PART

One

Introduction to Project Management and AntiPatterns

CHAPTER

1

Project Management and AntiPatterns

"Rowing harder doesn't help if the boat is headed in the wrong direction."
—Kenichi Ohmae
business-strategy expert

The frequent problem with project management these days is analogous to the project manager being a train engineer. The train track has been laid by the organization and the project manager need only take a train down the track. However, too often the train needs to be taken in different directions than where the track has been laid. The project manager needs to stop the train, get out, and lay track. When this doesn't happen, what lies ahead for the project and the project manager is the great train wreck.

The tracks laid by organizations are the processes, technology, and culture (people) that are steeped within the tradition of the organization. The train tracks represent a pattern of behavior and practice for the project manager. Some of these patterns are good, some bad. Unfortunately, the processes used by organizations become so ingrained that there is a failure to evaluate and reevaluate. In fact, questioning these practices is akin to mutiny. In these organizations you will hear the mantra, "That's the way we've always done it!" What is required is superior leadership by the project manager. The Brawl AntiPattern, in Chapter 2, discusses the need for strong leadership and what is required by the project manager in order to achieve a successful project.

"If you keep doing what you've always done, you'll keep getting what you've always gotten."

—Scott W. Thomas
engineer, author

Fortunately, there are organizations that decide that they must change processes to keep up—to redirect the train. However, in the worst scenario possible, these organizations in their attempt to improve can create 100-plus-step processes for the execution of projects. They have thrown innovation out the window and rely on the rigors of the process to save them, rather than the intelligence of their employees. They need to wake up. A process doesn't save anyone, especially when it's a boat anchor around your neck. Hire smart people and give them room to operate. 100-plus-step development efforts only indicate that management has a need to microcontrol the project beyond the realm of reasonableness and that they implicitly don't trust their project managers.

The ultimate objective is to establish a repeatable, mature process that enables successful software development. The concept is similar to the Software Engineering Institute Capability Maturity Model (SEI CMM). The SEI CMM is a set of standards that deal with people, technology, and process. For example, the Systems Engineering Capability Maturity Model (SE-CMM) of the SEI describes the essential elements of an organization's software engineering process that must exist to ensure good systems engineering [SEI 1995]. The section later in this chapter on software development standards lists the SEI standards available. However, like many standard approaches, consistency and repeatability do not differentiate between good and bad software processes.

To be able to achieve a repeatable and mature process, it is necessary to understand the reasons for failure. Project failure is usually attributable to a deficiency in one of the following critical areas:

- A contiguous and well-defined software development process
- Ability to consistently repeat successful software development processes
- Investment of sufficient money and time to deliver
- Sufficient managerial and technically skilled staff
- Mature or slowly changing technology

These deficiencies can generally be categorized into three elements: people, technology, and process.

Programmed to Fail

How many classes and seminars in system engineering, software engineering, and related disciplines have you taken? How many books have you read

that tell you how to properly manage a software development effort? If you are like most project managers or developers involved in some aspect of software development, you can probably teach a course or write a book on how to design, specify, develop, test, train, deploy, schedule, and budget your project based on the knowledge you have assimilated through various educational vehicles.

The next question we must ask is, "If we have such well-trained project managers and engineers, why do we still see so many failures or instances of failure associated with these software development efforts?"

The answer is truly simple. It is because no class, seminar, or book can possibly define the thousands of variables that exist for any given project, at any given time. There is no steady state in software development. While each of the prescribed methodologies and processes that are portrayed in a class or book is valid, the approach assumes an otherwise perfect world (an unrealistic world). No professor, lecturer, or author can possibly know what your specific environment is at the time you embark on your project. You have been set up. Failure, of some type, should be anticipated. The approach a project manager should take is prescribed in the books; however, don't expect that all will go as planned. The real question is, "What do I do when something goes wrong and how do I avoid this problem in the future?" That is where AntiPatterns can help.

AntiPatterns are a revolutionary approach to software development. AntiPatterns help you refactor a solution when things go wrong. The concept of AntiPatterns is leveraged from the principle of using patterns to specify a good solution process or working practice and learning from other people's mistakes. AntiPatterns approach problems from a practical and pragmatic perspective. AntiPatterns prescribe a refactored solution for your problem. The refactored solution is not an end-all but rather a point solution for a specific problem in time, from which other variations can be derived.

Software projects fail at an astounding rate. Even those projects that succeed realize some type of failure, at some time. The question is why? The answer is elusive for technically oriented people. Engineers and technocrats expect that all processes, methodologies, people, and technology are binary. Obviously that is not the case and we do not have to look very far in our personal lives to understand this. We live in a random world. While we can predict with some certainty a majority of the variables at any given time, we cannot predict all of the variables, all of the time. It is an indisputable fact.

For example, we can predict with reasonable accuracy the time we will leave for work in the morning. This is achieved by instituting a process, with

known relationships to both time and performance. We rely on technology (alarm clock, car) and personnel (us, our spouse) to achieve the process. However, we cannot predict the one morning our cars won't start. However, once we know that our cars won't start, we can refactor a solution knowing the variables at that moment. For example, identifying that the fuel gauge shows empty means that the refactored solution to the problem is to fill the car with gasoline. Alternately, the problem may be more complex where we enlist the help of a mechanic to identify the specific cause of the problem and the required solution, such as replacing a fuel line. Once we survive this instance we can document (in our minds) an AntiPattern and we are aware of how to respond in the future. In fact, we can try to prevent the AntiPattern from occurring once we've mapped the symptoms and root causes. A car, for instance, can usually be prevented from not starting by ensuring that basic automotive requirements are fulfilled, such as regular servicing and sufficient gas supply.

So it is with processes in our software projects. There is randomness and we cannot anticipate every possible failure. However, we can prepare for failure and try to recognize it before it occurs. Imagine that you had the knowledge of all possible failures, their symptoms, root causes, and refactored solutions. While you may not catch every failure before it occurs, you may catch some, and the others you would be better equipped to address when they occur. That is the concept of this book—to document the common software development project management AntiPatterns so that you will be better equipped to prevent and respond to those failures.

People, Technology, and Process

People, technology, and process are the major variables in the failure and success of software development projects. If any one of these three is poorly dealt with, then the software development project will suffer. Steve McConnell identifies some of the more prevalent classic mistakes that occur if the balance between these critical factors, such as those in Figure 1.1, is not addressed sufficiently. Both Jim McCarthy and McConnell are the de facto gurus of software development [McCarthy 1995, McConnell 1996, McConnell 1998]. Any of the mistakes can cause further mistakes to occur and result in the following [Moynihan 1989]:

- Cost overruns
- Premature termination of project
- Development of the wrong product
- Technical failure

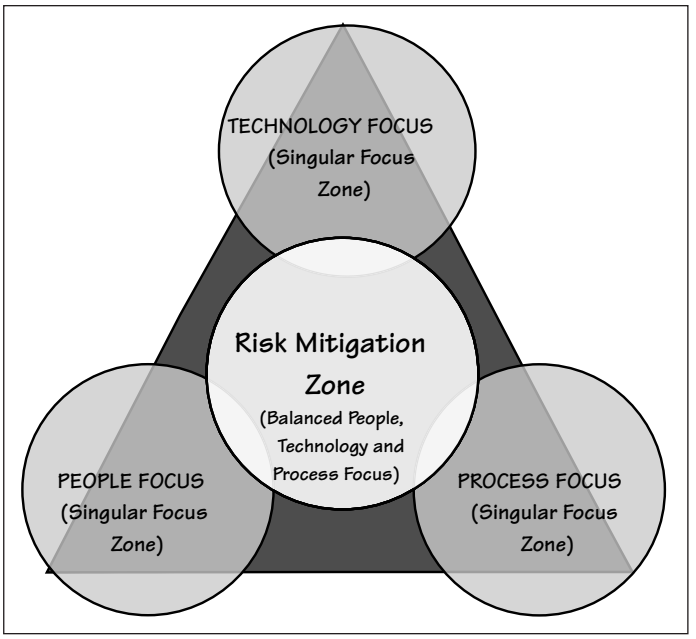


Figure 1.1 Drivers of failure.

These critical areas of focus are fully dealt with in the Micro-Management AntiPattern in Part Two of this book.

People

The most difficult aspect of project management ultimately becomes the management of human resources. The management of the project team, support team, customers, and executive management (bosses) all contributes to the most grueling facet of management. Even under the best circumstances there is friction between individuals; this is human nature. When the human function is added to an environment where interaction and cooperation are critical for success, and then this is compounded by a compressed delivery schedule using minimal resources, the complexity and the magnitude of the project are all too frequently overwhelming. Chapter 2 covers the collection of people AntiPatterns.

Software development gurus like Steve McConnell and Grady Booch have focused on social interaction because the sociology of the software development environment is poorly understood, and yet it has such a major impact on even the smallest task. The political aspects of the cliques and hierarchies make up the dynamic fabric in most software development organizations, making it one of the most valuable assets of a software development project manager to master and control.

People interaction occurs in two ways: horizontally and vertically. *Horizontal interactions* are between developers or managers in the same peer group, such as programmers, architects, or team leaders. Horizontal interactions require strong cooperation to achieve shared goals. *Vertical interactions* are between subordinates and superiors within a corporate hierarchy. Vertical interactions require iterative negotiation to ensure that the subordinate(s) own the task that they are implementing and that the manager understands the importance of the feedback from the developers.

Technology

Technical risk factors are attributed to the organization's maturity state as well as its understanding of technology and the maturity of the technology. Both development-time and runtime commercial off-the-shelf (COTS) software, such as a code library or database, can create cost sinks and extreme versioning problems in order to deliver a product. Often COTS software is embedded within the runtime products, creating technical and financial dependencies.

The primary technology areas that require understanding are tailored COTS software and programming environments. This is because often the stability is poor, the interoperability weak, and the rate of codebase change frequent. Competition drives organizations to always push the envelope of technology, and technological products are forced to be dynamic due to product differentiation and product improvements. Of course, the amount of technological advances an organization is willing to undertake is always mitigated by the risk the organization is willing to assume. Compromises limit future functionality with the result being having to replace software based on the earlier decision to use more stable software that is rapidly becoming dated. Chapter 3 covers the collection of technology AntiPatterns.

For example, take distributed object technologies (DOTs), where the Distributed Component Object Model (DCOM) and Common Object Request Brokerage Architecture (CORBA) currently dominate. The interoperability has been initially limited to bridges, which do very little but provide a data integration framework, while the DOTs continue to change not just within themselves but within the associated software and specifications, such as the Microsoft Transaction Service and the CORBA Notification and Asynchronous Messaging Services.

The ability of the organization and the software project manager to manage technology is as critical to success for a software development as technical failure is easy to achieve. Basic software project management techniques such as risk management and proof-of-concept prototyping

must be pragmatically used to succeed. Corporate strategies must be balanced with line of business needs for individual projects.

Process

A software development process is meant to be a standard, repeatable set of activities and tasks that produce a required software development deliverable. The problem with processes is that they don't accommodate the variance that occurs with the people and technology. Processes need to be dynamic in supporting these changes. Additionally, they must balance the schedule, skills, technology, organization, and development environment of the project and the project team. Processes must be supported by the appropriate combination of standards, disciplines, and tools that support and enforce those standards and disciplines.

The processes should match the need of the particular type of development. Developing a demonstration does not need the full-blooded processes that are needed to develop a satellite reconnaissance system. The planning, configuration management, and development processes that are extensive and rigid in space shuttle development are greatly reduced and often deliberately not even considered in prototyping production. This is reflected in the various software development lifecycles that are discussed in detail in the Lifecycle Malpractice AntiPattern in Chapter 4.

However, there are some basic software development processes that are critical for any type and scale of development (see Figure 1.2): project management and configuration management. They both need to occur at any level of software development. That does not mean, however, that they have to be as strongly defined and rigorously followed for every type of software development, but the basic principles must be applied in all cases.

Basic planning must include the creation of a schedule of tasks with their interdependencies and estimated effort with planned start and end dates, which is covered by the Planning 911 AntiPattern discussed in Chapter 4. Basic software configuration management must include versioning of critical software development artifacts, such as requirements, design, and most especially code. These topics are dealt with in our previous book, *AntiPatterns and Patterns in Software Configuration Management* [Brown 1999].

Software Development Standards

The best software development processes are standardized so that they can be repeated in an efficient and well-understood manner to give consistent

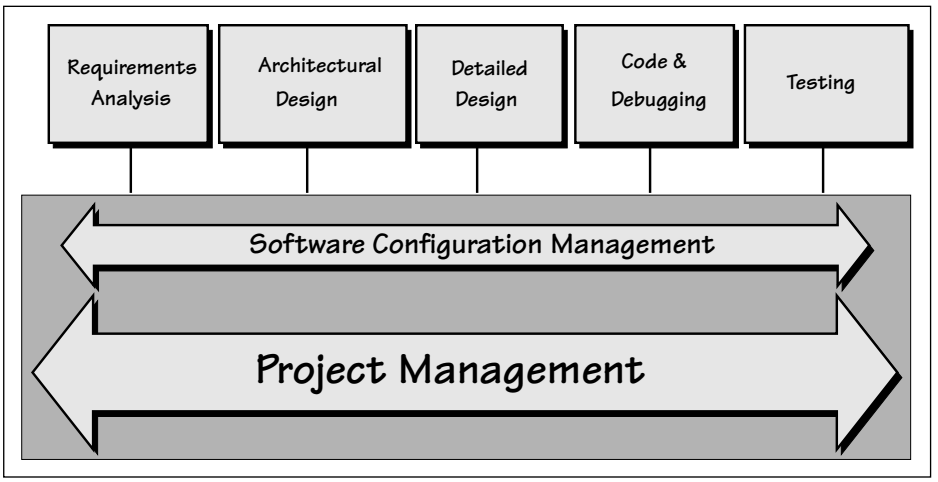


Figure 1.2 Critical software development processes.

high-quality results. Software development standards provide a much-needed template for organizations who are adopting standardized software development processes.

Standards are intended to be a repeatedly implementable solution pattern that enables good practices in software development. This is exemplified by the sterling work of the National Aeronautics and Space Administration (NASA) Software Engineering Laboratory. Software development standards are often tightly coupled to SEI's Capability Maturity Models to measure the effectiveness of software process improvement.

There are four major categories of standards in the computer industry: formal, de jure, de facto, and consortium [Brown 1998]:

- *Formal standards* are those advocated by accredited formal standards bodies such as International Standards Organization (ISO), American National Standards Institute (ANSI), and Institute of Electrical and Electronics Engineers (IEEE).
- *De jure standards* are standards mandated by law and endorsed by a government authority, including such standards as Ada95 and Government OSI Profile (GOSIP.)
- *De facto standards* earn the status through popular use such as in Microsoft Windows and Microsoft Office, Transmission Control Protocol/Internet Protocol (TCP/IP), and the various Internet protocols such as Internet Inter-ORB Protocol (IIOP).
- *Consortium standards* are created by groups of companies supporting a particular cause such as the OMG and The Open Group. Typi-

cally, formal and de jure standards are specifications only, whereas de facto and consortium standards may also include an implementation.

Software development standards can directly affect the three major variables of people, technology, and process. Just consider the Software Engineering Institute's management practices models:

- Capability Maturity Model Integration (CMMI)
- Capability Maturity Model for Software (SW-CMM)
- People Capability Maturity Model (P-CMM)
- Software Acquisition Capability Maturity Model (SA-CMM)
- Systems Engineering Capability Maturity Model (SE-CMM)
- Integrated Product Development Capability Maturity Model (IPD-CMM)

Software development standards are ignored, misused, and blamed for many software development failures. But there are two sides to this argument: pro standards and anti standards. Standards actually can provide many benefits, such as SEI's goals in developing CMMs [SEI 1999]:

- Addressing software and disciplines that have an impact on software
- Providing integrated process improvement reference models
- Building broad community consensus
- Harmonizing with standards
- Enabling efficient improvement across disciplines

Standards can also cause many problems for many reasons, ranging from a bad standard to one that is poorly implemented:

- Bureaucratic working practices
- Too much detail to be useful
- Taking focus away from actual software development activities
- Reducing productivity

The next section provides the reader with the Standards AntiPattern that is at the heart of the debate on standards, defining consistent working practices in software development. It is intended to be a tutorial that introduces the nature of an AntiPattern and tackles one of the critical aspects of software development that a project manager must deal with: adopting, deriving, or avoiding a software development standard for people, technology, or process.

AntiPattern

The Standards

AntiPattern Name: The Standards

Also Known As: The Necessary Evil

Most Applicable Scale: System

Refactored Solution Name: Back to Zero

Refactored Solution Type: Process

Root Causes: Ignorance, Sloth, Apathy, Narrow-Mindedness, or Pride

Unbalanced Forces: Management of Complexity

Anecdotal Evidence:

"Standards are bureaucratic and worthless."

"We'll be okay if we just stick to exactly what the standard says!"

BACKGROUND

Standards claim to be the solution for many development projects by solving the problems of quality and standardization, yet seemingly they never live up to the claim. Project managers use standards as tools that enable visibility into the progress and quality of the development effort, and yet developers often resist implementing standards at every turn, citing too much bureaucracy, gold plating, impediments to progress, and unnecessary work.

When projects attempt to implement standards and fail to achieve the desired result, it doesn't take long for management and project managers to begin to question the effectiveness of standards. For many organizations, standards are discounted and considered only valuable for academics, having little *real* value. It is typically those same organizations that fail to meet schedule, cost, and quality requirements on most of their projects. This AntiPattern will explain why standards fail to address the needs of many organizations and provide a refactored solution.

GENERAL FORM

Why don't standards work? Standards do work if the proper standard is selected and is properly implemented. But when standards are improperly implemented or the improper standard is used, it is easier to blame the standard than assign fault to someone on the project. The following list is a summary of the reasons why standards fail to achieve success on projects:

- Standards receive resistance from developers because they restrict the freedoms of developers.
- Conforming to a standard is perceived as time-consuming, while management is perceived to demand time reduction.
- The proper standard isn't selected or tailored for the project.
- The standard isn't properly implemented.
- The organization isn't committed to implementing the standard.
- The perception is that the organization doesn't value standards.

Developer Resistance

There are two primary reasons that standards are difficult to implement from a developer perspective. One reason is because standards restrict

developers in their ability to freely apply "artistic value" to the development effort. The second reason is similar, where the developers feel that they don't require standards. Standards are simply irrelevant, demanding additional time, when time is already at a premium. Developers feel that if they understand the objective, shortcuts will save time with no harm.

In both cases, the developers feel restricted. It is frequently the view of developers that standards limit their freedom [Bennatan 1995]. This resistance makes implementation and its oversight difficult without becoming Machiavellian. The project manager must identify the benefits of standardization and quality to the developers. The standardization or commonality is valuable to a developer's ability to review and understand other developers' code. This is also positive from a project management perspective, because a standard product lessens the value of any one developer: It reduces the truck factor of any one developer. (*Truck factor* refers to the inordinate value of a team member, and when a truck hits that team member, the project is in big trouble.)

Selection and Tailoring of Standards

Another reason that standards don't work is because the wrong standards are selected or standards are improperly tailored for the project. This can be the result of a well-intentioned manager wanting to bring some type of control or quality to the development effort. The manager yanks the first available and seemingly applicable standard off the shelf and throws the standard in the developers' office, telling them to implement. Unfortunately, because the standard wasn't properly understood or tailored to the specific project, some parts of the standard that the developers are forced to implement aren't applicable. The result is that the developers resist the implementation of any part of the standard and, moreover, form the opinion (rightfully so) that the standard requires unnecessary work. Nevertheless, forcing the developers to implement an unnecessary process or the unnecessary requirements of the standard furthers their opinion that all standards are worthless and unnecessary. Whatever the reason, the standards must be properly selected and tailored for the project. Before a standard is selected it is imperative that the requirements for the standard are clearly documented and understood. This ensures that potential standards can be evaluated for their usefulness and fitness for purpose. It is likely that any standard adopted will need to be modified to fit the specific software development culture to be successful.

Implementation of Standards

Assuming the correct standard was properly selected and tailored, it must also be properly implemented to be effective. Often the standard isn't read or understood well enough to be implemented. This can be true of standards that are mandated by the customer. The standard is thrown at the developers and they are expected to correctly implement it without any instruction, previous experience, or consultation from an expert and, like most aspects of software development, experience is all important. Moreover, the word *understanding* should be defined in this context. While it is important for the project team to have thoroughly read the standard, there should be an individual, an expert, who has experience in implementing the standard. Understanding should also refer to standards that are instituted and used throughout the enterprise. They should be supported at the corporate level with the appropriate infrastructure. This includes policies, procedures, tools, and experienced individuals to assist in their implementation. Repeatability is key for the implementation of standards.

One last word about implementation. It is imperative that the development team know in advance what standards are to be implemented. It is not acceptable to have the development team attempt to implement a coding standard once coding has begun. While this should seem all too obvious, it is never beyond the realm of possibility. Imagine a project leader who actually reads the contract and discovers the requirement for the implementation of the standard halfway through the project. All too frequently it's the development team that suffers from this sudden discovery (see Figure 1.3).

Commitment to Standards

One of the most significant factors in implementing a standard is ensuring that the management, project leadership, and development team are committed to implementing the standards. Frequently standards are used as nothing more than a badge, a buzzword, or a check in a box to make the customer happy. Standards require that all members of the organization are committed to successful implementation. Should any one layer in the organization fail to stand by the commitment, the implementation of the standard will likely fail. The implementation of standards requires a great deal of discipline, which is sometimes contrary to the software development environment [Bennatan 1995]. The collateral impact of a failed implementation of a standard is the perception that *all* standards don't work.

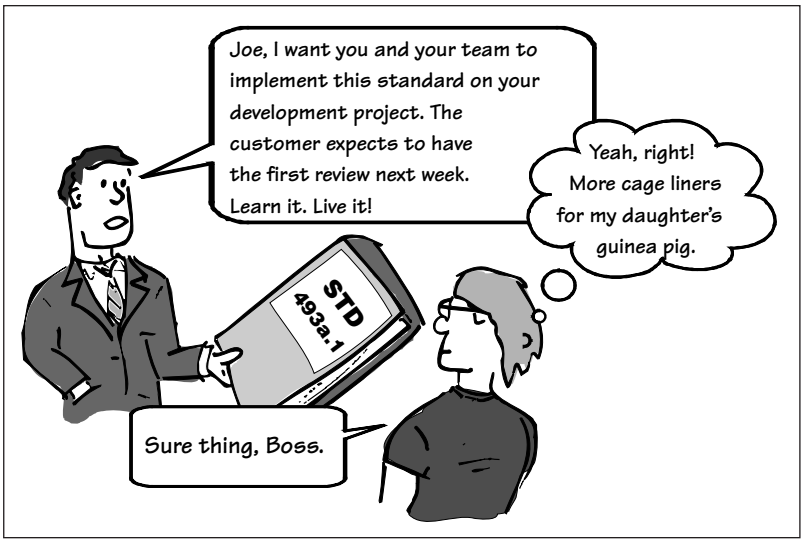


Figure 1.3 Standards toss.

Perception of Standards (Organizational Values)

The power of perception within an organization is substantial and must be recognized appropriately. Some perceptions are formed by the organizational policies of a company, others by the attitude of management. Organizational policies sometimes reveal that standards are not valued. This may be true even when standards are in use or have been used in an organization for a substantial period. The standards are taken for granted without realizing their true value. The cost of implementing a standard is often calculable; however, the value is often intangible.

Of course it is possible that in any given instance the standard is costly and ineffective, but this analysis should be carefully reviewed to determine if it is an accurate assessment. It may be that the wrong standard was selected or improperly implemented, but it may also be that the value of the standard is underestimated because it has been in place for so long that the organization no longer realizes its impact. The value of the standard is taken for granted. Essentially this is the "I can't see the forest for the trees" analysis where a manager requests that someone demonstrate the cost-effectivness of implementing a particular standard. If the individual performing the analysis isn't astute or observant enough to look beyond the obvious, the value of the standard may not be uncovered in the analysis. Specifically, the analysis must

look at those aspects that aren't directly related to the standard and that may result in cost avoidance for the organization.

SYMPTOMS AND CONSEQUENCES

How can you tell if your program is in the midst of suffering from this AntiPattern? What are the symptoms? The symptoms of this AntiPattern are somewhat transparent since they can be the symptoms of other AntiPatterns, poor program management, and the eminent nature of program failure if the AntiPattern progresses too far. However, in the early stages of the AntiPattern, one of the first symptoms is the initial resistance toward the implementation of standards. Other indications that standards aren't implemented or are poorly implemented may be determined if the project manager engages the staff in discussion about how they are implementing the standards. The project manager should ask specific questions and anything less than a convincing answer should be concern for further investigation. Another method to uncover poorly implemented standards is during program milestone reviews or testing evolutions. Should it turn out during a program milestone review or through testing that substantial failures exist, then there is good reason to thoroughly investigate the cause, beginning with the standard used. This type of investigation may turn out to be inconsequential; however, it may turn out that the standard wasn't fully understood or properly implemented. If these direct and obvious symptoms are missed, a continued disintegration will produce final symptoms that look more like the project death spiral (see The Brawl AntiPattern).

The consequences that can result from this AntiPattern should be apparent from the symptoms. First, the standard is not achieved. This primary result is also reflected in the inability to achieve success within the project discipline for which the standard was to be applied (e.g., configuration control, coding practices, testing). The failure to achieve success within a project discipline can often result in collateral impact to other project disciplines, which eventually leads to project failure or, at minimum, failure of some aspect of the project. Varying degrees of failure may result; nevertheless, it has substantial and fundamental impact on the project. Additionally, the impact to the organizational perspective of standards is affected as well, usually for the worst. Failure of a project that has implemented standards leaves the perception that all standards are worthless.

These same consequences are the result of improperly implementing standards or failing to commit to the implementation of a standard. Of course, failure to use standards at all may have similar results but not necessarily. Not using standards may not result in failure. The ability of the

project team, as well as the scope of the project, determines whether a project requires standards at all. The rigor of the project team may overcome the need for standards on a project; however, while immediate project failure is averted, other problems may occur subsequent to deployment. For example, it may be difficult or costly to sustain maintenance of the code if standards weren't used.

TYPICAL CAUSES

Several causes of this AntiPattern have already been discussed. To summarize, the typical causes include:

- Misuse of the standard due to the selection:
  - Hypothetical requirements
  - Badge or checklist mentality of the person performing the selection
  - Requirements for the standard not understood
- Improper selection due to not defining the requirements for the need of a standard
- Improper tailoring due to misunderstanding the purpose of a standard
- Improper implementation due to misunderstanding the culture of the organization that has to use the standard, such as:
  - No management or project team buyin
  - Insufficient funding to implement fully
  - Lazy implementation, resulting in the standard being improperly implemented
  - Inadequate process for implementing the standard, resulting in poor implementation

Misuse of Standards

Misuse of standards is probably the most common cause of the Standards AntiPattern. Misuse stems from project teams viewing standards as a hypothetical requirement only: that standards are used only by academia or government, when in reality standards are frequently used on quality projects by quality organizations.

Misuse of standards also occurs when project teams (with an ample amount of irreverence for a standard) haphazardly attempt to implement the standard, resulting in a less than desirable result, and yet they claim a

rigorous implementation. The same result occurs when the project team is merely using the standard as a checklist by which to achieve a contractual requirement or so that they can claim compliance for marketing purposes. There is no intention on the part of the project team to properly implement the standard: It is simply a checkbox to be marked as complete or a line on a marketing brochure. Worst of all, it is difficult for the project manager to discern this cause of the AntiPattern without investigation. The project team will have the outward appearance of having implemented the standard and will likely look the customer square in the eye and state that they have implemented the standard.

Finally, this cause of the AntiPattern occurs when the standard isn't fully understood by the individuals responsible for implementation. So while the project team diligently attempts to comply and implement the standard to the best of their ability, it is still less than successful. This cause can also be true when the standard isn't properly tailored, again because the individual responsible doesn't fully comprehend or have an appropriate level of experience in implementing this standard.

Improper Selection of Standards

Improper selection of the standard by either the project team or the customer is another cause of the Standards AntiPattern. Understanding what you are attempting to achieve and having knowledge of the standards that are available within that domain are critical to making the proper selection. In some cases it may be that a standard isn't required at all. Discerning the need for the standard should be established first. Then there should be a review of the standards that are available within that domain. The person responsible for selection should have some knowledge of and experience with implementing the standard under similar circumstances.

One of the reasons for improper selection of standards is because there are so many standards organizations, and the standards that result from their efforts make it difficult to discern what is appropriate for your project. Because there are so many related standards it is impossible to become familiar with each one, much less become knowledgeable about all of those that exist (see Figure 1.4).

Improper Tailoring of Standards

Next on the list for causes of the Standards AntiPattern is improper tailoring. It must be understood that standards are written to address a wide

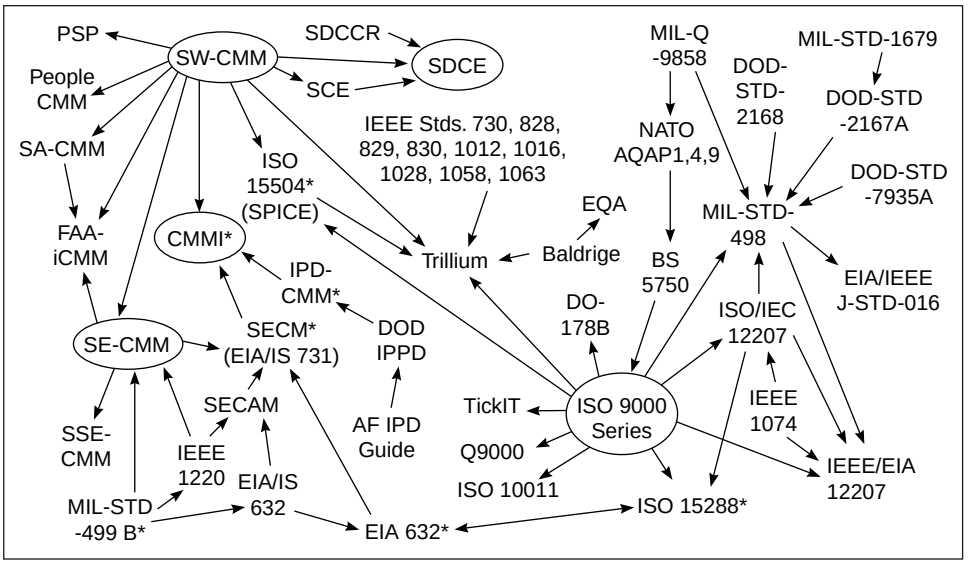


Figure 1.4 Frameworks quagmire.

Source: Drawing by Sara Sheard © 1998 Software Productivity Consortium NFP. Used with permission.

range of environments and circumstances. Without a comprehension of the project environment and the project goals as well as a familiarity with the standard being implemented, it may be difficult to discern which sections of the standard are appropriate for this implementation. It is possible that even the right standard may result in the wrong implementation. Here again, this is where experience comes into play. Having implemented a standard many times under varying circumstances provides a knowledge base otherwise unachievable by just reading the standard. This experience will enable the domain expert to know which sections of the standard are critical and which can be laid aside.

Improper Implementation of Standards

The final cause for the Standards AntiPattern is improper implementation of the standard. Specifically, gaining buyin from both the management and the project team is critical to achieving success in implementing standards, as illustrated in Figure 1.5. Unfortunately, management doesn't always understand that commitment typically translates into additional funding, skilled personnel, and time (schedule) to achieve success. However, no matter how many resources management "throws" at the implementation, if the project manager and the team aren't dedicated to achieving success, the standards may just as well be toilet paper. Finally, there must be a process

for reviewing and ensuring that the standards are being implemented properly. If the project manager doesn't initiate reviews that provide insight into the implementation of the standards and are appropriate to the disciplines, there is no need to bother implementing the standards.

KNOWN EXCEPTIONS

The exception to the Standards AntiPattern is standards that are inappropriate for use by the project. These projects are typically smaller in scope and complexity and demand little rigor in the implementation and maintenance. However, the presumption that a standard is not required may precipitate the AntiPattern. Choosing when and when not to implement standards is difficult at best and while in some instances the lack of standards may be overcome by project team finessing, it usually comes with the appropriate amount of problems described by this AntiPattern and others.

REFACTORED SOLUTION

The refactored solution for the Standards AntiPattern is to return to zero, that is, the beginning. This is true whether you are at the beginning of the AntiPattern or smack in the middle. The only difference may be that there

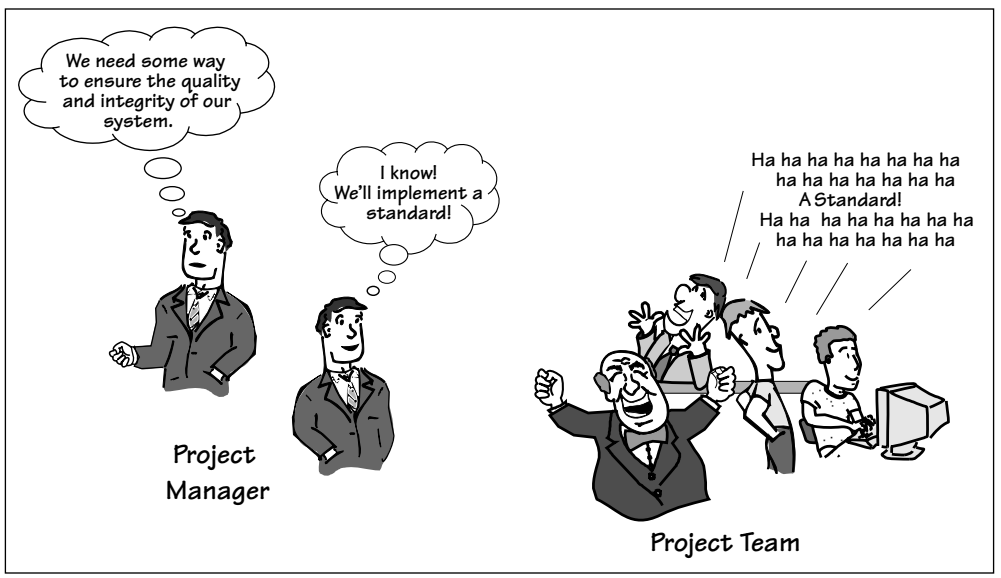


Figure 1.5 Standards buy-in.

needs to be some assessment of how salvageable the deliverables are, for each project discipline, before "trashing" begins. For example, code should not be trashed in its entirety. Rather there should be a determination of what is required in order to have the code achieve the standard, and based on that evaluation it should be determined if the code is salvageable. However, that can only occur subsequent to determining if the appropriate standard was selected, tailored, and properly implemented.

Therefore, notwithstanding the assessment of the status of each project discipline, the first step is to determine if you have sufficient in-house expertise to follow through this process. This is somewhat of a catch-22 since you don't know what you don't know. Nevertheless, the project manager should evaluate the abilities of the individuals responsible for implementing the various standards and determine if they are knowledgeable and capable. If they are deemed acceptable, then proceed; otherwise you will probably need a consultant who is an expert in the project disciplines you are seeking to address. If you choose a consultant, then he or she should lead you through the remainder of the refactored solution presented here.

The next step is to determine if you need standards at all. This determination may be simple and straightforward if there are contractual requirements or a corporate policy, or if you are attempting to achieve a status of compliance (e.g., ISO). Otherwise, the determination is made using criteria that describe the magnitude of the effort and the complexity, reliability, and maintainability of the software. Software that requires a high degree of reliability (e.g., software for controlling nuclear operations) must be developed in an environment that demands a high degree of reliability and confidence through standards. Chances are, if you are developing that type of software you are required to follow specific standards very closely, with a high degree of review and oversight. Beyond high reliability, project complexity as well as maintenance requirements should be evaluated to determine whether standards are useful. The project manager should consult with the individuals responsible for implementing the standards in each of the project disciplines and seek their advice with regards to the implementation of standards. A note of caution must be added since those involved with the development effort often lose sight of the maintenance aspect of software. Determining the need for standards must be accomplished with foresight and with regard to a lifecycle perspective. The project manager may be surprised to learn, after talking with the project domain experts, that standards are already implemented, in which case it must be determined if they are the proper standards and if they have been appropriately implemented.

Determining which standards are appropriate is probably the single most important step in the process. James Moore's book, *Software Engineering*

Standards: A User's Roadmap [Moore 1998], is extremely helpful in comprehending the relationship between standards and your project and determining which is appropriate for your project (see standards sidebar). Aside from contractual, corporate, or compliance, the best bet is to go with IEEE or ISO standards for software development. These standards are widely accepted and are the de facto standards throughout the community.

Moore categorizes standards by these bodies using the following three models [Moore 1998]:

- Context model
- Layered model
- Object model

The context model shown in Figure 1.6 displays the relationships between software engineering and other related disciplines. A layered model (see Figure 1.7) is essentially a hierarchical view of the standards, with the more general standards at the top and more specific standards at the bottom. And finally the object model, Figure 1.8, demonstrates the relationships between key objects. Moore uses these models to map between the standards that adhere to these three models (see Figure 1.9).

SOFTWARE ENGINEERING STANDARDS ROADMAP

Understanding standards and their organization is key to selecting appropriate standards that are beneficial to your software development. The first step for the project manager is to understand that there are various ways in which standards can be organized. One method for categorization is based on the various bodies that issue standards. Some of the more prominent bodies include:

- **DOD-STD (Department of Defense Standards)**
  MIL-HDBK (Military Handbooks)
  MIL-SPEC (Military Specifications)
  MIL-STD (Military Standard)
- **IEEE (Institute of Electrical and Electronic Engineers)**
- **ISO (International Standards Organization)**
- **STANAG (Standardization Agreement [NATO])**
- **TIA/EIA (Telecommunications Industry/Electronic Industries Association)**

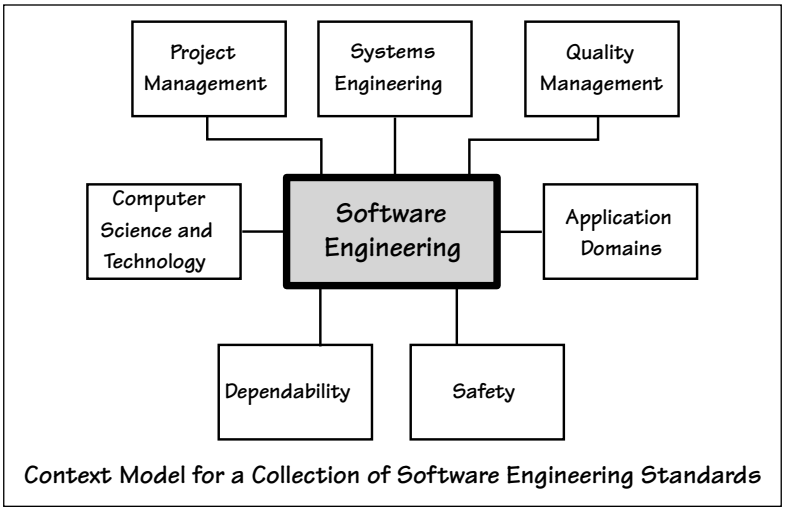


Figure 1.6 Context model.
Source: Drawing by J. Moore © 1998 IEEE. Used with permission.

Moore's book provides a great organization and mapping of the standards. An example of the kind of breakout you can expect is provided in Figure 1.10, which is an overview of the Software Engineering Standards Committee of IEEE (SESC) collection of standards.

Additionally, individual standards are broken down as well. A good example of how the book maps specific standards is demonstrated in the breakdown of the supporting processes of IEEE/EIA 12207.0 (see Figures 1.11, 1.12, 1.13). Moore's book contains 37 figures, 98 tables, and a catalog of 199 standards [Moore 1998]. It should be on the bookshelf of every project manager.

Subsequent to choosing the standards for implementation is tailoring the standards to the specific project. The tailored standard should be commensurate with the project objective (see Figure 1.14). This also applies to the rigor with which it is implemented. Rigor reflects the level of detail, as well as the review and audit that takes place, to ensure that the standards are appropriately implemented. This is a subjective effort and requires that the individual responsible be cognizant and knowledgeable of not only the standard, but also the objectives of the project, to determine what is appropriate for the project.

After successfully choosing and tailoring the standards, the project manager must commit to ensuring a successful implementation. This means that the appropriate funding and staffing are applied to ensure that the standards can be executed. The project manager should be able to identify to his

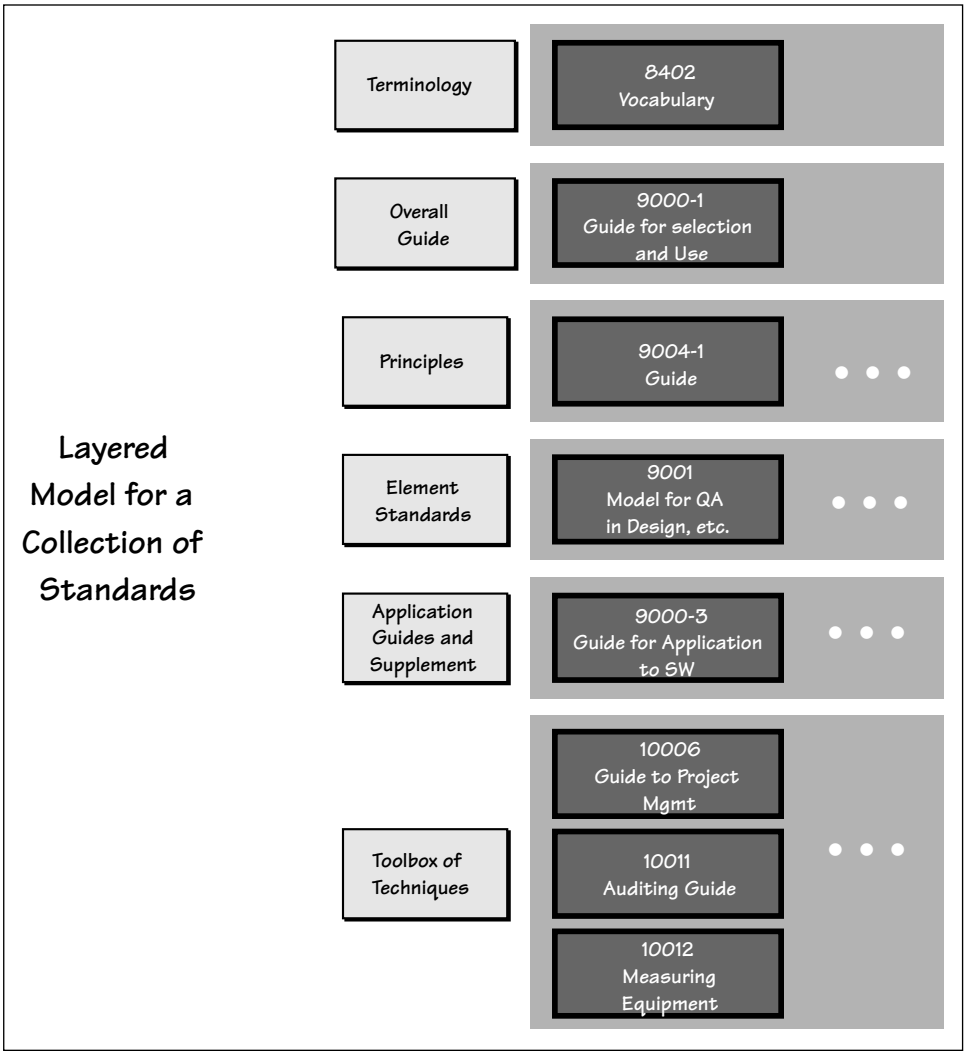


Figure 1.7 Layered model.
Source: Drawing by J. Moore © 1998 IEEE. Used with permission.

or her management what specific and tangible benefits will be achieved through the implementation of the standards. Metrics should be developed and acquired to ensure that the benefits are being derived. If they are not, an analysis should be undertaken to find out why. It may turn out that the standards aren't properly implemented because the project team hasn't been properly trained or made aware of the practices necessary to accomplish the standard. Direct benefits should be identified to management in briefings and reviews as appropriate. The customer or management should have clear insight into what they are getting for their money.

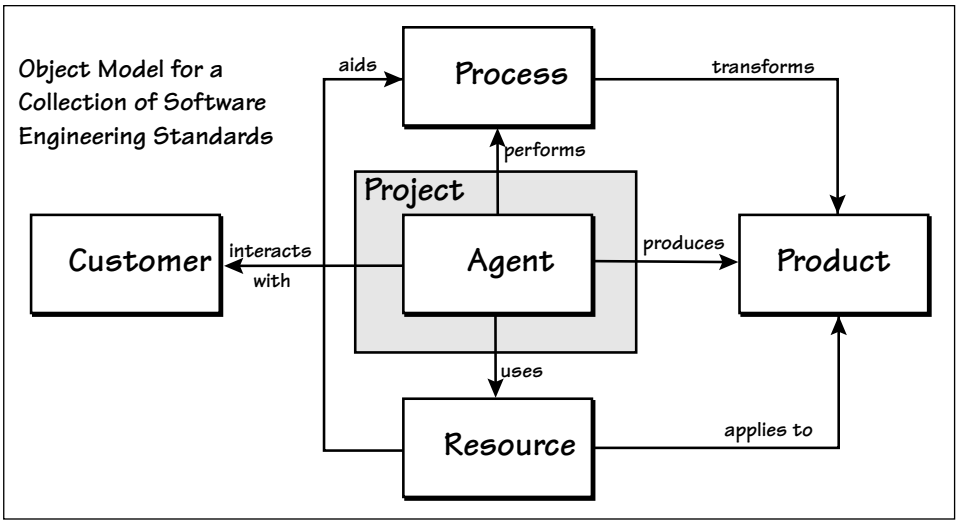


Figure 1.8 Object model.
Source: Drawing by J. Moore © 1998 IEEE. Used with permission.

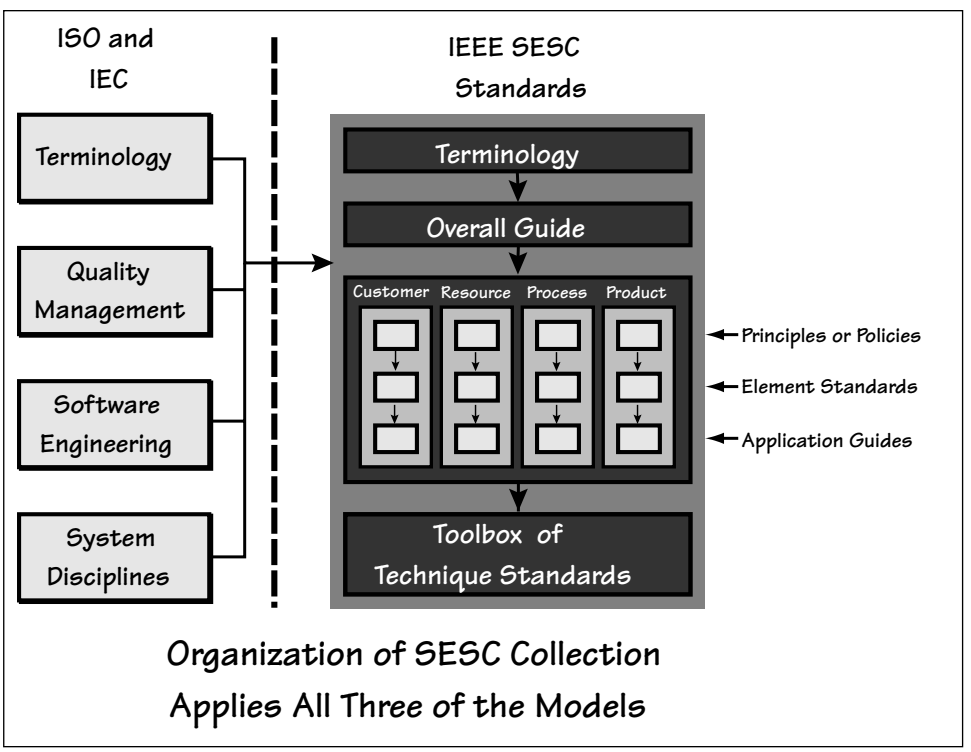


Figure 1.9 Mapping of models.
Source: Drawing by J. Moore © 1998 IEEE. Used with permission.

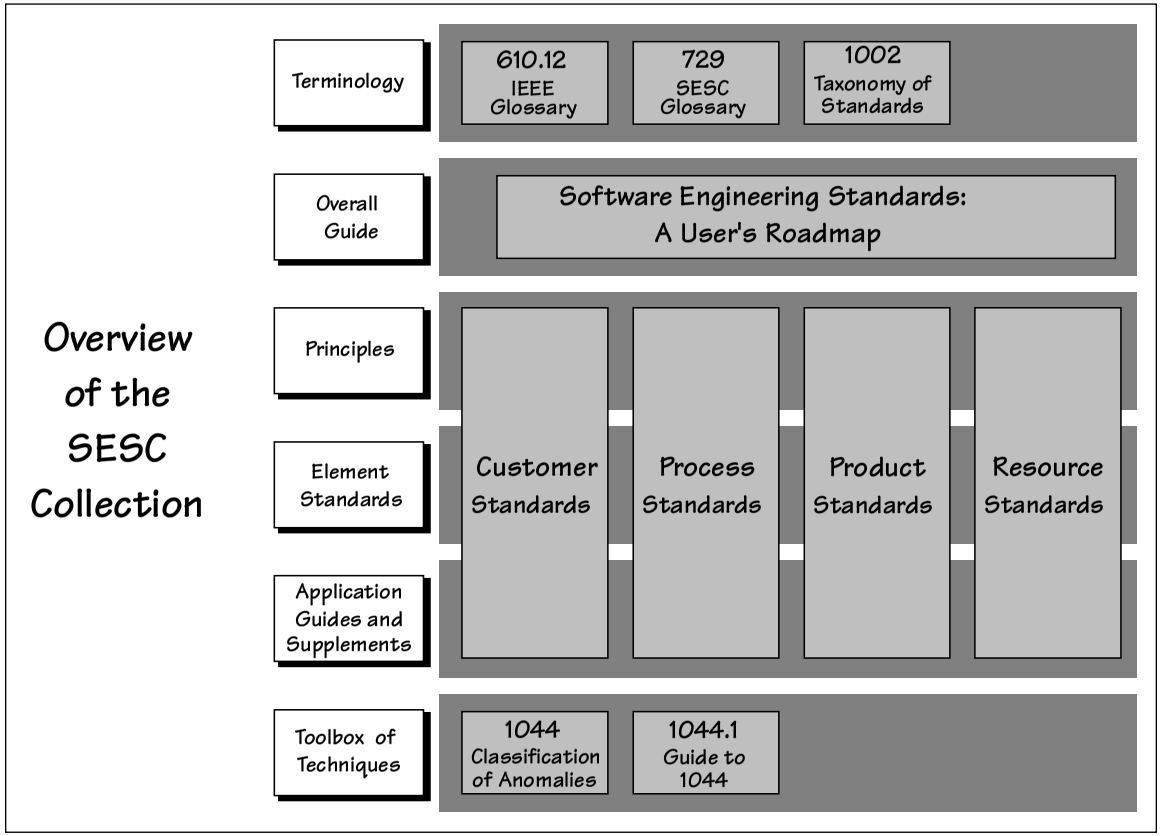


Figure 1.10 Overview of SESC collection.

Source: Drawing by J. Moore © 1998 IEEE. Used with permission.

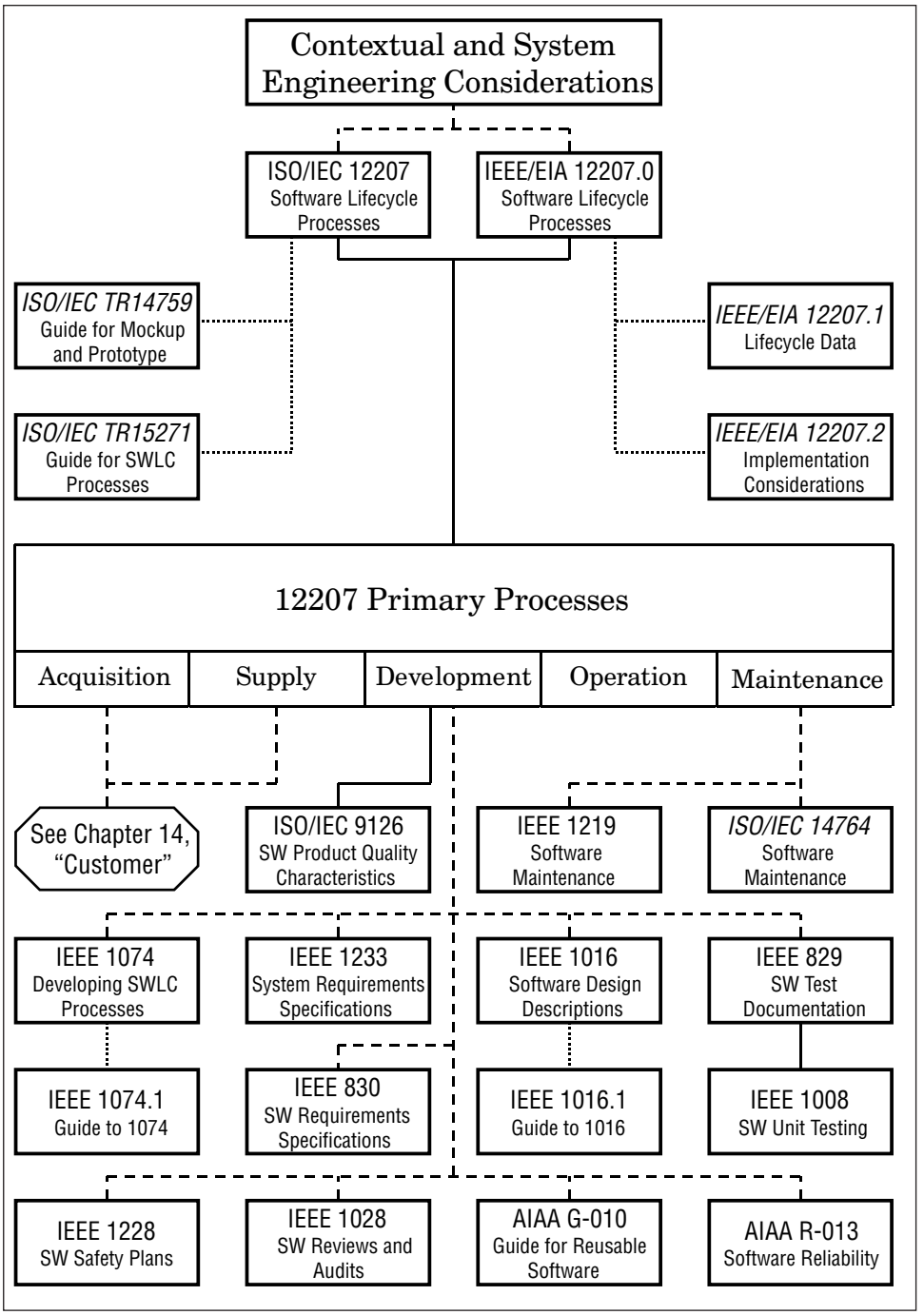


Figure 1.11 IEEE 12207.0 primary processes.

Source: Drawing by J. Moore. © 1998 IEEE. Used with permission.

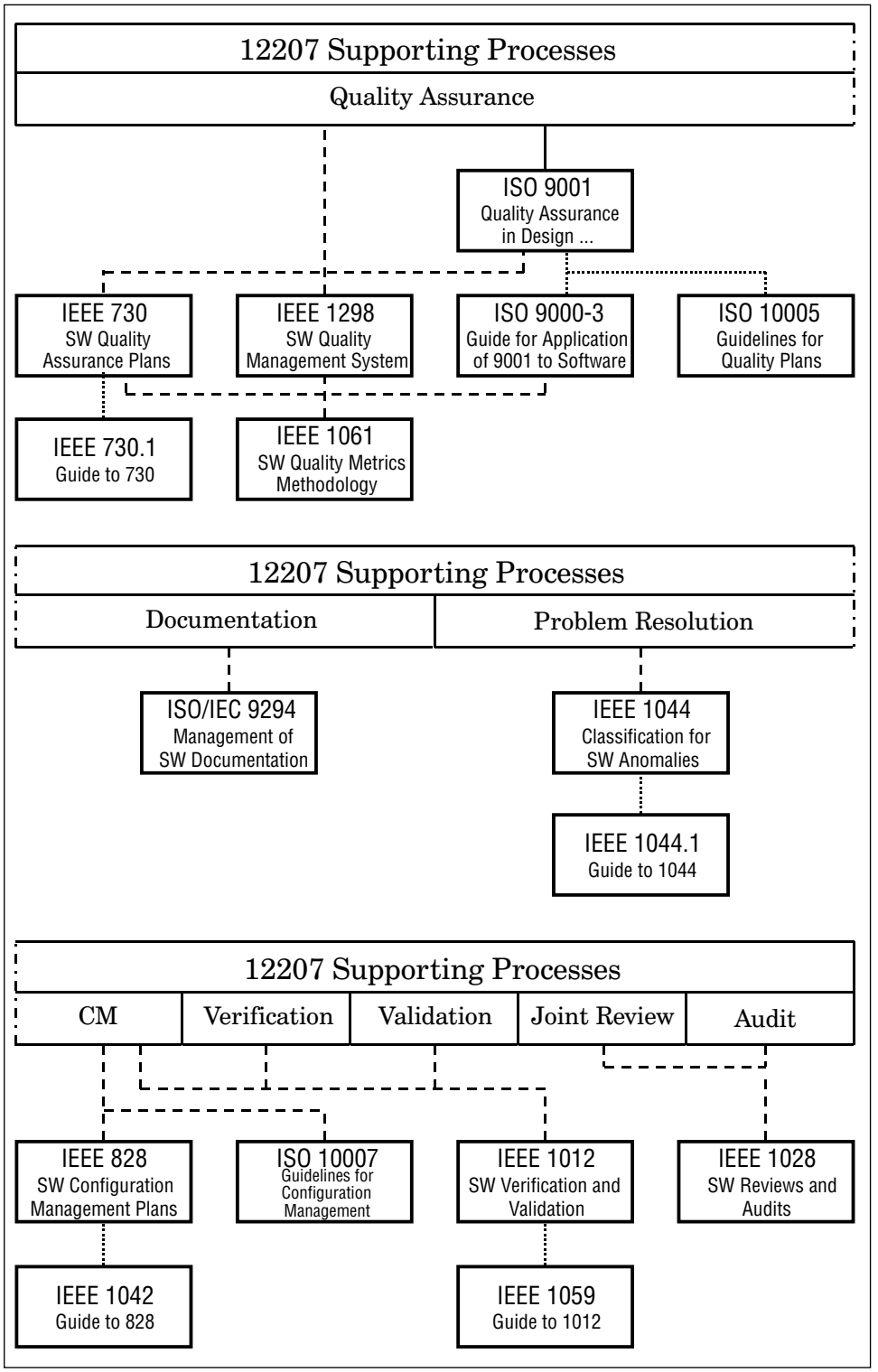


Figure 1.12 IEEE 12207.0 supporting processes.

Source: Drawing by J. Moore. © 1998 IEEE. Used with permission.

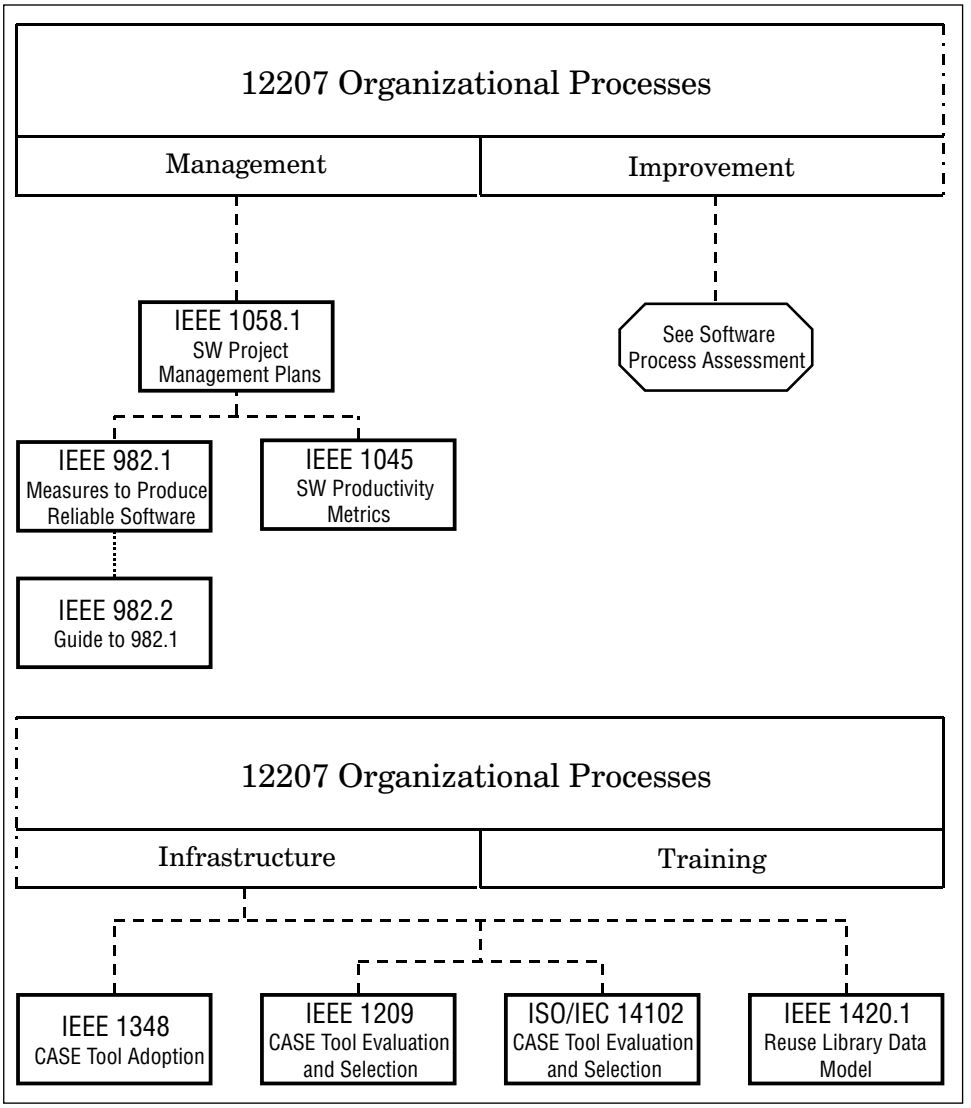
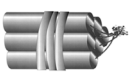


Figure 1.13 IEEE 12207.0 organizational processes.
Source: Drawing by J. Moore. © 1998 IEEE. Used with permission.



VARIATIONS

One variation to the Standards AntiPattern is when so-called experts begin to talk about implementation; you can typically find some deviation in the interpretation of what was intended by the standard. The project manager should be very careful to discern which implementation (read that interpre-

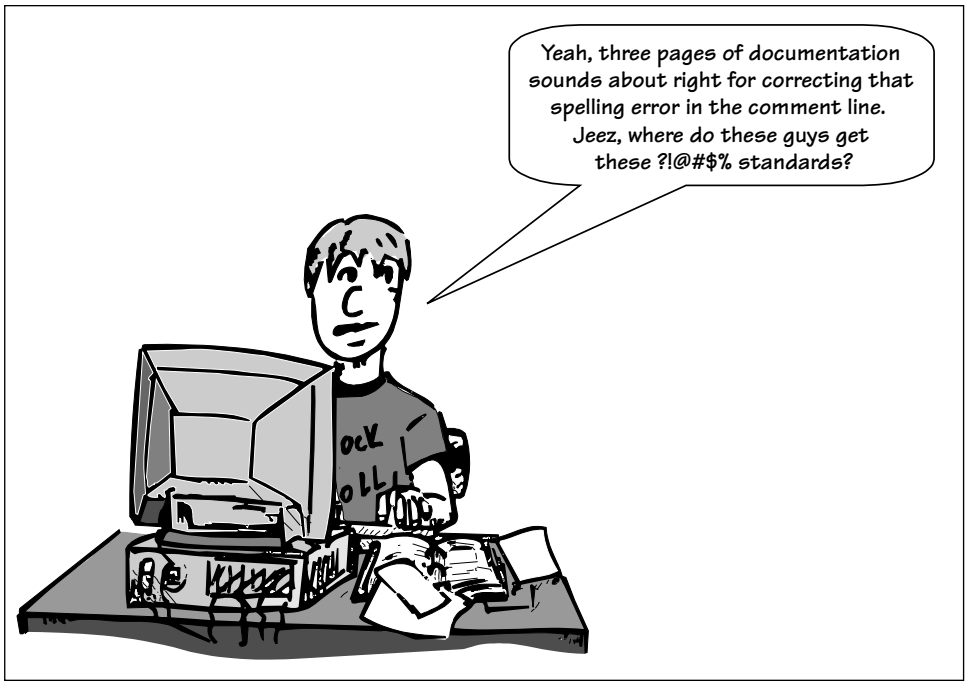


Figure 1.14 Standards commensurate with the project objective.

tation) of the standard is appropriate for the project. The safest bet is to go with the more rigorous implementation.

You must also beware of requiring standards when they are not required. Experts will argue that they are required when the scope and magnitude of the effort don't dictate that type of implementation. The expert is also likely to recommend the standard that she or he is most familiar with, rather than the most appropriate.

EXAMPLE

This is an example of how poorly standards can be used in many ways. It is a story of the development of a configuration management (CM) plan that was initially developed without direction or instruction from the project manager as to what standard to use in developing the plan. It is also the story of how the project manager failed to convey the importance of the plan. A well-developed plan based on IEEE standards was tossed aside for yet another plan that was based on no standard at all. There was no project management buyin and project management didn't value standards in general. This is an example of everyday life in a project.

The project was a software development effort that attempted to develop an initial baseline and then replicate that baseline with slight deviations for several related systems. While reuse of the initial code was expected to be high, it was anticipated that each subsequent baseline would have to be configured and managed separately. Initially a configuration management (CM) expert was called on to write a software configuration management plan for the control and management of the software baseline. The CM plan was to be written against the standards that made sense: DOD, IEEE, or whatever standards could be "pulled off" the Web and coordinated within the context of the project and the organization. The initial result by the CM expert was a mishmash of cut and paste from various CM standards, making little or no sense. After several more attempts by the expert to improve the standard (which resulted in the removal of meaningless boilerplate), the plan was promptly placed in the circular file with all other relevant material. Subsequently, a discussion ensued that identified the real problem. The expert assumed that the plan was merely for completing a contractual requirement rather than actually attempting to create a methodology from which a complex software baseline and its variations could be managed. The expert explained that he really had little time to participate in the project and was trying to help the project leader by putting together something that appeared good enough for completing the contractual requirement.

The story doesn't end there. A real software engineer was then tasked to develop the software configuration management plan. The program manager sat down with the engineer and talked at great length about the purpose and the importance of the plan, as well as the need to anchor the plan with a specific standard that was tailored for the project. The engineer undertook the task of utilizing the relevant IEEE standard for software configuration management. His plan was to develop a single plan for the organic baseline and have another plan for how each of the variations would be managed. This was briefed to the program manager and agreed to by all parties. The plans were developed in great detail with rigor and quality. The plans were submitted for review; however, due to other priorities, the plans never left the inbox. In the meantime the project hired a configuration manager and the software engineer was placed back on the development team. The plans developed by the software engineer were turned over to the newly hired configuration manager.

The newly hired configuration manager was from the old school and had her ways of approaching CM. Her approach was limited to the tool that she was taught to use when she joined the corporation. She had a fundamental understanding of how CM worked, especially in the context of the tool she planned to used. The high-quality, standards-based software configuration management plans developed by the software engineer were promptly

placed in the circular file with all other relevant material. The configuration manager pulled an old CM plan from a previous project and proceeded to replace all appropriate information, such that the previous project name was replaced with the new project name. The new CM plan was based on a de facto standard of how the configuration manager operated.

RELATED SOLUTIONS

There are several related solutions to this AntiPattern through the implementation of refactored solutions to other AntiPatterns, such as The Blob, Lava Flow, Golden Hammer, Spaghetti Code, and Cut and Paste Programming AntiPatterns [Brown 1998], that are examples of bad design or development practices that could be avoided by the implementation of standards. Additionally, the Software Configuration Management Expert AntiPattern is a good example of failing to utilize a standard to control the project disciplines. The Failure to Audit AntiPattern is a good example of failing to adhere to the requirements implemented by a standard. The key is to discern which AntiPattern is applicable and apply the refactored solution in combination with the refactored solution presented within this AntiPattern.

APPLICABILITY TO OTHER VIEWPOINTS AND SCALES

This AntiPattern is applicable to the enterprise perspective at nearly every level of the organization. The impact on developers expected to implement a design that may be ambiguous and highly complex is stressful to say the least. Substantial project risk is incurred, and that is reflected in the protracted schedule and increased costs. Perhaps worst of all is the cyclical effect of not implementing standards because they weren't properly implemented, resulting in the Standards AntiPattern.