MANAGING LARGE-SCALE, HIGH-PERFORMANCE EMBEDDED SYSTEMS DEVELOPMENT PROJECTS

Richard C. Jensen
Applied Microsystems Corp.
Redmond, WA

Richard Jensen is vice president of new business development at Applied Microsystems Corp. in Redmond, Wash. Prior to joining Applied Microsystems in 1984, he was founder, architect, and vice president of engineering at CADDEX Corp., a supplier of electronic publishing equipment. He has also been vice president of engineering at ATEX/AKI, operations manager for Metrodata Corp., and a systems engineer for Boeing's flight simulation team.

INTRODUCTION

Rapid advances in microprocessor technology have opened a broad spectrum of new opportunities for embedded systems development. The allowable size and sophistication of embedded applications has grown by more than an order of magnitude over the past decade. Today, it is not uncommon to find embedded systems containing over a megabyte of code and data. While this generally bodes well for the embedded systems industry, it can pose management problems during the development cycle.

Management methods that work well for small- or medium-scale projects are not necessarily suitable for large-scale projects. Managers transitioning from small to large projects are therefore susceptible to making mistakes in a number of areas. This paper identifies some of these mistakes and discusses techniques for avoiding them. Also discussed are the following topics:

- The relationship between product complexity and product development
- Identifying a project's "domain" (its scale and scope)
- The technologies that are pushing domains from small-scale to large-scale
- Project overhead
- Techniques for managing large-scale projects
- Project entropy
- Risk management
II) PRODUCT DESIGN DOMAINS

One of the most important steps in managing a project is identifying the scope of the effort. Scoping represents the cornerstone of effective project management. This section discusses a scheme for classifying projects and examines the relationship of a project's classification to the Product Development Process.

Software industry consultant Timothy Lister has developed the concept of design domains, which provide a useful scheme for classifying projects. In this scheme, all development projects can be thought of as falling into one of four specific domains. A project's design domain identifies the scope and scale of that project, as well as the management method used to run it. The four domains are:

Ad Hoc
The ad hoc domain involves a product of simple design that an engineer could solve alone, with minimum knowledge of the processor. The domain is so named because the process employed to design and build the product is "ad hoc," with only a minimal amount of structure or organization necessary to ensure a high-quality product. An example of a product falling into this domain is a programmable VCR controller. A formal, detailed production process is not required.

Intuitive
The intuitive domain involves products that are more complex than those in the ad hoc domain, but are still of a scale and scope that the manager can grasp intuitively. The majority of today's software and embedded systems projects fall into this category. Products in the intuitive design domain usually require a single, well-defined process plan that can be implemented in a sequential fashion. A single team of engineers with one manager is sufficient to manage the production of the product. An example of a product falling into this category is a fax machine controller.

Process
In the process domain, the scope of the product design is sufficiently complicated that rigorous pre-implementation planning is essential to the success of the development effort. The manager (or management team) must ensure that a set of well-defined processes are developed prior to the implementation and adhered to throughout development. The processes typically operate in parallel (or overlap to some degree) and determine the tasks, timeframes, and interactions of separate engineering teams working on major program components. A small number of processes can describe the development effort. Process coordination is managed primarily through meetings and discussion, rather than through a larger master plan. An example of a product in this domain is a vehicle power-train controller.

Systems
The most complicated design domain. The system domain is similar to the process domain, but typically involves a larger number of processes and necessitates process coordination through an overall "master" plan, rather than through informal communication among managers. It describes a formal system for linking together and coordinating the various processes and subprocesses that direct the day-to-day activities. Instead of meeting informally to discuss and alter process coordination, managers adhere to a complete plan that describes the entire development effort. A flight simulator would fall into this domain.

Switching Domains
Although domains are useful for categorizing projects, the real area of interest lies in the implications of switching from one domain to the other.

The ad hoc and intuitive domains both represent classes of projects that are managed in an a priori manner. This means that managers who transition from the ad hoc to the intuitive domain typically do not need to modify their management methods. At most, the intuitive domain requires more thought and structure but is still within the a priori realm. It is therefore useful to group both the ad hoc and intuitive domains into the single category of "small-scale" domains.

The most interesting (and potentially the most dangerous) transition is from either the ad hoc or intuitive to the process or systems domains. In this case, many managers make the mistake of assuming that their management methods "scale up" to handle the additional complexity. Projects in either the process or systems domains transcend a priori management, and much of the spontaneity so prevalent in small-scale projects is no longer possible. The process and systems domains therefore fall into a completely separate category that we can refer to as "large-scale" domains.

The key point for those accustomed to managing small-scale projects is that there are no "warning" signals when the transition is made from a small-scale to a large-scale domain. The tendency is to continue using small-scale management techniques because they have worked well in the past. As will be discussed later, this can be a primary source of project extension or failure.

The Relationship of a Product's Design Domain to the PDP
A product's design domain usually dictates how the product development process (PDP) is implemented. A typical PDP is shown in Figure 1.
II) PRODUCT DESIGN DOMAINS

One of the most important steps in managing a project is identifying the scope of the effort. Scoping represents the cornerstone of effective project management. This section discusses a scheme for classifying projects and examines the relationship of a project's classification to the Product Development Process.

Software industry consultant Timothy Lister has developed the concept of design domains, which provide a useful scheme for classifying projects. In this scheme, all development projects can be thought of as falling into one of four specific domains. A project's design domain identifies the scope and scale of that project, as well as the management method used to run it. The four domains are:

Ad Hoc

The ad hoc domain involves a product of simple design that an engineer could solve alone, with minimum knowledge of the processor. The domain is so named because the process employed to design and build the product is "ad hoc," with only a minimal amount of structure or organization necessary to ensure a high-quality product. An example of a product falling into this domain is a programmable VCR controller. A formal, detailed production process is not required.

Intuitive

The intuitive domain involves products that are more complex than those in the ad hoc domain, but are still of a scale and scope that the manager can grasp intuitively. The majority of today's software and embedded systems projects fall into this category. Products in the intuitive design domain usually require a single, well-defined process plan that can be implemented in a sequential fashion. A single team of engineers with one manager is sufficient to manage the production of the product. An example of a product falling into this category is a fax machine controller.

Process

In the process domain, the scope of the product design is sufficiently complicated that rigorous pre-implementation planning is essential to the success of the development effort. The manager (or management team) must ensure that a set of well-defined processes are developed prior to the implementation and adhered to throughout development. The processes typically operate in parallel (or overlap to some degree) and determine the tasks, timeframes, and interactions of separate engineering teams working on major program components. A small number of processes can describe the development effort. Process coordination is managed primarily through meetings and discussion, rather than through a larger master plan. An example of a product in this domain is a vehicle power-train controller.

Systems

The most complicated design domain. The system domain is similar to the process domain, but typically involves a larger number of processes and necessitates process coordination through an overall "master" plan, rather than through informal communication among managers. It describes a formal system for linking together and coordinating the various processes and subprocesses that direct the day-to-day activities. Instead of meeting informally to discuss and alter process coordination, managers adhere to a complete plan that describes the entire development effort. A flight simulator would fall into this domain.

Switching Domains

Although domains are useful for categorizing projects, the real area of interest lies in the implications of switching from one domain to the other.

The ad hoc and intuitive domains both represent classes of projects that are managed in an a priori manner. This means that managers who transition from the ad hoc to the intuitive domain typically do not need to modify their management methods. At most, the intuitive domain requires more thought and structure but is still within the a priori realm. It is therefore useful to group both the ad hoc and intuitive domains into the single category of "small-scale" domains.

The most interesting (and potentially the most dangerous) transition is from either the ad hoc or intuitive to the process or systems domains. In this case, many managers make the mistake of assuming that their management methods "scale up" to handle the additional complexity. Projects in either the process or systems domains transcend a priori management, and much of the spontaneity so prevalent in small-scale projects is no longer possible. The process and systems domains therefore fall into a completely separate category that we can refer to as "large-scale" domains.

The key point for those accustomed to managing small-scale projects is that there are no "warning" signals when the transition is made from a small-scale to a large-scale domain. The tendency is to continue using small-scale management techniques because they have worked well in the past. As will be discussed later, this can be a primary source of project extension or failure.

The Relationship of a Product's Design Domain to the PDP

A product's design domain usually dictates how the product development process (PDP) is implemented. A typical PDP is shown in Figure 1.
III) DOMAIN MIGRATION

In order to understand the adaptations which managers must make when transitioning to new design domains, it is important to understand the forces that are shifting managers into this new territory. Not surprisingly, these forces are technology-based. This section describes the two technologies that are having the greatest impact on design domains - CPUs and operating systems.

CPU Capability

Once confined to less than 4KB of total code and data space using 4-bit processors, embedded systems engineers can now write applications that consume up to 4 Gigabytes using 32-bit processors. In other words, programs can be roughly a million times larger than they were ten years ago. Although only the most sophisticated products currently need this much addressability, the nature of engineering is to utilize resources when they are available. Even simple designs will expand to fill more of the chip's addressable space, especially as prices of these semiconductors move lower.

With 32-bit addressability, large-scale embedded applications consisting of hundreds of thousands of lines of source code are already in service. Programs consisting of millions of lines are now being produced. This addressability will also make it possible to implement embedded applications with very large data sets (such as artificial intelligence programs).

Driving such large quantities of code and data through a CPU will require a commensurate increase in processing speed. 32-bit processors run today at 40 MHz and will soon be available in 50 MHz versions. Upcoming 64-bit processors (such as the Intel i960MM, i860XP and MIPS R4000) will eventually be produced to run at over 100 MHz. The 64-bit generation will increase speed further through heavy use of pipelining, cache (internal and external) and superscalar architectures that enable up to three instructions per clock cycle at peak rates.

It is expected that the trend of approximately doubling performance every two years will continue. By the late 1990s, we should see chips with sustainable execution speeds of over 500 MIPS.

Using bus size (in bits) multiplied by maximum clock speed as a unit of measure, we can gain a rough idea of the growth in capability of different generations of CPUs. Notice in Figure 2 that 64-bit CPUs measure 800 times greater than 4-bit CPUs. Note: Microprocessors come in many different models. The figures represent approximate maximum values for the specified generation of CPU and do not take into account multipliers that result from architectural features.

III) DOMAIN MIGRATION

In order to understand the adaptations which managers must make when transitioning to new design domains, it is important to understand the forces that are shifting managers into this new territory. Not surprisingly, these forces are technology-based. This section describes the two technologies that are having the greatest impact on design domains - CPUs and operating systems.

CPU Capability

Once confined to less than 4KB of total code and data space using 4-bit processors, embedded systems engineers can now write applications that consume up to 4 Gigabytes using 32-bit processors. In other words, programs can be roughly a million times larger than they were ten years ago. Although only the most sophisticated products currently need this much addressability, the nature of engineering is to utilize resources when they are available. Even simple designs will expand to fill more of the chip's addressable space, especially as prices of these semiconductors move lower.

With 32-bit addressability, large-scale embedded applications consisting of hundreds of thousands of lines of source code are already in service. Programs consisting of millions of lines are now being produced. This addressability will also make it possible to implement embedded applications with very large data sets (such as artificial intelligence programs).

Driving such large quantities of code and data through a CPU will require a commensurate increase in processing speed. 32-bit processors run today at 40 MHz and will soon be available in 50 MHz versions. Upcoming 64-bit processors (such as the Intel i960MM, i860XP and MIPS R4000) will eventually be produced to run at over 100 MHz. The 64-bit generation will increase speed further through heavy use of pipelining, cache (internal and external) and superscalar architectures that enable up to three instructions per clock cycle at peak rates.

It is expected that the trend of approximately doubling performance every two years will continue. By the late 1990s, we should see chips with sustainable execution speeds of over 500 MIPS.

Using bus size (in bits) multiplied by maximum clock speed as a unit of measure, we can gain a rough idea of the growth in capability of different generations of CPUs. Notice in Figure 2 that 64-bit CPUs measure 800 times greater than 4-bit CPUs. Note: Microprocessors come in many different models. The figures represent approximate maximum values for the specified generation of CPU and do not take into account multipliers that result from architectural features.
Figure 1 - A Typical Product Development Process

In management theory, the PDP shown in Figure 1 is sometimes referred to as the waterfall model because it is represented pictorially as a waterfall, flowing sequentially from one phase to the next. More sophisticated models exist, but this is the one most commonly used today.

A product's design domain represents a major influence on how the PDP is defined and implemented. For projects in the ad-hoc and intuitive domains, the development team is small, the project is easily defined, and the individual steps of the PDP may be completed with a minimum of formality or structure. Process overhead (the time spent managing, communicating, measuring progress and reporting) is low compared with large-scale projects.

Ad hoc and intuitive domain projects generally allow more flexibility in executing the PDP. Entire steps may be skipped or done out of order, with little or no harmful effect on the overall process. Many managers freely admit that they have run at least one project where implementation preceded specification.

As products become more sophisticated, a greater percentage of development efforts will move into the process and systems domains. For projects in these domains, a large degree of flexibility within the PDP is rarely possible. The success of these more complicated projects depends on a PDP that is performed in a formal, structured manner. Engineering managers transitioning from the ad hoc and intuitive domains may not be accustomed to (or even resist) defining or working with this structure.

III) DOMAIN MIGRATION

In order to understand the adaptations which managers must make when transitioning to new design domains, it is important to understand the forces that are shifting managers into this new territory. Not surprisingly, these forces are technology-based. This section describes the two technologies that are having the greatest impact on design domains - CPUs and operating systems.

CPU Capability

Once confined to less than 4KB of total code and data space using 4-bit processors, embedded systems engineers can now write applications that consume up to 4 Gigabytes using 32-bit processors. In other words, programs can be roughly a million times larger than they were ten years ago. Although only the most sophisticated products currently need this much addressability, the nature of engineering is to utilize resources when they are available. Even simple designs will expand to fill more of the chip's addressable space, especially as prices of these semiconductors move lower.

With 32-bit addressability, large-scale embedded applications consisting of hundreds of thousands of lines of source code are already in service. Programs consisting of millions of lines are now being produced. This addressability will also make it possible to implement embedded applications with very large data sets (such as artificial intelligence programs).

Driving such large quantities of code and data through a CPU will require a commensurate increase in processing speed. 32-bit processors run today at 40 MHz and will soon be available in 50 MHz versions. Upcoming 64-bit processors (such as the Intel i960MM, i860XP and MIPS R4000) will eventually be produced to run at over 100 MHz. The 64-bit generation will increase speed further through heavy use of pipelining, cache (internal and external) and superscalar architectures that enable up to three instructions per clock cycle at peak rates.

It is expected that the trend of approximately doubling performance every two years will continue. By the late 1990s, we should see chips with sustainable execution speeds of over 500 MIPS.

Using bus size (in bits) multiplied by maximum clock speed as a unit of measure, we can gain a rough idea of the growth in capability of different generations of CPUs. Notice in Figure 2 that 64-bit CPUs measure 800 times greater than 4-bit CPUs. Note: Microprocessors come in many different models. The figures represent approximate maximum values for the specified generation of CPU and do not take into account multipliers that result from architectural features.
Operating System Services

A million lines of source code does not necessarily make a program sophisticated or complex. While addressability and clock speed are major contributors to application size, they do not completely drive application complexity. Equally important are operating system services, including multitasking, multiprocessing and interprocess communication. The current generation of applications are beginning to take advantage of these services and some already find them indispensable. For example, programs that can spawn multiple child processes, some of which run on different CPUs and communicate with each other through shared memory are being produced now. These applications will require much more complex architectures and programming tools than their predecessors.

Shifting Gears

As embedded systems applications begin to integrate these new technologies, designs will slowly "migrate" upward into higher domains. Although the majority of today's embedded systems applications still represent small-scale domains, a large percentage of these will eventually shift to large-scale domains. While there will always be a need for small-scale applications, many managers will find themselves supervising large-scale projects for the first time.

IV) MANAGING COMPLEXITY

Implementing complex 32-bit applications involves larger design and development teams, different tooling strategies, different team organizations, team communications, and other requirements. This section discusses common mistakes that managers can make when switching domains. Frequently, managers make the mistake of applying small-scale management techniques to large-scale projects. This can lead to process delays and even complete breakdowns during product implementation. The following mistakes are typical:

- In smaller design domains, managers tend to abdicate process-oriented decisions to junior managers or engineering team leaders. In larger projects, this practice can cause problems because process coordination is a more complex undertaking and not necessarily manageable by less experienced team leaders or programmers. The need for managers to be involved with day-to-day operations and process management is even more critical in large-scale projects.

- The complexity of a large-scale development effort implies that a single individual cannot visualize or intuit the entire project. In process or systems design domains, one or more groups are responsible for producing dozens or hundreds of sub-component modules. Coordinating the development, testing, and integration of these modules requires a formal process plan that is not used in small-scale projects. This plan represents a higher level of abstraction, involving more emphasis on global issues and less detail on day-to-day operations. Without such a process plan, a manager is prone to making decisions without appreciating or understanding their effect on the entire process. Although meant to increase productivity, these decisions can often have the opposite effect.

- Embedded systems projects have typically been managed in a serial fashion. The waterfall model (Figure 1) is applied in stepwise format - each phase of the product development process follows the previous one. Large-scale projects tend to involve multiple processes which overlap, requiring a more parallel management scheme. Ad hoc and intuitive methods are not suitable for managing multiple, interacting processes.

A typical scenario is a senior manager leading a 25-person design effort consisting of four junior managers and 21 engineers. One of the junior managers informs the senior manager that a small design change to one of the key program modules will result in completing the module two weeks ahead of schedule. Having trust in his junior managers, the senior manager abdicates the decision to the junior manager, who makes the change. Later, when all the product modules from the four groups are brought together for integration, it is found that the "small" design change results in the module's failure to interoperate with other modules in the program. The project's schedule slips by four weeks while the module is re-coded and tested. To make matters worse, saving the two weeks in the first place was irrelevant because this only meant that one group would finish early and wait for the other three to complete their modules.

Instead of managing the decision across all four teams and adhering to a process plan, the senior manager took an unnecessary risk and was not able to anticipate the effect the change would have on the entire system.
Operating System Services

A million lines of source code does not necessarily make a program sophisticated or complex. While addressability and clock speed are major contributors to application size, they do not completely drive application complexity. Equally important are operating system services, including multitasking, multiprocessing and interprocess communication. The current generation of applications are beginning to take advantage of these services and some already find them indispensable. For example, programs that can spawn multiple child processes, some of which run on different CPUs and communicate with each other through shared memory are being produced now. These applications will require much more complex architectures and programming tools than their predecessors.

Shifting Gears

As embedded systems applications begin to integrate these new technologies, designs will slowly "migrate" upward into higher domains. Although the majority of today's embedded systems applications still represent small-scale domains, a large percentage of these will eventually shift to large-scale domains. While there will always be a need for small-scale applications, many managers will find themselves supervising large-scale projects for the first time.

IV) MANAGING COMPLEXITY

Implementing complex 32-bit applications involves larger design and development teams, different tooling strategies, different team organizations, team communications, and other requirements. This section discusses common mistakes that managers can make when switching domains. Frequently, managers make the mistake of applying small-scale management techniques to large-scale projects. This can lead to process delays and even complete breakdowns during product implementation. The following mistakes are typical:

- In smaller design domains, managers tend to abdicate process-oriented decisions to junior managers or engineering team leaders. In larger projects, this practice can cause problems because process coordination is a more complex undertaking and not necessarily manageable by less experienced team leaders or programmers. The need for managers to be involved with day-to-day operations and process management is even more critical in large-scale projects.

- The complexity of a large-scale development effort implies that a single individual cannot visualize or intuit the entire project. In process or systems design domains, one or more groups are responsible for producing dozens or hundreds of sub-component modules. Coordinating the development, testing, and integration of these modules requires a formal process plan that is not used in small-scale projects. This plan represents a higher level of abstraction, involving more emphasis on global issues and less detail on day-to-day operations. Without such a process plan, a manager is prone to making decisions without appreciating or understanding their effect on the entire process. Although meant to increase productivity, these decisions can often have the opposite effect.

- Embedded systems projects have typically been managed in a serial fashion. The waterfall model (Figure 1) is applied in stepwise format - each phase of the product development process follows the previous one. Large-scale projects tend to involve multiple processes which overlap, requiring a more parallel management scheme. Ad hoc and intuitive methods are not suitable for managing multiple, interacting processes.

A typical scenario is a senior manager leading a 25-person design effort consisting of four junior managers and 21 engineers. One of the junior managers informs the senior manager that a small design change to one of the key program modules will result in completing the module two weeks ahead of schedule. Having trust in his junior managers, the senior manager abdicates the decision to the junior manager, who makes the change. Later, when all the product modules from the four groups are brought together for integration, it is found that the "small" design change results in the module's failure to interoperate with other modules in the program. The project's schedule slips by four weeks while the module is re-coded and tested. To make matters worse, saving the two weeks in the first place was irrelevant because this only meant that one group would finish early and wait for the other three to complete their modules.

Instead of managing the decision across all four teams and adhering to a process plan, the senior manager took an unnecessary risk and was not able to anticipate the effect the change would have on the entire system.
Complexity and Project Overhead

In his widely read book *Software Engineering Economics*, Barry Boehm describes a software productivity model that includes 14 factors which can increase the time it takes to develop a product. Boehm proposes that reducing the contributions of each of these factors helps a company move closer to maximum efficiency on any given software development effort.

Figure 3 shows Boehm’s productivity chart. The chart was originally developed for the purpose of software development project cost estimation. The exact figures therefore do not pertain specifically to embedded systems, but the relative measurements apply to any development effort.

The number beside each bar represents the “overhead” associated with that factor. For example, language experience has an associated productivity overhead of 1.2, meaning that a project normally taking 10 man-months would take 12 months if the team lacked the appropriate language experience.

The chart shows that with only one exception (team capability), product complexity represents the largest influence (a multiplier of 2.36) on overall project productivity. In other words, as project size increases, project overhead increases. Figures 4a and 4b illustrate this point, showing how team communication becomes more cumbersome as project size (and therefore staff) increases. The time (overhead) necessary to exchange information in a small group is minimal and can be efficiently accomplished through informal and spontaneous meetings.

In a large project, both intra-team and inter-team communication become more complex. Meetings must be scheduled, planned and coordinated. The number of issues to discuss also increases, leading to the need to meet more frequently and for longer periods of time.

Managing Process- and Systems-Domain Projects

Managing large-scale projects requires significantly more planning in the early stages of the product development process. Not all managers realize this. The following are a few points to consider when planning a large-scale development effort. This is not a comprehensive list, but an outline of a more complete procedure that is beyond the scope of
Complexity and Project Overhead

In his widely read book *Software Engineering Economics*, Barry Boehm describes a software productivity model that includes 14 factors which can increase the time it takes to develop a product. Boehm proposes that reducing the contributions of each of these factors helps a company move closer to maximum efficiency on any given software development effort.

Figure 3 shows Boehms' productivity chart. The chart was originally developed for the purpose of software development project cost estimation. The exact figures therefore do not pertain specifically to embedded systems, but the relative measurements apply to any development effort.

The number beside each bar represents the “overhead” associated with that factor. For example, language experience has an associated productivity overhead of 1.2, meaning that a project normally taking 10 man-months would take 12 months if the team lacked the appropriate language experience.

The chart shows that with only one exception (team capability), product complexity represents the largest influence (a multiplier of 2.36) on overall project productivity. In other words, as project size increases, project overhead increases. Figures 4a and 4b illustrate this point, showing how team communication becomes more cumbersome as project size (and therefore staff) increases. The time (overhead) necessary to exchange information in a small group is minimal and can be efficiently accomplished through informal and spontaneous meetings.

Managing Process- and Systems-Domain Projects

Managing large-scale projects requires significantly more planning in the early stages of the product development process. Not all managers realize this. The following are a few points to consider when planning a large-scale development effort. This is not a comprehensive list, but an outline of a more complete procedure that is beyond the scope of
this paper. For additional information regarding large-scale project management, the
Software Engineering Institute (SEI) is a good resource.

Domain Assessment

♦ Determine the project scope

A project’s scope is usually determined from the marketing and functional requirements of the product. Scoping a project involves more art than science and is usually estimated by a manager with experience in the particular product area. The scoping stage helps to determine the design domain.

♦ Identify the design domain

To manage a project with maximum efficiency, managers must be “domain-aware.” Domain awareness means the project is identified at the PDP scoping stage as being in one of the four design domains. A product’s domain is not always self-evident, especially in circumstances where the management team has no prior experience with the product. In these cases, care must be taken to make the proper domain designation.

♦ Communicate the domain and its consequences to the management chain

If all members of the management chain are not in agreement regarding the product’s identified domain, the implementation effort may fail to meet management expectations. A typical scenario is that senior management assumes the product is simpler than it really is, making the implementation process appear unreasonably slow. Junior and senior managers alike should agree not only on the product’s domain designation, but on the definition of the domain itself.

Capability and Tools Assessment

♦ Team capability

As Figure 3 shows, application experience and team capability are dominant factors in software productivity. In large-scale projects, it is essential to study the skill sets of the engineering staff and assess their ability to deliver the product as specified. Acquisition of additional engineers with more appropriate skills and experience may be necessary. Training for engineers already on board can result in additional productivity.

♦ Tools assessment

Tooling has become a complex process in large-scale projects. The sophistication of development systems has made choice of the proper tools a major factor in software productivity. Tools that worked well for prior projects might need to be upgraded or replaced.

Process Planning

♦ Identify and specify the processes

Large-scale projects are typically conglomerations of many processes. Some of these processes are interrelated while others stand alone. Time should be invested to identify and specify the various processes involved in the development effort, including who is accountable for each process. The creation of block diagrams showing process flow and interrelation can help managers “visualize” the entire project. A variety of software packages are available that can help with the project management process. These packages can produce Pert and Gant charts, and also store and maintain various project databases.

♦ Specify inter- and intra-team communications

The process plan should include a specification for inter-team communication, including how often interaction should occur, how the interaction should take place (face-to-face, electronic mail, and so on) and the type of information to be shared. Also important are the controls placed on the information; for example, which portions of project documents are to be given group access and change privileges.

♦ Progress measurement and reporting

Specify how progress will be measured and reported, and how often this is to take place. All junior and senior managers must agree to the measurement and reporting process. Although engineers and managers generally regard measurement as a bureaucratic hassle, measurement is relatively quick and easy after well-defined measurement standards have been set. As Gilb’s law states: “Anything you need to quantify can be measured in some way that is superior to not measuring it at all.”

V) Change, Entropy and Risk Management

Over the years, a variety of management models have proven their effectiveness. However, models are ideal representations that provide only a rough approximation of reality. Regardless of how diligently it may be followed, no management theory can predict and respond to the many technical problems, staffing changes, management pressures, political decisions and other challenges inherent in every large project. This section describes forces that are likely to influence the PDP and discusses techniques for handling them.

For smaller projects, challenges experienced during the product development cycle can be handled with relative quickness. Since the project’s scope is small, the effect on the whole project of a change in design, staffing, tooling, or timeframe can be assessed swiftly. Usually, managers can employ their intuition to suggest a new course of action or change to the development plan.

In larger domains, the effect of design or process changes is difficult to assess. Since the system is more complex, more time must be spent sorting out the myriad of issues and
this paper. For additional information regarding large-scale project management, the Software Engineering Institute (SEI) is a good resource.

Domain Assessment

- Determine the project scope
  A project's scope is usually determined from the marketing and functional requirements of the product. Scoping a project involves more art than science and is usually estimated by a manager with experience in the particular product area. The scoping stage helps to determine the design domain.

- Identify the design domain
  To manage a project with maximum efficiency, managers must be "domain-aware." Domain awareness means the project is identified at the PDP scoping stage as being in one of the four design domains. A product's domain is not always self-evident, especially in circumstances where the management team has no prior experience with the product. In these cases, care must be taken to make the proper domain designation.

- Communicate the domain and its consequences to the management chain
  If all members of the management chain are not in agreement regarding the product's identified domain, the implementation effort may fail to meet management expectations. A typical scenario is that senior management assumes the product is simpler than it really is, making the implementation process appear unreasonably slow. Junior and senior managers alike should agree not only on the product's domain designation, but on the definition of the domain itself.

Capability and Tools Assessment

- Team capability
  As Figure 3 shows, application experience and team capability are dominant factors in software productivity. In large-scale projects, it is essential to study the skill sets of the engineering staff and assess their ability to deliver the product as specified. Acquisition of additional engineers with more appropriate skills and experience may be necessary. Training for engineers already on board can result in additional productivity.

- Tools assessment
  Tooling has become a complex process in large-scale projects. The sophistication of development systems has made choice of the proper tools a major factor in software productivity. Tools that worked well for prior projects might need to be upgraded or replaced.

Process Planning

- Identify and specify the processes
  Large-scale projects are typically conglomerations of many processes. Some of these processes are interrelated while others stand alone. Time should be invested to identify and specify the various processes involved in the development effort, including who is accountable for each process. The creation of block diagrams showing process flow and interrelation can help managers "visualize" the entire project. A variety of software packages are available that can help with the project management process. These packages can produce Pert and Gant charts, and also store and maintain various project databases.

- Specify inter- and intra-team communications
  The process plan should include a specification for inter-team communication, including how often interaction should occur, how the interaction should take place (face-to-face, electronic mail, and so on) and the type of information to be shared. Also important are the controls placed on the information; for example, which portions of project documents are to be given group access and change privileges.

- Progress measurement and reporting
  Specify how progress will be measured and reported, and how often this is to take place. All junior and senior managers must agree to the measurement and reporting process. Although engineers and managers generally regard measurement as a bureaucratic hassle, measurement is relatively quick and easy after well-defined measurement standards have been set. As Gilb's law states: "Anything you need to quantify can be measured in some way that is superior to not measuring it at all."

V) CHANGE, ENTROPY AND RISK MANAGEMENT

Over the years, a variety of management models have proven their effectiveness. However, models are ideal representations that provide only a rough approximation of reality. Regardless of how diligently it may be followed, no management theory can predict and respond to the many technical problems, staffing changes, management pressures, political decisions and other challenges inherent in every large project. This section describes forces that are likely to influence the PDP and discusses techniques for handling them.

For smaller projects, challenges experienced during the product development cycle can be handled with relative quickness. Since the project's scope is small, the effect on the whole project of a change in design, staffing, tooling, or timeframe can be assessed swiftly. Usually, managers can employ their intuition to suggest a new course of action or change to the development plan.

In larger domains, the effect of design or process changes is difficult to assess. Since the system is more complex, more time must be spent sorting out the myriad of issues and
outcomes that can result from a single change. This is why process planning is so important in large projects - it minimizes the number of "course corrections" needed during implementation. However, regardless of how thorough a manager is in preparing for a product implementation, all software development projects will experience the following two "laws":

The Law of Continuous Change: "A system will undergo continuous modification throughout its life cycle"

The Law of Increasing Entropy: "The entropy of a system increases with time, unless special care is taken to maintain or reduce it."

Here, "change" refers to changes made with the intention of improving some aspect of the system, while "entropy" refers to a natural state of disorganization towards which a system will gravitate. These laws apply equally well to both software and the software development process. Their impact is especially magnified within large projects, where change and entropy are more difficult to control. Steps can, however, be taken to minimize the impact of changes within a large project. An important step is utilization of risk management techniques.

Risk Management

Any change to a system, especially a large one, carries a certain amount of risk with it. Since change and entropy are inevitable, it makes sense to plan for risks appropriately. In process management, the following items should be considered:

- Obtain management consensus
  In large projects, all significant changes to either the product design or the process should be communicated through the management chain, along with descriptions of the "upside" and "downside" risks, and the direction you have chosen. Senior managers should be asked to respond or suggest alternative solutions if they do not agree with your approach. The goal is to achieve consensus throughout the management chain.

- Both the product and the process should be of modular design
  Product modularity is important in large-scale projects. The product components should be self-contained and as independent as possible of other modules. Well-defined and well-documented interfaces should describe the interaction between system components. Changes made to a component are therefore less likely to create the need for changes in other modules.

  The concept of modularity is also important when defining the process. Process plans should define clear and unambiguous processes that do not depend heavily on each other. Overall process design should make it possible to make changes in one process without unreasonably affecting other processes.

VI) CONCLUSION

Advances in technology have created a new generation of more complex embedded systems products. Management techniques typically applied in small-scale projects do not necessarily scale up to meet the needs of complex environments. Being domain-aware at the scoping stage of the PDP is the first step towards employing the proper management techniques. Once the project's domain has been recognized, establishing a thorough process plan and effectively managing risk can help prevent many of the problems usually associated with large-scale projects.

VII) REFERENCES

1) Boehm [1981]

2) Boehm [1988]

3) Lehman and Belady [1985]

4) Humphrey [1988]

5) DeMarco and Lister [1987]
outcomes that can result from a single change. This is why process planning is so important in large projects - it minimizes the number of "course corrections" needed during implementation. However, regardless of how thorough a manager is in preparing for a product implementation, all software development projects will experience the following two "laws":

The Law of Continuous Change: "A system will undergo continuous modification throughout its life cycle"

The Law of Increasing Entropy: "The entropy of a system increases with time, unless special care is is taken to maintain or reduce it."

Here, "change" refers to changes made with the intention of improving some aspect of the system, while "entropy" refers to a natural state of disorganization towards which a system will gravitate. These laws apply equally well to both software and the software development process. Their impact is especially magnified within large projects, where change and entropy are more difficult to control. Steps can, however, be taken to minimize the impact of changes within a large project. An important step is utilization of risk management techniques.

Risk Management

Any change to a system, especially a large one, carries a certain amount of risk with it. Since change and entropy are inevitable, it makes sense to plan for risks appropriately. In process management, the following items should be considered:

- Obtain management consensus

  In large projects, all significant changes to either the product design or the process should be communicated through the management chain, along with descriptions of the "upside" and "downside" risks, and the direction you have chosen. Senior managers should be asked to respond or suggest alternative solutions if they do not agree with your approach. The goal is to achieve consensus throughout the management chain.

- Both the product and the process should be of modular design

  Product modularity is important in large-scale projects. The product components should be self-contained and as independent as possible of other modules. Well-defined and well-documented interfaces should describe the interaction between system components. Changes made to a component are therefore less likely to create the need for changes in other modules.

  The concept of modularity is also important when defining the process. Process plans should define clear and unambiguous processes that do not depend heavily on each other. Overall process design should make it possible to make changes in one process without unreasonably affecting other processes.

VI) CONCLUSION

Advances in technology have created a new generation of more complex embedded systems. Management techniques typically applied in small-scale projects do not necessarily scale up to meet the needs of complex environments. Being domain-aware at the scoping stage of the PDP is the first step towards employing the proper management techniques. Once the project's domain has been recognized, establishing a thorough process plan and effectively managing risk can help prevent many of the problems usually associated with large-scale projects.

VII) REFERENCES

1) Boehm [1981]  

2) Boehm [1988]  

3) Lehman and Belady [1985]  

4) Humphrey [1988]  

5) DeMarco and Lister [1987]  