

PROCESS PATTERNS

CHAPTER 3



Process Patterns

*To some physicists Chaos is a science
Of process rather than state,
Of becoming rather than being.¹⁹*

Process Patterns are equivalent to a meme complex (a group of mutually supporting memes) that metaphorically relate to the software development process. Adaptation of these memes by a software development organization should result in a cultural transformation consistent with the goal of leveraging technology for competitive advantage.

This transformation, along the lines alluded to in this chapter, and described from different angles in subsequent chapters, is the secret sauce that has, for the most part, eluded business computing organizations (and others as well). It is by far more important than the latest wiz-bang technology. In order to transform the culture, you must infect the organization with the appropriate memes and disinfect

it of inappropriate ones. This implies a deep understanding of the desired results, even as you walk down a road less traveled.

Work in Progress

My intent is to provide a starter set of patterns, a hint of the possibilities. Hopefully, other practitioners will contribute to this effort over time. I will be more than happy to aggregate the content and publish it on an appropriate site (e.g. www.theB2Bdepot.com).

Most patterns deal with the non-linearity of the communications process during software development. Non-linearity means that the act of playing the game has a way of changing the rules²⁰. In terms of the software development process, it matters who the players are and what they do (to, with, and for each other). Most Process Patterns relate to the social and organizational interactions of the players, as well as the consequences of these interactions.

The premise and overriding principle of Process Patterns is that they impact the manner in which the players interact. Positive patterns improve the human dialog among team members by orders of magnitude. Improved dialog dramatically reduces the DIP, resulting in quality and time-to-market advantages, as described in Chapter 2. The willful use of Process Patterns drives cultural changes by allowing the formation of creative and productive spaces.

Patterns = Good & Bad Behaviors

Some patterns are negative and increase entropy (see *Bad Actor & Blame Game* sections below). Therefore, they need to be understood so they can be recognized and eliminated. However, most patterns are entropy-reducing and positive in nature. These patterns need to be quickly introduced into the meme pool and propagated. The Missing Factory is more of an organizational pattern, as opposed to a process pattern, and will be discussed in Chapter 5. The remaining patterns are discussed next, in no particular order. Many of the pattern names have connotations within the film industry. That was not the intent when I started down

this road, but it began to make more and more sense, so much so that I decided to dedicate an entire chapter to the similarities.

I must re-emphasize the fact that pattern names, while appearing to name roles, are generally intended to describe *behaviors*. These behavioral patterns, although principally embodied in a specific role, increase in effectiveness the deeper they are embedded within the consciousness of the team. The wider and deeper the distribution, the more effective communication becomes among team members. In other words, any and all team members should be capable of invoking a specific pattern when necessary, their individual talents permitting.

A Set of Heuristics

I need to clarify one important point before proceeding. Process Patterns can be considered interesting heuristics, but they are not intended to replace iterative methodologies, such as the Rational Unified Process (RUP). In fact, Process Patterns can be thought of as observable behaviors that provide insights, and complement such methodologies. I am a strong believer in iterative development and often use a modified version of RUP as a development guide.

The following Process Patterns will be covered:

- Call to Arms
- Supporting Cast
- Producer
- Director
- Bad Actor
- Key Abstractions
- Breaking Ground
- Collaborator
- Organizing Principle
- Planning the Plan
- Wandering Around the Desert
- Super Star
- Blame Game

Call to Arms Pattern

Call to Arms concerns itself with motivating the troops to perform at the extraordinary levels required to deliver compelling solutions. The pattern should be delivered early by the *Producer* (e.g. the executive sponsor) and subsequently (at critical points in the project) by the *Director* as deemed necessary. The importance of the mission must be constantly reinforced.

If this is a mission-critical project (which, by definition, any project purporting to deliver a competitive advantage would be), then a number of key (and important) stakeholders must take an interest in it. You know you have succeeded in infecting the extended team (all the players as opposed to just the IT players) with this meme when you see the pattern used by individual, non-management contributors.

Ned Johnson, the man primarily responsible for turning Fidelity Investments into the financial powerhouse it is today, delivered many *Call to Arms* messages by his mere presence:

His chief passion seemed to be computers: the installation, maintenance, and use of the giant mainframes that would be the backbone of the company in the 1980's was what most engaged him, and he was constantly in the computer room, watching, asking questions, tinkering with the machines. Naturally, his staff thought that he was pouring too much money into computer capacity, and viewed his obsession as folly. Naturally, they would be proved wrong as soon as the bull market [of the 1980's] began.²¹

In the many years that I have spent in this business, I have rarely witnessed key executives paying close attention to *their* technology projects, even those deemed to be mission critical. How do you think the employees at General Electric might have felt if, in the days of Jack Welch, he started hanging out with a project team? My guess is that they would have been motivated beyond belief. The *Call to Arms* meme would have spread through his organization like wildfire.

Delivering the message is important, but the pattern will replicate itself faster by the things that you do.

Supporting Cast Pattern

The *Supporting Cast* relates to the edification of certain individuals and groups who have immense power to contribute significantly (or not) to a project's outcome. Depending on the project, the supporting cast might include prospects, customers, domain experts, quality assurance, technical writers, office administrators, contractors, junior staffers, and secretaries. It is important for the *Director* to make a concerted effort to propagate this meme by inviting the supporting cast to appropriate meetings, and by celebrating their contributions. It is critical that the supporting cast be edified and made to feel important because:

- Their contributions can make or break your project.
- Everyone's head must be in the game if you are going to deliver a compelling work product.

Do not confuse this pattern with altruism. If you value the prize (i.e. competitive advantage), then you will propagate this meme out of pure selfishness. Oftentimes, the best insights come from places where you least expect them.

Producer Pattern

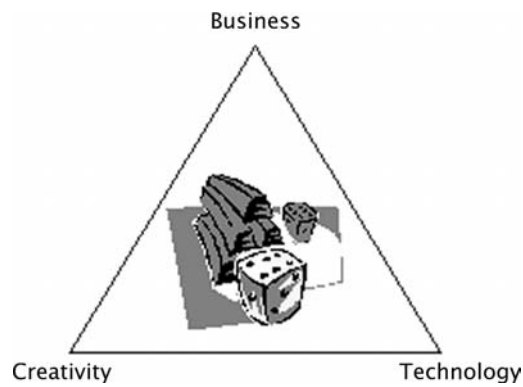
The *Producer* is all about *'da money*. It embodies the business and fiscal implications of the current project, and captures the viewpoint of the executive sponsor who is risking *their* reputation, and oftentimes *their* money, on the current venture.

When this meme propagates to the entire team, the work becomes grounded, thereby helping team members keep things in perspective.

It is often all too easy, especially for technologists, to forget who is paying the rent, and for that matter, that the rent needs to be paid at all! The *Producer* must imbue

the team with this pattern so that all project stakeholders focus on business and fiscal responsibility, or at least develop a keen awareness of these issues.

Maintaining a business perspective often facilitates appropriate tradeoffs when team members are debating alternative strategies. Show ‘em the money and the team will learn to have a healthy respect for it. The graphic below represents three tradeoffs that must constantly be balanced in any project.



Three Tradeoffs

Director Pattern

The *Director* is all about leadership and vision, and is responsible for promoting the project’s story and ensuring that entropy is fought every step of the way. It is principally responsible for meme propagation and insights. It concerns itself with timely decision making (or breaking) and maintaining the tenuous balancing act between creativity, business, and technology.

The *Director* is the creative genius and the rock. It is the pattern (and the role) that project team members must turn to when (not if) they find themselves *Wandering Around the Desert* from time to time. It provides both food and drink when nourishment is needed, and it accepts responsibility without attempting to assign blame.

The *Director* is also a problem-solving pattern. It concerns itself with the project as a whole: part inspiration, part perspiration, part genius, and part sheer will and determination. It is the heart and soul of the endeavor, willing to sacrifice anything (almost) for the end game.

Where there is no vision, the people perish. The *Director* is the keeper and the defender of the flame. It is the *Director's* responsibility to confront *Bad Actors* and



The Director Pattern

to inspire great performances instead of simply good ones. The *Producer* may put up the money, but it's a bet that the *Director* will deliver the goods. This pattern (and role) requires passion.

Movies aren't for me a piece of life. It isn't a job, it's everything. You can't make them unless you are obsessed. They tend to take you over. They guide your dreams as much as they guide your waking. There's no escaping movies. ²²

–*Director Phillip Noyce*

The same holds for projects that contain software as a principal component. Without the *Director* meme, great software is incapable of coming to life. The *Director* must be an embodiment of what Jim McCarthy calls the “group psyche”:

*Your personal authority stems from knowing how to create the team, to initiate its growth, to double and redouble its growth, while empowering the team with good techniques and procedures for creating great software and shipping it on time. Your vision is of the group psyche and of the product of the group psyche, and of how the two relate. Your goal is to be a genuine authority in these matters, not some bogus institutional authority...*²³

Bad Actor Pattern

The *Bad Actor* is an individual, in a specific role, whose behavior tends to increase, rather than decrease, the entropy of the system. It is the responsibility of the *Director* to modify this individual's behavior early in the game (or at any other time when the pattern is manifested), or to eliminate that individual from the role (but not necessarily the organization).

Bad Actors tend to have lots of questions, but they appear totally incapable of contributing to a solution. They tend to postpone or outright sabotage the decision making process. They are cancerous, and if their behavior cannot be dramatically improved, it must be cut out or you risk killing the patient.

Big Automation Company hired a project manager to lead the development of their next generation simulator. This individual appeared to have all the right credentials and pedigree. She had a solid technology background, project management expertise, and similar product experience. I was the Chief Architect, but it was *her* project.

Initially, I coordinated some *Breaking Ground* activities in order to get the ball rolling. I gradually eased myself out of the process in order to let her assume a leadership role (which is what she was hired to do). After three months, it became agonizingly painful to endure the endless excuses as to why she could not develop a meaningful project plan.

This was the *next generation* simulator. The current product had been extremely successful in the marketplace, but it had, by now, become long in the tooth (which is an understatement). The plan was to re-architect and re-develop the product using state of the art technologies, and to also add some features that would leapfrog the competition.

In other words, the next generation product had to functionally do everything that the old product did, and a little more, only with a different set of base technologies. We knew the “what,” it was the “how” that was in question. This *Bad Actor’s* primary complaint was that no one had sufficiently established the requirements (never mind that it was her responsibility to do so).

Finally, after numerous (far too many) attempts to rectify the situation, my patience was stretched to the breaking point. I was forced to call *bullshit*. Most of the requirements were embodied in the current product. The task at hand was something that a competent project manager should have been able to handle with aplomb. But there was another dynamic at work here (i.e. fear, uncertainty, and doubt). The entropy was increasing at a significant rate and impacting the entire team.

The product development manager eventually took the appropriate action and removed this individual from the organization (since there was not another

suitable position for her to assume). The removal of *Bad Actors* is a painful process for all concerned. However, it comes with the territory and must be handled in a professional but expedient manner. Elimination of *Bad Actors* will dramatically improve the script.

Key Abstractions Pattern

Key Abstractions are intimately related to the project's story line. It is a term borrowed from the Rational Unified Process (www.rational.com) and given a somewhat broader meaning within the process pattern context. In technical terms, key abstractions refer to entities (people, places, and things) about which persistent information will be maintained. There should be widespread agreement among all project stakeholders relating to the centrality of these entities, since the entire story is subsequently built around them.

At the project's inception, we are dealing with first order *Key Abstractions*. For any non-trivial project, there will exist second order *Key Abstractions*, third order *Key Abstractions*, and so on. During inception, it is next to impossible to deal with any abstractions beyond those considered to be first order. Moreover, it is not necessary to do so. Subsequent abstractions will become obvious as the project's iterations progress, and they will become easier to recognize and/or define at the appropriate time.

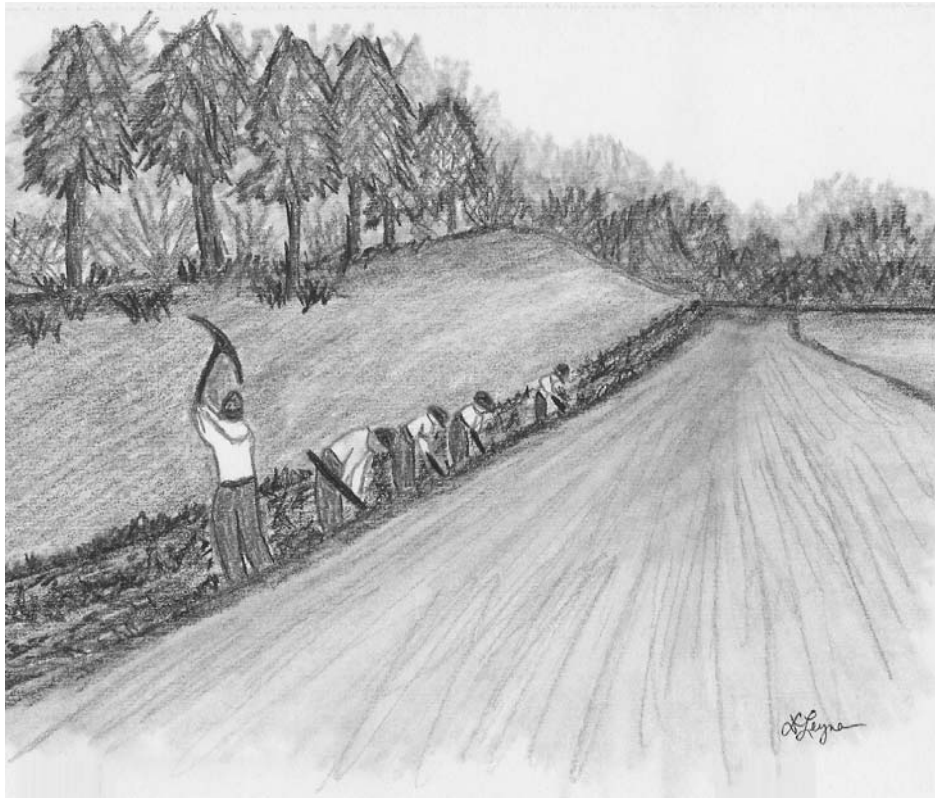
Once *Key Abstractions* (of whatever order) have been identified, it becomes necessary and plausible to identify the processes that act upon these entities. Given a set of processes and a set of *Key Abstractions*, a reasonable iteration plan can be developed. Therefore, you should focus on *Key Abstractions* in their relative order of importance.

Breaking Ground Pattern

Breaking Ground relates to getting started—duh. You might think that this pattern is only manifested at the start of a project, but you would be totally mistaken. Each non-trivial project is composed of hundreds, sometimes thousands, of smaller sub-

projects that are not explicitly captured in the plan. Even experienced project hands often find it difficult to get started on these. What results is akin to the writer's block phenomenon. Sub-projects often require small R&D efforts and can send team members down a number of tangents. Critical intervention can jettison the effort by reducing the DIP.

It is the responsibility of the *Director* to ensure that *Breaking Ground* happens early and often. At the project's inception, this usually means developing the project story and coming up with the first order *Key Abstractions*. Until the project story (as embodied in the *Key Abstractions*) is reduced to paper, nothing can happen—and nothing should happen.



Breaking Ground Pattern

If it hasn't been written down, it hasn't been said. Be sure to institute a lightweight process that requires that all important and relevant project decisions be documented.

–D. Leyva

The *Breaking Ground* pattern should be invoked whenever the *Director* suspects that writer's block is preventing progress. The *Director* must be constantly vigilant in seeking out *Breaking Ground* opportunities, since team members are often reluctant to come forward of their own accord. They are, more often than not, likely to thrash about and ask for help only out of desperation. Keep your shovel in hand. *Breaking Ground* opportunities are everywhere!

Collaborator Pattern

The *Collaborators* act as the team's neuron cells. These are principal meme receptors and propagators. The *Collaborators* love white boards and heated constructive arguments. Winning or losing is not the crux of the matter. The improved understanding that results from the battle is the prize.

In order to increase the velocity of this meme, the *Director* must ensure that there are ample war rooms available for impromptu engagements, and that team members feel “safe” to explore ideas without fear of retaliation or humiliation. Team members who are hopelessly infected should be encouraged to engage in the fray as often as possible. Those not infected need to be seduced onto the battlefield.

One of the side benefits of this pattern is the identification of *Bad Actors*—you bet your ass! *Bad Actors* easily tire of war room battles, and they tend not to participate regularly. Why is that? Well, it is mostly due to the fact that arguments against a position are only considered valid if a better alternative is proffered. If a better alternative is not offered up for dissection, then the contestant is called on the carpet for whining. A contestant that consistently whines is almost always a *Bad Actor*.

Although there is certainly a role for *Super Stars*, it is the contribution of every team member that collectively defines success. Individual stars must be rewarded relative to the team's success, and relative to their contributions as *Collaborators*. However, that said, do not hesitate to adequately reward a *Super Star*. Remember, collaborators run the project's central nervous system.

Organizing Principle Pattern

Organizing Principle refers to context and connectedness. Rarely in today's software universe does a product stand alone—either for external release or internal consumption. One-off offerings are headed for a short shelf life and will quickly find themselves with a permanent residence in history's dustbin.

The project team must keep in mind the relevant stories that the current effort plugs into (i.e. has to play nice with). These stories can be standards, product suites, architectures, add-on offerings, protocols, etc. The *Organizing Principle* must be made as explicit as possible during the development of first order *Key Abstractions*, and then propagated continuously during the project's lifetime.

The *Organizing Principle* is an important forward-looking pattern that must be constantly revisited in order to ensure a degree of seamless integration with surrounding stories. One-off offerings are DOA before launch.

Planning the Plan Pattern

Planning the Plan relates to the planning process itself, not the concrete deliverable. It is the entire team's responsibility to participate in the process. However, during the inception phase of the project, the responsibility for *Breaking Ground* falls most squarely on the shoulders of the *Director*.

The *Producer*, at this point, has tentatively decided to pursue the project and has some vague idea about the details of the value proposition. It is up to the *Director* and the team (management, sales, marketing, technology, etc) to clarify the value proposition and devise a strategy for attaining it. In order to expedite the process,

the *Director* must frame the problem by developing a “straw man.” The straw man contains the *Director’s* initial thoughts related to the following (non-exhaustive) list, depending on the project:

- Project Theme and Story Line
- Key Abstractions
- Organizing Principle
- Development Plan (RUP)
- Architecture Plan (RUP)
- Requirements
- (Very) High Level Use Cases

Once the *Director* has gathered and documented (crudely) his or her initial thoughts, he or she meets informally with key stakeholders to surface the ideas and refine them. The straw man is a device that helps plan the plan. It is pure tradecraft—far removed from anything that vaguely approximates a science. Anyone that claims otherwise is completely (and utterly) full of shit.

The straw man should take no more than a couple of weeks to develop. Once developed, it is time to meet with the entire team for a kickoff discussion. The purpose of the kickoff discussion is to interrogate the straw man. The interrogation(s) result(s) in a list of tasks required to fill in the holes (i.e. identify risks, gather additional requirements, develop rigorous use cases, further define project economics, etc.). Filling in the holes should take (approximately) two to eight weeks, depending on resources, complexity, etc. If the holes cannot be filled to the team’s satisfaction, then throw the fucker out and start again.

At the point that holes have been filled, the inception phase of the project is complete. Do you have all the answers that you need? Can pigs fly? The RUP defines only four project phases:

- Inception
- Elaboration
- Construction
- Transition

The *Elaboration* phase contains iterations that are aimed at reducing risks and getting *real* answers. Okay, so what's the point? It is an exercise in futility to drag the Inception phase on and on in search of perfect requirements (use cases, economics, or any other fucking thing). Software is n-dimensional, temporal, and organic; it is a living, breathing beast.

You will never be able to model the beast on paper. If you want to understand the beast, you need to interact with it! Only by interacting with it can you determine if the beast should live or die. This determination is made during *Elaboration*.

Elaboration iterations should be planned in the same way. Develop an iteration straw man. Get some feedback. Refine it. Meet with the team and get task estimates. Refine them. The planning process should be as iterative as the development process. Expedite it by *Breaking Ground* early and often.

If you make it to the construction phase, then, by definition, you have all the answers you need to give the beast life (or so you think). During construction, you build the beast. During transition, you cut the umbilical chord and hope that the world treats this thing kindly. Hopefully, it survives birth, adolescence, college and goes on to have a long and distinguished career.

Planning is a motion creation tool. Planning reduces entropy by organizing chaos. A plan is outdated and wrong, with respect to specifics, as soon as it is committed to paper. However, without one, you cannot proceed. It is a development tool and nothing more. Learn to plan the plan and don't hesitate to modify it when necessary. And remember, the actors must help write the script.

Wandering Around the Desert Pattern

Wandering Around the Desert relates to what goes on during the middle of a project. During inception and early iterations of *Elaboration*, you are generating excitement, anticipating the prize, building momentum, identifying and eliminating risks, and selling the story. There is a perceptible *buzz* around the team's activities. You know that you will be challenged and you gear up for it.

During the middle of the project, the long, hard winter sets in. You start to believe that you have answered the disbelievers, eliminated the most serious risks, and have all the executive sponsorship you need to get you through the night. All that remains is flawless execution and a tremendous amount of work. The excitement diminishes. The work begins to take its toll. You can't hear the buzz anymore. Things start to slip.

In every project, there comes a time when you wish with all your heart that there was some way out. Everything is in doubt. Cynicism and mutiny are in the air. No one has seen land for months. The crew is suffering from scurvy, the larder is empty, the stormy seas seem endless.

*Welcome to the middle game.*²⁴

It is time for the entropy police (the *Director*) to go on patrol 24/7. The *Director* knows that the team and project are a long way from being out of the woods. In fact, the forest and trees continually converge and then disappear. All that is left is miles and miles of desert and a very unreliable compass called your instinct—which can't be very good or it wouldn't have led you into the fucking desert!

The only ray of hope is that the *Director* knows the desert and is able to:

- Find food and water at the appropriate times.
- Recognize landmarks that lead to nearby villages, wherein the council of elders can be called upon.
- Know when to push and when to rest.

- Be constantly vigilant about team members succumbing to sun stroke.
- Mitigate disputes that arise as team members start to lose faith.
- Know that progress must be made each day, even though there is no guarantee that the team is walking in the right direction.

In the middle of the desert, bandits that you thought were friends will try to ambush you. You must anticipate their arrival and forewarn your team of their inevitable appearance. They will strike with greater force and frequency the closer you are to emerging from the long and arduous pilgrimage.

The final problem of middles is often the most frustrating. Even if you have built a coalition and involved key stakeholders, the critics, the skeptics, and cynics will challenge you. They will be strongest and loudest not at the beginning but in the middle of your efforts, just when the project itself is not quite ready and thus is most vulnerable. It is only then the possible impact of the change becomes clear. And those that don't like it have had time to formulate their objections and harden their positions...And now that it looks like the venture might succeed, the threat to those that oppose it increases.²⁵

The desert can be conquered, but no matter how many trips you have made, it always poses a life-and-death challenge. Don't celebrate too much during early stages because the desert and the bandits await you. Be ready.

Super Star Pattern

Super Star refers to exceptional and bankable actors. These are individuals who are head and shoulders above their peers (e.g. Michael Jordon and Larry Bird), and those who tend to make everyone around them better as well. Every project team should consider itself blessed if it has a *Super Star* in its cast. (But more than one may be problematic.) Not only do these individuals pull their weight, they often pull more than their fair share.



Superstar Pattern

Do you believe that it is reasonable, and just, for *Super Stars* to be compensated only marginally better than their peers? Would Al Pacino or Robert De Niro be happy with this type of compensation, knowing they are the box office draw? It is widely known, for example, that the best developers are capable of producing more quality “stuff” than average developers. In business computing organizations, I have never seen this compensation chasm adequately addressed. I’m not suggesting that you should pay *Super Stars* ten times more than everyone else, but you do need to recognize their achievements and provide preferential compensation.

All this talk about the importance of teams should not prevent us from treating special individuals in unique ways. Money should always be part of the equation, but preferential treatment need not stop there. There are other ways to compensate, such as paid vacations, T1 lines at home, or whatever. Use your imagination! Yeah, this will drive HR up the fucking wall, but so what? I will go out of my way to accommodate this type of individual, as long as the reward system does not turn them into a *Bad Actor*.

If you are a *Director* and you are asking yourself how you know when someone is a *Super Star*, then you should fire yourself. A good *Director* always knows who their *Super Stars* are! And you should be nurturing and compensating them for the outstanding work they do.

Blame Game Pattern

The *Blame Game* is the most insidious of patterns. It is anathema to team dynamics and causes a cancerous growth. Once it reaches a certain magnitude, it becomes next to impossible to excise. In business computing organizations, the *Blame Game* is a team sport (often played to the death) between end users and technology. In companies that ship commercial software, the sales and marketing organizations replace end users as contestants. In both cases, symptoms and remedies are the same.

Before suggesting a remedy, let's examine the root cause and discuss the reasons why the game always has a tragic ending. The premise is that you have two disparate organizations jointly responsible for project deliverables (e.g. finance and technology). The inevitable dispute centers on that most venerable of words within computing history: requirements.

The business people are supposed to provide correct requirements. This is because the underlying assumption is that they know what they want, and they know what they are doing. Both assumptions are almost always incorrect. Business people have (almost) never been trained in the development of formal requirements. They typically do not think in terms of systems, entities, and relationships between entities—or any of the other arcane artifacts and constructs that technology people require in order to get what they need. They also perceive (correctly) that all of these activities are a royal pain in the ass and an additional burden to their already busy schedules. They see the requirements gathering process as all downside and no upside. Is it any wonder that they are not active, willing and enthusiastic participants?

When technology people abdicate their responsibility for requirements, they set themselves up to lose. They also set up a scapegoat. Requirements must be coaxed out of business people. Technology folks must become business savvy enough to facilitate this process. They must develop “straw men” that provide a point of reference. Business people are significantly better at telling you what is *wrong* than at telling you what is *right*. They are far better at telling you what they *don't like* than at telling you what they *do like*. It is a dereliction of your responsibilities and inexcusable not to have a better understanding of the business, so quit blaming the business people for incorrect requirements.

When business people change their minds “willy-nilly” in the middle of a project; when they ask for “must have” features without considering costs; when they ask for functionality that is obscure, absurd and often ridiculous in the grand scheme of things—they are setting themselves up to lose. They are also setting up a scapegoat. Business people must become more techno-literate. They must learn the basics of the process. They must become team players and accept their share

of responsibility, as well as their fair share of the load. They must do this in order to ensure team success and to get their fair share of the credit. So, quit blaming the technology people for your carelessness and afterthoughts.

There is only one remedy. There cannot be two teams on any project that is mission-critical and that is supposed to provide competitive advantage. There is one team. The *Director* reports to the *Producer* and everyone else reports the *Director* (either directly or indirectly). The *Blame Game* ends because everyone is to blame. There is only one Berlin. The wall must be permanently and irrevocably blown up and dismantled. We live together and we die together, and anyone who continues to sit on the fence gets shot first.

Eliminating the *Blame Game* is a major first step toward a successful blockbuster movie, and it is a major first step toward creating one cross-functional (non-dysfunctional) team.

Summary

Damn, almost an entire chapter and I rarely exercised my favorite adjectives. I am definitely fucking up. But not as bad as you are, (Mr. or Mrs. CEO, CIO, VP, VC, and Chief M. F. In Charge), if you don't start paying attention! Follow Ned Johnson's lead. Hang out with the geeks, ask questions, tinker with the technology and participate in the process.

But I must warn you: if you can't stand the heat, stay the fuck out of the war rooms. It's awfully hot and messy in there. Check your titles at the door; they won't do you any good. By the way, bring some value to the battlefield because the geeks are getting damn good at recognizing *Bad Actors*.

*It's about one B-I-G idea: innovation/a "top-line" obsession.*²⁶

And in particular, it's about process innovation within creative spaces. Although this chapter (and book) focuses largely on the software development process, all business processes will soon be impacted. No stone will be left unturned (so watch

out for snakes). Software will permeate everything it touches, and with it, creativity as well.

Far better to dare mighty things, to win glorious triumphs, even though checkered by failure, than to take rank with those poor spirits who neither enjoy much nor suffer much, because they live in the gray twilight that knows not victory, nor defeat.²⁷

—Theodore Roosevelt

You are in the movie business. Hire yourself a *Producer*, a *Director*, a *Supporting Cast*, and as many talented young actors as you can find. Inspire them and they will create the future for you. I guarantee that you won't be disappointed!

During dot-com mania, there was nothing but bad scripts and bad acting. And everyone was busy blaming everyone else. No wonder things turned to shit! Fear, greed, and chaos are no substitute for creativity—they always lead to broken dreams and endless nightmares.