One way to manage change is to harness its momentum and direct it in desired directions. An appropriate analogy is a canoeist trying to negotiate his or her canoe down rapids. Ninety-five percent of the canoe’s downstream progress is determined by the force of water rushing around boulders and over falls. Perhaps 5 percent is directed by the canoeist, who by means of controlled paddling can move the canoe laterally, allowing it to proceed along one channel rather than another and enabling it to avoid collisions with rocks. Attempts to go against the powerful current lead to frustration and failure. Success is based on controlling while going with the flow.

Rapid prototyping is a methodology that arose in the 1980s that manages change by going with the flow. Its basic premise is that in a rapidly changing world, it is often impossible to prespecify requirements precisely. Even when possible, it may be undesirable to do so.

Proponents of rapid prototyping recognize that a major problem of project staff in specifying requirements is that the customers they attempt to serve don’t know their needs or wants. Furthermore, customers are incapable of determining whether the requirements presented to them by project staff early in the project life cycle truly represent their interests because at this stage these interests are rather abstract and difficult to visualize. Consequently, as a deliverable gradually emerges and they see what they are actually getting, they begin demanding changes to the requirements. If they like what they see, they may very well request enhancements to the evolving deliverable. (“What you’re producing is fantastic! Now, can you add the following bells and whistles to make it superlative?”) If they don’t like what they see, they will request changes to make it right. The point is: there will be change no matter what.

The rapid prototyping perspective sees this phenomenon not as a problem but rather as an opportunity. Why not obtain customers’ involvement in developing requirements by having them react to prototypes that are presented to them from time to time? Because the prototypes represent a tangible deliverable, customers can respond to them more meaningfully than they can to abstract statements.

The Rapid Prototyping Procedure

The first steps in the process represent classic systems analysis. Project staff interview customers to identify their needs and wants. They then try to determine whether these needs and wants can be satisfied in view of budget, schedule, and technical constraints. They also review how these needs and wants are being addressed by current procedures. As a result of this process, they formulate their view of what the customer requirements should be.

With classic systems development, these requirements serve as the basis for developing a new system. The requirements are used to form a design. Then the design is employed to actually build a deliverable. For this approach to result in a deliverable that meets customer needs and wants, the original requirements must be on target. The problem is that they often are not.
With rapid prototyping, the requirements emerging from systems analysis are viewed as just the first step in the development of customer-focused requirements. They are used to build a prototype so that customers can see what they will get. In software development, the prototype vehicle is computer screen images. For example, in a project to develop a simple management information system (MIS), three sets of prototypes might be used. One would be screen images illustrating data entry forms. Another would be screen images of data retrieval forms. The third would be help menus.

When the first prototype is ready, the development team安排s to meet with a panel of customers. The composition of this panel is crucial to the success of the project. Panel members must accurately reflect the interests of the full range of customers. For example, the panel may reflect the perspective of data entry clerks, data retrieval personnel, and a set of the people who use the data to help them in their decision making. If the wrong panel members are chosen, the resulting requirements will not be relevant to customer needs and wants.

The first meeting between customers and developers is a classic kickoff meeting. Developers introduce themselves and the prototyping process. They show the customer panel the prototype they have developed. At this point, the prototype is nothing more than screen images. There is no depth behind them. Still, they serve a valuable function because customers see what they will get. At this kickoff meeting, they are generally delighted to find that they are working with something that has the “look and feel” of the final product. They may immediately begin offering suggestions for improving the prototype.

After the first meeting, developers go off for several weeks and begin adding muscle and sinews to the prototype skeleton. They are careful not to add too much detail. (This process is called top-down design.) Their objective at this point is to produce as quickly as possible a prototype that customers can work with to give them an accurate sense of what they are getting. Once this revised prototype is ready, they hand it to the panel of customers for their review. The panel then “exercises” the prototype. For example, if it is well developed, they may be able to enter sample data into the data entry forms: Because the prototype is just a shell at this point, it is incapable of doing anything meaningful with the entered data. Nonetheless, this data entry exercise is important because it may bring to the surface issues that might otherwise be ignored ("Look, the employee identification number is too short. Two more digits should be added to it.")

On small systems, the prototype-exercising process can be undertaken very quickly—in a matter of hours. On larger systems, it will consume substantially more time.

As a consequence of hands-on experience with the emerging product, the customer panel is prepared to offer useful guidance to the development team on further development of the product. Once the prototype exercise process is complete, the customer panel meets with the developers to present their reactions. In our MIS example, they may find that the design of the data retrieval forms is a bit cumbersome. Some data fields can be eliminated, others added. They may also complain that the help menus use arcane terminology that they do not understand. The development team assesses these comments and determines which can be accommodated and which cannot. They then go off and add more detail to the still skimpy prototype.
The whole process is repeated over and over again until a satisfactory product emerges.

In the early days of prototyping, there was a great deal of concern that refinements of the prototype could go on indefinitely. Experience with prototyping suggests that this is not really a problem. Three very simple rules have emerged for the process. First, the process can cease when a target date has been reached. Second, it can stop when budget has been fully expended. Third, it can stop when there is general agreement between the customer panel and developers that enough has been done. Actually, with prototyping you can stop the process at almost any time and still have viable requirements. (Remember: the process began with a set of requirements established through conventional systems analysis.)

At this point in the cycle, rapid prototyping can travel down one of two roads. With small, simple projects, it is possible that the prototype has gradually evolved into a usable product. If this is the case, the development team may hand over the prototype to the customers and end the whole process. Rapid-prototyping purists do not like this approach. They argue that the product that emerges through the prototyping process is poorly designed since it was put together in a piecemeal fashion. The chief problem with poorly designed systems is that they are difficult to maintain. When they break, they are hard to fix. When enhancements are desired, they may be impossible to develop.

The second road that rapid prototyping can take is called the “throwaway model” approach. With this approach, the prototype that emerges from the process is given to the development group’s requirements experts. They study it to see what features the customers find valuable. As a consequence of this study, they write up detailed requirements that are ultimately used as the core of a highly disciplined development process. Once the detailed requirements have been created, the prototype is thrown away.

**Strengths of Rapid Prototyping**

A key advantage of rapid prototyping is that it leads to unprecedented levels of customer acceptance of deliverables. High customer acceptance occurs because customers are actively and meaningfully involved in defining their requirements. The serious problem of setting unrealistic customer expectations—a common problem in project management—disappears with rapid prototyping since customers see exactly what they will get. Another familiar difficulty—developers who ignore customer sensibilities—fades away because developers are forced to listen to what customers have to say.

Some of the greatest benefits of rapid prototyping are tied to the customers’ exercising of the prototype. This effort has a number of salubrious effects. First, it entails constant testing of the evolving product. It exposes bugs early and offers developers an opportunity to deal with them while they are still manageable.

Second, it gets customers actively involved in developing requirements. They now become part of the development team. In fact, rapid prototyping is a customer partnering methodology. To the extent that their suggestions are incorporated into the final requirements, customers are committed to living with the solutions the developer-customer team offers.
Third, exercising the prototype is a form of training. In order to exercise the prototype, customers must learn how to work the system. The more they exercise the prototype, the more comfortable they will be in using the final product.

**Pitfalls of Rapid Prototyping**

Although rapid prototyping has led to astonishing levels of customer satisfaction, it is not a panacea. Over the last few years, I have interviewed customers and developers associated with more than fifty prototyped projects in order to learn of their experiences. The people I interviewed—both customers and developers—overwhelmingly supported the prototyping process. However, they also warned me of some of the pitfalls they encountered. Following are the pitfalls they described.

**The Development Team Is Not Equipped to Deal with Customers Effectively.** The most serious complaints I heard about rapid prototyping center on the toll it takes on the development team. “It’s killing my technical people,” complained the director of data processing of a large international organization. He explained that his technical staff didn’t possess the training nor the inclination to deal effectively with customers. They did not become programmers and analysts because they enjoyed dealing with people. So when they were directly exposed to customers they encountered a number of frustrations. For example, their customers had difficulty distinguishing between prototypes and the real thing. As the prototype became more sophisticated, customers didn’t understand why the developers didn’t simply turn it over to them as the final product. They did not fully appreciate that the prototypes they were exercising were not real.

A number of approaches have been implemented to deal with this problem. One is to appoint someone to the development team who has both people skills and knowledge of the technology. This individual serves as the main point of contact between the customer panel and the development team, playing the role of buffer. The problem inherent in this solution is that an additional communications layer has been placed between customers and developers, increasing the likelihood of some measure of miscommunication.

A second approach is to put one or two educated customers on the development team. If they grasp the technological issues involved in the development process, they can interact productively with the development team members. Two potential problems can arise with this approach: (1) the customers on the development team can start changing requirements at whim, aggravating difficulties of rubber baselines—I call this the Trojan Horse effect; (2) the customers can become captivated by the technology and lose touch with their customer base so that after a while, they no longer represent customer interests.

A third approach is to improve the human relations skills of the members of the development team through training.

**The Prototype Platform Is Different from the Platform on Which the System Will Actually Run.** The single most frequent piece of advice I received from system developers during my interviews was: never prototype a system on a platform that is different from the platform on
which the built system will function. I heard many horror stories of teams that developed prototypes of mainframe systems on personal computers, employing user-friendly fourth-generation language report generators (4GLs) as the prototyping vehicle. Customers became accustomed to working with prototypes that were flexible, colorful, and easy to adjust. Upon delivery of the real system, customers were horrified to see that it had the cumbersomeness of a mainframe system and used monochrome monitors. Even the keyboard was different from what they used in exercising the prototypes. Needless to say, they vented their frustrations upon the development team.

**The Prototyping Process Lacks Discipline.** A key goal of prototyping is customer involvement in developing requirements in a dynamic environment. For this to work properly, prototypes must be built as quickly as possible. Creative perspectives are emphasized. In all the excitement, it is not surprising that rapid prototyping efforts frequently lack the discipline of documentation and effective change control. Some consider rapid prototyping as all heart and no discipline.

Discipline must be built into the prototyping effort. Rules must be established that each round of the prototyping process be fully documented. Key decisions and actions should be written down. In addition, changes should not be made in a haphazard, free-flow fashion. A change control board made up of representatives of key stakeholders might be established to review change requests that have a measurable impact on schedule, budget, and specifications.

**Resisting Change with Configuration Management**

Just as rapid prototyping is all heart and no discipline, configuration management (CM) is all discipline and no heart. CM is a methodology whose chief tenet is to treat specifications like a contract. Customers should get nothing more or less than a deliverable that meets the specs. No deviations from the specs are accepted unless changes to the specs have gone through a rigorous screening process and have been approved by the proper authorities. Such a contractual approach protects customers from developers and implementers digressing from the specs. It also protects project staff from whimsical customer changes to the requirements.

Note that the underlying philosophy of such an approach is to satisfy the specifications (that is, the contract), not to satisfy the customer. This may seem out of touch with the current focus on customer satisfaction, but it really is not. The key to making CM customer focused is to make sure that the specifications truly respond to customer needs and wants.

CM’s origins date to the U.S. defense contracting community in the 1950s. At that time, it became obvious that building weapons systems had become too complex to be done in a traditional ad hoc fashion. In particular, a consensus emerged that all changes to a complex system should be fully documented and tracked by a sophisticated tracking system. Without proper documentation of changes, it becomes nearly impossible to fix or enhance complex systems. CM was the proposed solution to this problem. Beginning in the mid 1950s, builders of complex defense systems were required to employ CM to document and track changes on all their larger projects. Today, employment of CM has gone beyond hardware development to software projects as well. A well-known software variant of CM is called version control.
Basic Steps in Developing a System with Configuration Management

In developing or modifying a system with CM, every attempt is made to resist incidental change. What customers sign off on is what customers get—not an iota more or an iota less. The following development process allows CM to minimize specious change.

Step 1: Develop Detailed Specifications

With CM, the development process begins with the creation of detailed specifications. Traditionally, these specifications would be generated through classic systems analytical procedures. That is, systems analysts would go out with clipboard in hand, review existing technologies and procedures, interview key people, identify future needs, and then develop system specifications. The problem with this approach is that it often leads to the generation of specifications that are not truly responsive to customer needs and wants. Today, customer-focused specifications are increasingly being generated by means of rapid prototyping.

Once the detailed specifications have been created, then they must be approved by pertinent authorities in both the customer and developer organizations. After they have signed off on the specs, the specs become a baseline.

Step 2: Develop a General Design

Guidance on developing the general design comes from the baseline (that is, the specifications) and only the baseline. As the general design takes form, it is frequently tested against the baseline for traceability. That is, every element of the general design must be tied to a specification. A forward trace starts with a specification and attempts to find a corresponding general design element. If no such element exists, it must be added. A backward trace starts with a general design element and attempts to find a corresponding specification. If none is found, the general design element is eliminated since it represents an addition to the design that is out of line with the specifications.

After a satisfactory general design has been developed and approved by pertinent authorities in both the customer and developer organizations, it becomes the new baseline. The specifications can be put away for a while. This is not a problem since they are actually embedded in the general design.

Step 3: Develop a Detailed Design

The detailed design is built according to the new baseline (that is, the general design) and only the baseline. As in Step 2, care is taken to maintain traceability. When the detailed design is nearly finished, it may be subjected to a functional configuration audit. That is, independent experts may be asked to review the design and to offer their opinion as to whether the system that emerges from it will function in the prescribed fashion. After the detailed design has been approved by pertinent authorities in both the customer and developer organizations, it becomes the new baseline.
**Step 4: Build and Test the System**

The system is built in accordance with the last baseline (that is, the detailed design). As it takes form, it should be tested periodically against the latest version of approved specifications. When the system is fully built, it may be subjected to a physical configuration audit, a full-scale test to ascertain whether the actual system behaves properly in accordance with the specifications. At the end of this step, the development effort is complete.

**Change Control**

Change on projects is inevitable. Even as the development process proceeds in a disciplined way, change will occur and configuration management must be prepared to deal with it. For example, during the general design stage it may become obvious that some of the specifications defined earlier are not realistic and must be modified. Or during the building stage the project team may find that an important component needed in the emerging deliverable is no longer produced and that a substitute must be found.

Configuration management deals with change through careful screening of change requests, meticulous documentation, and controlled updates incorporating changes. Each of these elements of change control in CM will be discussed briefly.

**Screening of Change Requests**

Screening of change requests focuses on determining which change requests have merit and which do not. The process generally begins when someone (for example, a customer, a manager, or a member of the technical staff) submits a change request to the project manager on a form. Different organizations have different names for these forms: request-for-change forms, mods (modification requests), or engineering change proposals (the term commonly used on defense projects).

Upon receiving the change request, the project manager must make an important classification decision. Is this a Category A change (one with a major impact on schedule, budget, or quality) or a Category B change (a low-impact change)? If it is a Category B change, the project manager may make a decision on the spot as to whether the change should be effected. If it is a Category A change, the screening process will be more deliberate. The change request will be turned over to a change control board (CCB) for careful review.

The CCB is primarily interested in the management impacts of a change. In reviewing a change request, the board members want to know its effect on the project budget, schedule, and specifications. Only after conducting a managerial review of this sort is the CCB prepared to weigh the costs against the benefits of a change.

The CCB is typically made up of a small number of people representing different stakeholders in the organization. Ideally, representation comes from technical, financial, marketing, and production groups. With such an interdisciplinary composition, the team is unlikely to view change requests from an overly narrow perspective.
Many organizations subject change requests to technical scrutiny as well as managerial scrutiny. To do this, they establish an engineering review board (ERB) that operates in parallel with the CCB. The chief objective of the ERB is to determine whether change requests have technical merit. In organizations that have both a CCB and an ERB, if both boards approve a change it will be granted. If both disapprove of a change, it will be denied. If there is a mixed verdict, the two groups will work together to establish a consensus about future action.

The principal strength of this rigorous screening process is that it discourages trivial change. “Scope creep” is not likely to occur if individual change requests are subjected to rigorous scrutiny. Another strength is that the CCB can act collectively to resist harmful change requests from powerful players within or outside the organization. A properly constituted CCB—operating as a collective unit representing a broad range of organizational interests— is empowered to say no to even the most commanding individuals. As such, CCBs can serve as project managers’ friends, enabling them to say no to unreasonable change requests through indirect means.

An obvious problem with the CCB is its potential as a bottleneck. If the CCB spends too much time reviewing change requests, progress on the project may grind to a halt. Effective employment of the CCB entails a balancing act: on the one hand, scrutiny of change requests must be rigorous; on the other, the best interests of the project require that the review process occur as quickly as possible.

**Documenting Change**

Very small projects do not need elaborate documentation. When you change the washer of a leaky faucet, it doesn’t make sense to document the process in detail. However, as projects become larger and more complex, documentation takes on increasing importance. Following are some functions of documentation:

- It enables the project team to maintain an audit trail of their actions. If at some point a customer complains that the team did not undertake certain crucial steps on the project, the team can pull out the pertinent documentation to support their position.
- It records information that is beyond the capability of people to retain in their heads.
- It serves as a tool for coordinating the actions of different sets of team players. When someone new arrives on the scene, the quickest way he or she can get up to speed is to review documentation on what has transpired on the project so far.
- It is necessary for debugging and enhancing systems. For example, if a building experiences an electrical problem, the first step taken to fix the problem should be to acquire a wiring diagram. Without such a diagram, the repair job will be heavily dependent upon trial and error fixes.

In CM, careful documentation of ‘change requests and actions is important. Effective CM systems have CM data bases, libraries, and librarians to maintain the documentation. This, of course, means that CM systems involve a large amount of paperwork.
The CM process is heavily bureaucratic. Today, the term *bureaucracy* has negative connotations. This was not always so. As Max Weber showed in his work, bureaucracies arise for a purpose. They are natural organizational responses to dealing with complexity. It is a fundamental premise of CM that bureaucracy is the price we must be willing to pay in order to manage complexity.

*Updating Change*

As changes are accepted and incorporated, the project must be revised to reflect their presence. Baselines must be revised. People downstream must be warned, “Change is on the way!” A key component of CM is updating change in an orderly fashion. The updating process is closely tied to the documentation effort since a major element of updating is revision of the documentation.