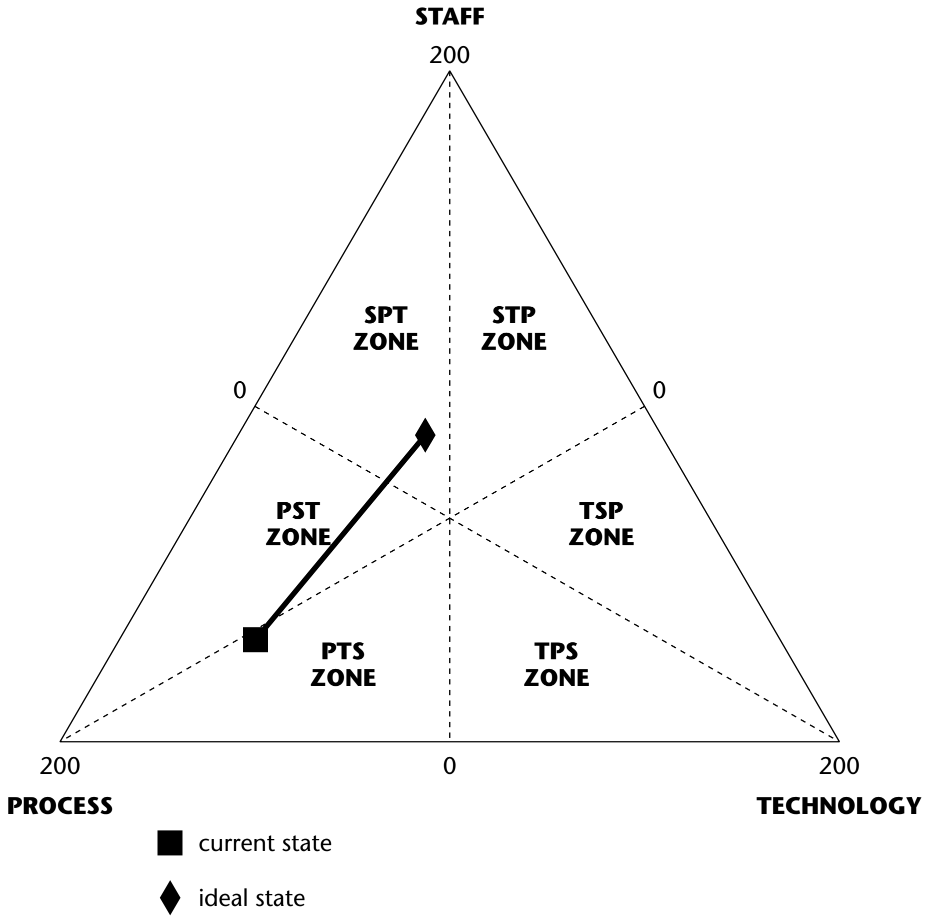


Figure 1-11: The current versus ideal SDPM environment

Note that the migrations are between neighboring zones only. For example, if the current state is zone TPS and the desired zone is SPT, then the migrations would be from zone TPS to TSP to STP to SPT.

Please contact me for further information on how you can learn more about the beta version. My e-mail address is `rkw@eiicorp.com`.

Now you'll take a look at the various driver sequences and what happens when this balance is compromised.

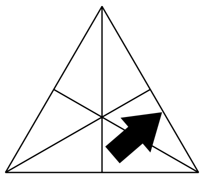
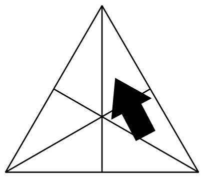
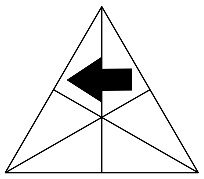
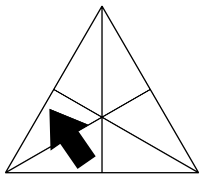
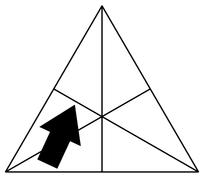
From Zone	To Zone	Comments on the Transition	
TPS	TSP		In this situation technology may have constrained the formation of project management processes because there was little or no input from staff. To correct this situation staff could be empowered to improve project management processes. This may result in some reversals or prior technology decisions.
TSP	STP		Even though technology might be a constraint, staff has had an opportunity to define project management processes. Staff needs to be empowered to make decisions regarding the appropriate technology as it relates to project management processes.
STP	SPT		In this situation the staff are in a enviable position. The remaining task is to create a more balanced relationship between process and technology. That will require slow changes so that the technology environment is adjusted to provide better support for project management processes.
PTS	PST		There may be good coherence between project management processes and the technology to support it but it would have happened without much priority given to the role of staff. That situation can begin to change by commissioning the staff to work on technology improvement initiatives. This may result in reversing prior decisions.
PST	SPT		This is a strong starting position. Project management processes are a high priority for the organization. Technology has been implemented to support both process and staff. The remaining step is to move staff into a higher priority position for further enhancement of the project management environment and the technical support of it.

Figure 1-12: Migration strategies

Staff-Driven Environments

Figure 1-13 illustrates the two people-driven environments that might be encountered.

In the first case (denoted by the star) staff drives SDPM strategy, and together staff and SDPM strategy drive technology. This should be the ideal state for all organizations. It is clear that by using this model you are leveraging the skill and competency capacity of your team and the characteristics of the project to decide how to approach the project from an SDPM perspective. The team should make that decision. The technology platform that they choose to use will take advantage of and build on the earlier decisions on SDPM strategy.

In the second case (denoted by the circle) staff drives technology, and together staff and technology drive process. The only problem with this model is that the choice of SDPM strategy will be constrained by the earlier decision on technology infrastructure. If the earlier decision on technology infrastructure is reversible, then the second case really morphs into the first case.

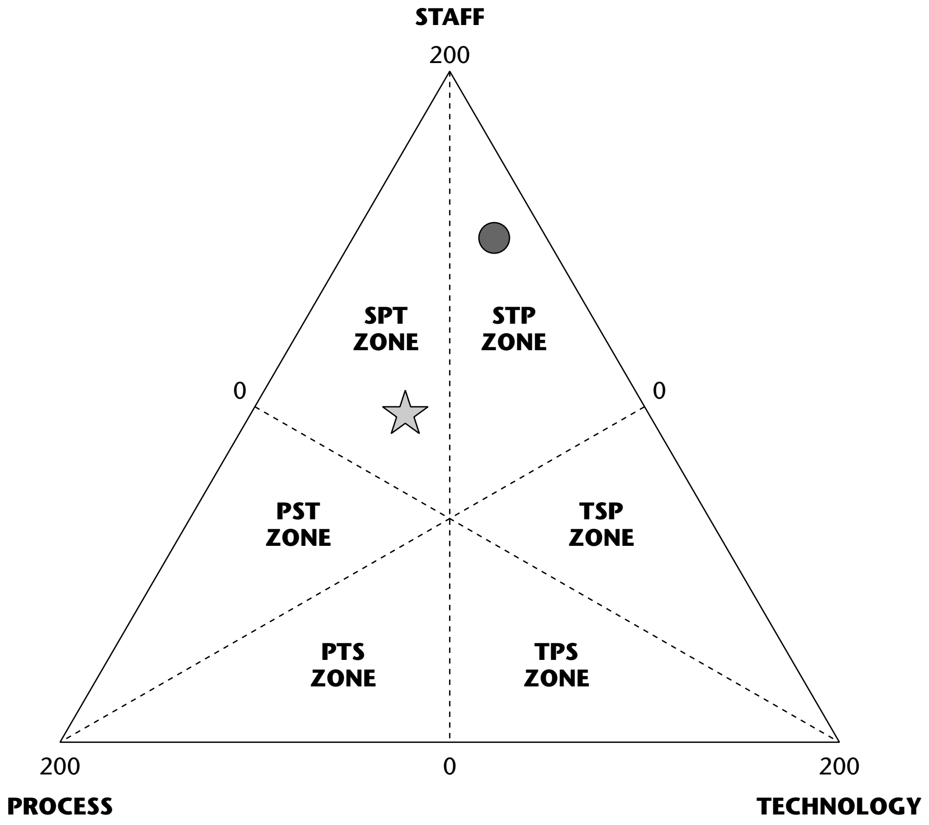


Figure 1-13: Staff-driven environments

Process-Driven Environments

Figure 1-14 illustrates the two process-driven environments that might be encountered.

In the first case (denoted by the star) choice of SDPM strategy drives staff, and together staff and SDPM strategy drive the choice of technology infrastructure. This case reminds me of organizations that might have only one SDPM strategy—a one-size-fits-all approach. In an iterative, adaptive, or extreme world, that can spell disaster. I have long advocated a project classification rule that puts the decision on SDPM strategy in the hands of the project team where it should be. To reverse the order is to put the organization in Quadrant 4—a solution out looking for a problem.

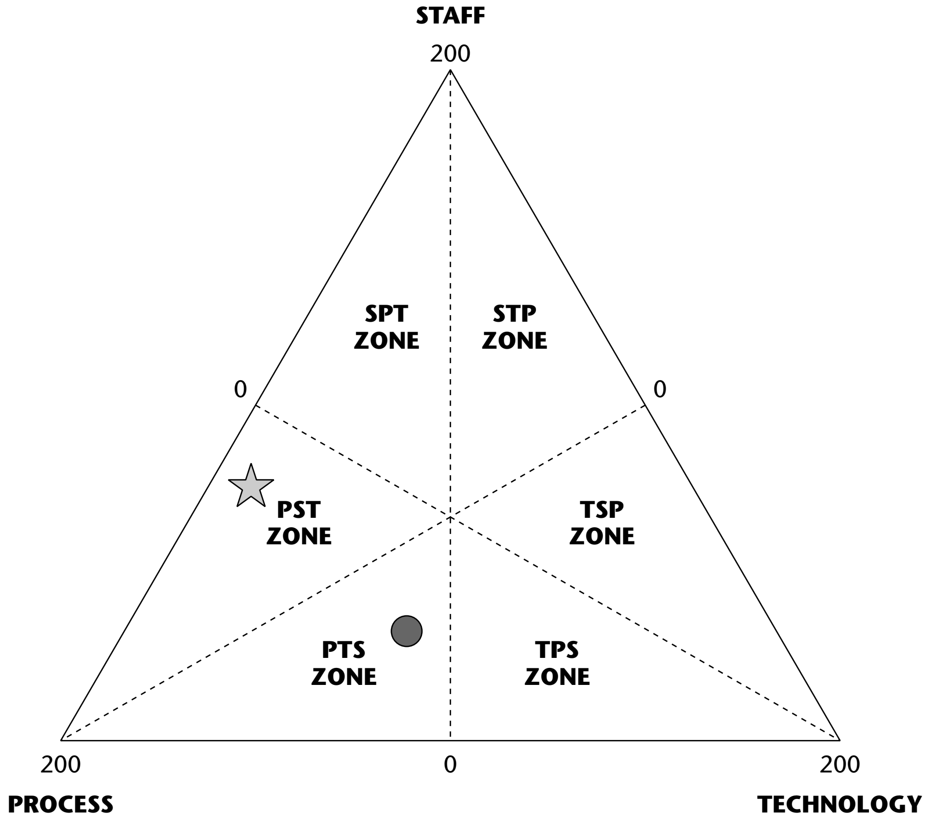


Figure 1-14: Process-driven environments

In the second case (denoted by the circle) SDPM strategy drives the choice of technology infrastructure, and together SDPM strategy and technology infrastructure drive the choice of staff. The die is cast early in the process and now the organization must have an effective and timely recruiting, hiring, and professional development plan in place to assure that project teams are staffed with capable members. This might not be a problem, but if it is, it is a problem that could have been avoided altogether. Why create your own problems when there are enough of them going around?

Technology-Driven Environments

Figure 1-15 illustrates the two technology-driven environments that might be encountered.

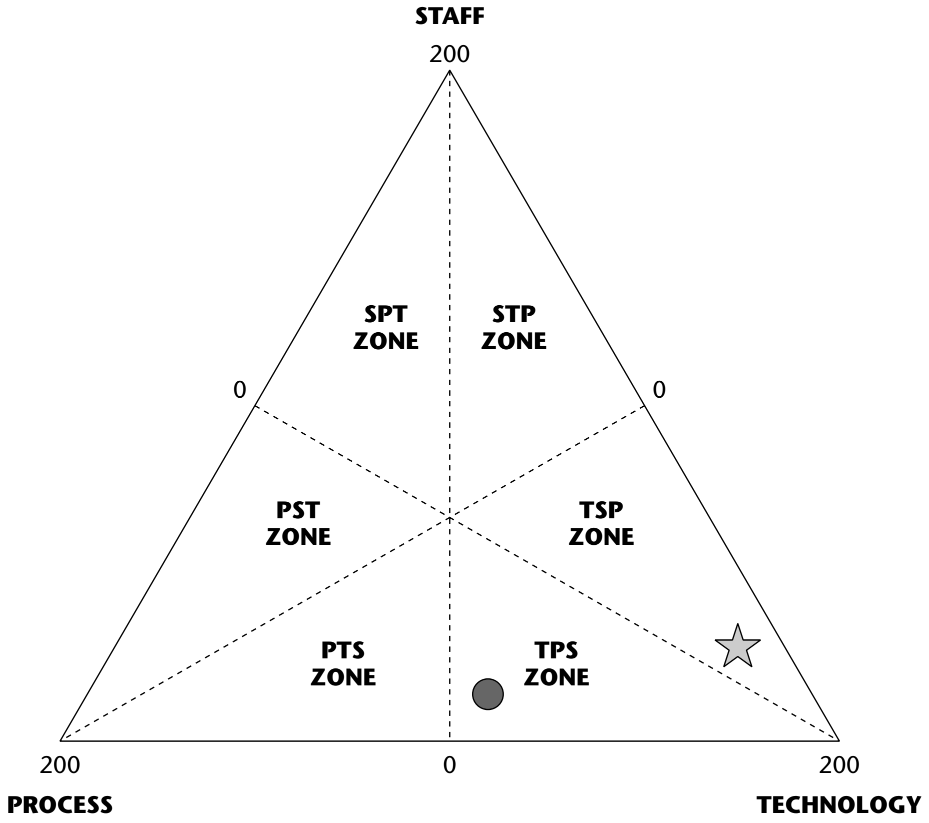


Figure 1-15: Technology-driven environments

In the first case (denoted by the star) the technology infrastructure drives the choice of staff, and together staff and technology infrastructure drive the choices for SDPM strategy. As long as the technology infrastructure doesn't prove to be a binding constraint on good project performance, the situation is workable. Most organizations find themselves in this or closely related situations. Earlier technology infrastructure decisions define the world of the software developer and to some extent the world of the project manager. In the short-term that constraint is fixed. The staff will have been chosen to be compatible with that infrastructure, and that is as it should be. The last variable, the SDPM strategy, is thus constrained to that environment. That might be no issue at all or it might be a serious constraint.

In the second case (denoted by the circle) technology infrastructure drives the choice of SDPM strategy, and together the technology infrastructure and SDPM strategy drive the process choice of staff. This can be made to work. It all depends on the earlier choices and how ready the staff is to embrace those decisions. If staff is given the authority to adapt the SDPM strategy to the project situation, this will work. If the SDPM strategy has been defined to accommodate further adaptation, the teams will have a better chance of success.

Discussion Questions

1. For years there has been debate over whether the development team should be a team of specialists or a team of generalists. Given what you have learned about the software development landscape, what are your thoughts about specialists versus generalists? Does your opinion change depending on which quadrant the project is in? Why or why not? Be specific.
2. What relationship, if any, exists between risk and business value for projects in Quadrant 1, 2, or 3?
3. Many teams have problems getting and maintaining meaningful customer involvement. What have been your experiences—both good and bad?
4. If the frequency of scope change requests is beyond your expectations and it has seriously compromised the project, would you ever consider changing the approach from a linear/incremental one to an iterative/adaptive one? Why or why not? If not, how would you deal with the problem? Be specific.

5. What type of organization do you work for? Is it staff-driven, process-driven, or technology-driven?
6. What type of organization do you work for? Is it staff-driven, process-driven, or technology-driven? What types of problems have you seen that may be the direct result of the type of organization? How might you go about correcting the problems?