Derek Hatley provides consulting, seminars, and workshops on the system-development process, system requirements, and architecture-specification methods that he developed with Imtiaz Pirbhai. These methods, described in their book, Strategies for Real-Time System Specification, have been adopted by corporations worldwide. Hatley has more than thirty years of industry experience and is now a principal staff engineer at Smiths Industries. He has an MS in electrical engineering.
ABSTRACT

System development is a multi-faceted activity, in which it is particularly true that the chain is only as strong as its weakest link. Some of its elements are: project management, configuration management, quality assurance, metrics, development process maturity, requirements specification and analysis, development methods (including test methods), automated tools, reviews, and personnel selection. A great deal of work has been carried out in recent years by this author and others addressing various of these elements; several of these workers properly emphasize the Systems nature of system development.

It remains, however, to address all the elements in a unified way, and this paper is an attempt to do so. It addresses all of the elements of system development, using the Software Engineering Institute's Capability Maturity Model as a guide, and defines the proper roles of process, methods, and tools.

THE SYSTEM DEVELOPMENT PROCESS

Figure 1 illustrates system development as a multi-layered, parallel process in which each layer:

- receives requirements and a design environment from the layer above,
- analyses the requirements,
- performs a design step subject to any imposed constraints,
- allocates requirements to the new design elements,
- passes the requirements and design environment to the layer below,
- develops an integration and test system for the layer,
- integrates and tests the design elements received from the layer below,
- passes the tested items on to the layer above.

Every path in this model includes feedback, and the whole process starts up in its entirety at the beginning of a project. Everything proceeds in parallel; the various products of the process are progressively stored in a database until they are all complete. These concepts were described in much more detail in [1], and serve as our starting point here.

Figure 2 is a more detailed look at a typical layer of Figure 1. It shows the three main activities in each layer—design, integration, and test—and the information and material flows between those activities and between this and other layers. An important aspect of this figure, and of the whole model, is the similarity and equal status between the design and test processes. Both receive the same set of requirements, both interpret and analyze those requirements, and both develop a system based on those requirements. The actual process of testing is in fact a comparison between the system under test and the test system, to determine if their interpretations of the requirements differ. If they do, then either one (or both) of the systems, or the requirements themselves, might be in error, and in any case, the error is then corrected.

![Figure 1: Multi-layered, Parallel Development](image-url)

A corollary to this duality between design and test development is their mutual impact on each other. It is a given that the test environment and process will be influenced by the system to be tested, but it is equally true that the system to be tested has requirements imposed on it by the test system—the system must be testable. These testability requirements might, for example, call for special test connectors, special built-in test...
ABSTRACT

System development is a multi-faceted activity, in which it is particularly true that the chain is only as strong as its weakest link. Some of its elements are: project management, configuration management, quality assurance, metrics, development process maturity, requirements specification and analysis, development methods (including test methods), automated tools, reviews, and personnel selection. A great deal of work has been carried out in recent years by this author and others addressing various of these elements; several of these workers properly emphasize the Systems nature of system development.

It remains, however, to address all the elements in a unified way, and this paper is an attempt to do so. It addresses all of the elements of system development, using the Software Engineering Institute's Capability Maturity Model as a guide, and defines the proper roles of process, methods, and tools.

THE SYSTEM DEVELOPMENT PROCESS

Figure 1 illustrates system development as a multi-layered, parallel process in which each layer:
- receives requirements and a design environment from the layer above,
- analyses the requirements,
- performs a design step subject to any imposed constraints,
- allocates requirements to the new design elements,
- passes the requirements and design environment to the layer below,
- develops an integration and test system for the layer,
- integrates and tests the design elements received from the layer below,
- passes the tested items on to the layer above.

Every path in this model includes feedback, and the whole process starts up in its entirety at the beginning of a project. Everything proceeds in parallel: the various products of the process are progressively stored in a database until they are all complete. These concepts were described in much more detail in [1], and serve as our starting point here.

Figure 2 is a more detailed look at a typical layer of Figure 1. It shows the three main activities in each layer—design, integration, and test—and the information and material flows between those activities and between this and other layers. An important aspect of this figure, and of the whole model, is the similarity and equal status between the design and test processes. Both receive the same set of requirements, both interpret and analyze those requirements, and both develop a system based on those requirements. The actual process of testing is in fact a comparison between the system under test and the test system, to determine if their interpretations of the requirements differ. If they do, then either one (or both) of the systems, or the requirements themselves, might be in error, and in any case, the error is then corrected.

A corollary to this duality between design and test development is their mutual impact on each other. It is a given that the test environment and process will be influenced by the system to be tested, but it is equally true that the system to be tested has requirements imposed on it by the test system—the system must be testable. These testability requirements might, for example, call for special test connectors, special built-in test

Hatley ©1992, Derek J. Hatley, System Strategies

Hatley System Development 321

©1992, Derek J. Hatley, System Strategies

Hatley System Development
software, or test panels. This design/test duality reinforces the idea that everything, but the test process in particular, must start at the very beginning of development so that the test requirements and test strategy are established at the same time as the rest of the development requirements and strategy.

Another important, and often misunderstood, characteristic of the development process is the pervasive and multi-faceted nature of requirements. We have already touched on this in the above paragraphs, but it is worth emphasizing that requirements do not just come at us from the customer; at any given point in Figures 1 and 2, they come from all around us: from other processes in our layer, from the layers above and below, and from sources outside the project other than the customer (such as management). And, of course, we join the fray by generating requirements ourselves from within our own domain in the process. We will discuss the nature of all these various requirements later.

PARALLEL, COOPERATIVE SYSTEM DEVELOPMENT

Traditionally, system development has been characterized by two properties: a sequential approach, typified by the waterfall model, and the division of the work into sharply divided domains, usually by discipline and organizational group. A little reflection will reveal that this traditional development paradigm is incompatible with the scenario described in the previous section. It seems that a new paradigm is needed, and we are not seeking change just for change's sake—the fact is that the old paradigm never worked anyway! System development has always ended up working as described in the previous section, but instead of being planned that way, it would perpetually take project management by surprise. When requirements or design changed late in the development for perfectly legitimate reasons, such as unforeseen limitations in the technology, it was considered a disaster; when issues requiring multi-disciplinary solutions arose, it would lead to finger pointing and shouting matches instead of cooperation.

What we need is an approach to system development that is based on the reality that everything is interdependent, that all tasks must be active throughout the development process, and that interdisciplinary issues (that is, most issues) require interdisciplinary teams working together cooperatively. Yes, it does make project planning and management harder, but this is a small price to pay for eliminating the horrors described above (and besides, who said project management should be easy?)

In the rest of this paper we will discuss what it means to have a parallel, cooperative development approach, and what impact it has on all the various elements of system development.

THE ELEMENTS OF SYSTEM DEVELOPMENT

There are many aspects to system development. The Software Engineering Institute's (SEI) Capability Maturity Model (CMM)[2,3,4] identifies Key Process Areas (KPA), and we will use these as a reference, discussing for each one how it should be realized in a parallel, cooperative process. In the paragraphs that follow, KPAs are identified together with the maturity level with which they are associated. These levels are: 1. Initial; 2. Repeatable; 3. Defined; 4. Managed; 5. Optimizing.

A weakness of the CMM is that, contrary to the theme of this paper, it deals only with software, not with total system development. However, it is not too difficult to extrapolate its principles to the whole system, and this extrapolation is assumed in the remainder of this paper.

It is important to understand that the development process is one of three factors on which system development is founded. The other two are people and technology. Both the SEI CMM and this paper focus on the process, but we discuss the other two below.
software, or test panels. This design/test duality reinforces the idea that everything, but the test process in particular, must start at the very beginning of development so that the test requirements and test strategy are established at the same time as the rest of the development requirements and strategy.

Another important, and often misunderstood, characteristic of the development process is the pervasive and multi-faceted nature of requirements. We have already touched on this in the above paragraphs, but it is worth emphasizing that requirements do not just come at us from the customer; at any given point in Figures 1 and 2, they come from all around us: from other processes in our layer, from the layers above and below, and from sources outside the project other than the customer (such as management). And, of course, we join the fray by generating requirements ourselves from within our own domain in the process. We will discuss the nature of all these various requirements later.

PARALLEL, COOPERATIVE SYSTEM DEVELOPMENT

Traditionally, system development has been characterized by two properties: a sequential approach, typified by the waterfall model, and the division of the work into sharply divided domains, usually by discipline and organizational group. A little reflection will reveal that this traditional development paradigm is incompatible with the scenario described in the previous section. It seems that a new paradigm is needed, and we are not seeking change just for change’s sake—the fact is that the old paradigm never worked anyway! System development has always ended up working as described in the previous section, but instead of being planned that way, it would perpetually take project management by surprise. When requirements or design changed late in the development for perfectly legitimate reasons, such as unforeseen limitations in the technology, it was considered a disaster; when issues requiring multi-disciplinary solutions arose, it would lead to finger pointing and shouting matches instead of cooperation.

What we need is an approach to system development that is based on the reality that everything is interdependent, that all tasks must be active throughout the development process, and that interdisciplinary issues (that is, most issues) require interdisciplinary teams working together cooperatively. Yes, it does make project planning and management harder, but this is a small price to pay for eliminating the horrors described above (and besides, who said project management should be easy?)

In the rest of this paper we will discuss what it means to have a parallel, cooperative development approach, and what impact it has on all the various elements of system development.

THE ELEMENTS OF SYSTEM DEVELOPMENT

There are many aspects to system development. The Software Engineering Institute’s (SEI) Capability Maturity Model (CMM)[2,3,4] identifies Key Process Areas (KPA), and we will use these as a reference, discussing for each one how it should be realized in a parallel, cooperative process. In the paragraphs that follow, KPAs are identified together with the maturity level with which they are associated. These levels are: 1. Initial; 2. Repeatable; 3. Defined; 4. Managed; 5. Optimizing.

A weakness of the CMM is that, contrary to the theme of this paper, it deals only with software, not with total system development. However, it is not too difficult to extrapolate its principles to the whole system, and this extrapolation is assumed in the remainder of this paper.

It is important to understand that the development process is one of three factors on which system development is founded. The other two are people and technology. Both the SEI CMM and this paper focus on the process, but we discuss the other two below.
In recent years, automated tools to support system development have been springing up like mushrooms, and many of their vendors have been selling them on the pretext that the tools will solve all our development problems. Before there were automated development tools there were development methods, some of whose advocates were making similar claims. None of these claims is true. The fact is that before either methods or tools come process. Within the scope of the process principles described in this paper there is enormous variety based on the types of products, the type and size of organization, the variety of disciplines involved, and many other factors. It is ludicrous to claim that any one set of methods or tools can serve all of these diverse needs. Each organization must first define the details of the particular development process that meets its needs, it can then choose methods that will support that process, and only then can it make an informed decision on which tools (if any) will support the chosen methods and process.

Later we will discuss development process definition and improvement, and how it relates to the choice of methods and tools.

People

Before proceeding with all the technical factors involved in system development, it is well worth reminding ourselves that above and beyond whatever techniques, procedures, methods, and tools we choose, having the right people is paramount. What do we mean by the right people? They are hard-working and conscientious, and have skill, experience, and training in the applicable disciplines. Giving the right technical aids to such people will help them perform their work even better; without such people, all the technical gimmicks in the world will not save the project. Unfortunately, there are some managers who have not understood this message, and who believe that once they have invested in some expensive methods and tools, it’s OK to replace their “expensive” experienced people with new grads at half the salary. Wrong!

Project Planning (CMM level 2)

The parallel cooperative development paradigm can cause much discomfort in project planning and management. Gone are the nice comfortable milestones that declare each activity complete and closed before starting the next, and gone are the nice simple assignments of one task to one discipline, which can then pass its results on to the next. Ah, if only the world were really that way.

Figure 3 illustrates a planning matrix for a single development layer and a single discipline. Notice that a review of a particular activity does not signify the completion of that activity. Rather than having milestones that indicate completion, we have milestones that signify progress towards the goal of delivering a quality system. And instead of assigning the whole task to one discipline, we identify all the disciplines involved, and their specific responsibilities during each phase of the task.
Process, Methods, and Tools

In recent years, automated tools to support system development have been springing up like mushrooms, and many of their vendors have been selling them on the pretext that the tools will solve all our development problems. Before there were automated development tools there were development methods, some of whose advocates were making similar claims. None of these claims is true. The fact is that before either methods or tools come process. Within the scope of the process principles described in this paper there is enormous variety based on the types of products, the type and size of organization, the variety of disciplines involved, and many other factors. It is ludicrous to claim that any one set of methods or tools can serve all of these diverse needs. Each organization must first define the details of the particular development process that meets its needs, it can then choose methods that will support that process, and only then can it make an informed decision on which tools (if any) will support the chosen methods and process.

Later we will discuss development process definition and improvement, and how it relates to the choice of methods and tools.

People

Before proceeding with all the technical factors involved in system development, it is well worth reminding ourselves that above and beyond whatever techniques, procedures, methods, and tools we choose, having the right people is paramount. What do we mean by the right people? They are hard-working and conscientious, and have skill, experience, and training in the applicable disciplines. Giving the right technical aids to such people will help them perform their work even better; without such people, all the technical gimmicks in the world will not save the project. Unfortunately, there are some managers who have not understood this message, and who believe that once they have invested in some expensive methods and tools, it’s OK to replace their “expensive” experienced people with new grads at half the salary. Wrong!

Project Planning (CMM level 2)

The parallel cooperative development paradigm can cause much discomfort in project planning and management. Gone are the nice comfortable milestones that declare each activity complete and closed before starting the next, and gone are the nice simple assignments of one task to one discipline, which can then pass its results on to the next. Ah, if only the world were really that way.

Figure 3 illustrates a planning matrix for a single development layer and a single discipline. Notice that a review of a particular activity does not signify the completion of that activity. Rather than having milestones that indicate completion, we have milestones that signify progress towards the goal of delivering a quality system. And instead of assigning the whole task to one discipline, we identify all the disciplines involved, and their specific responsibilities during each phase of the task.

Hadley
System Development

©1992, Derek J. Hadley, System Strategies

Project Tracking and Oversight (CMM level 2)

The effectiveness of this process area is very dependent on how well the planning process was carried out, so to add to the difficulty of defining milestones that signify degrees of completion rather than total completion, it is essential that these milestones are defined to be measurable. A very effective tool for such measurement is the peer review, which is discussed later. Another technique involves estimating the number of residual errors and using this as a measure of progress. The technique depends on keeping good error statistics.

The real test of this process area occurs when progress deviates significantly from the plan, and corrective action must be taken. This can happen due to unforeseen difficulties in development, unavailability of expected people power, changes in requirements, and many other causes. If milestones are well defined and tracked, then deviations will be recognized early, and those same effective planning techniques can be used to replan based on the new data. An informed report on the impact of the deviation, before it actually happens, can then be provided to management and the customer.

Development Layer: __________________________
Discipline: __________________________

<table>
<thead>
<tr>
<th>Activity</th>
<th>Requirements Analysis</th>
<th>Concept Development</th>
<th>Initial Design</th>
<th>Firm Design</th>
<th>Specific Tasks, Schedules, and Milestones</th>
<th>Implementation</th>
<th>Test Deployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Startup</td>
<td>Requirements Review</td>
<td>Design/No Reviews</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concept Review</td>
<td>Design/No Reviews</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial Design Review</td>
<td>Design/No Reviews</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Firm Design Review</td>
<td>Specific Tasks, Schedules, and Milestones</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation Review</td>
<td>Implementation Review</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Activity/Event Matrix for One Layer & One Discipline

©1992, Derek J. Hadley, System Strategies

Hadley
System Development
Configuration Management (CMM level 2)

The essence of configuration management (CM) is the precise definition of reconstructable baselines throughout the project. The baselines must include all parts of the development: hardware, software, firmware, development equipment and environment, test equipment and environment, and documentation. Automated support is invaluable for CM, and numerous software packages exist to provide it.

Parallel development makes the need for intermediate baselines very clear. The recognition that everything is in a state of flux throughout development underscores the need for baselines other than the obvious ones for deliverable configurations. The planning process should include definitions of these intermediate baselines, which will typically coincide with the intermediate milestones.

Quality Assurance (CMM level 2)

Quality is the responsibility of everyone on the project, not just the QA group. It is not something that can be added later, but must be built into every detail of the development. A well-defined process provides a framework for quality to be built around. If everyone knows what is expected of them, then they can concentrate on performing those tasks to high standards of quality. Without clear task definitions it is hard to know what constitutes good quality and what does not.

The process must include written standards, and the task of the QA group is to monitor that the process and the standards really are being followed.

Requirements Management (CMM level 2)

Earlier, we described the pervasive nature of requirements, and this nature underscores the need for requirements to be well defined and controlled. It is customary to think of system requirements, hardware requirements, and software requirements as just the incoming requirements to those three areas, nicely and cleanly bundled. Figure 1 makes it clear that there are similar sets of requirements surrounding every one of the development layers, and all these sets are equally important.

Figure 4 is an entity-relationship diagram illustrating all the different types of requirements and their relationships. We are used to seeing requirements expressed as Required Features—collections of statements that have some common theme, which will usually be implemented as a single function. These features can be broken down into Primitive Requirements—indivisible statements that define the fundamental components of a required feature. In the DOD paradigm, each primitive statement contains a single shall. Required features also include Descriptive Narrative—statements that do not impose requirements on the system, but nevertheless contain important background information on the rationale for the requirements, the environment in which the system is to operate, and other very valuable material. Primitive requirements can broadly be divided into two types: Required Capabilities and Required Constraints. Required capabilities define what the system or subsystem must do—generally what it is to provide on its outputs; required constraints define restrictions on how the system or subsystem may do it—they are divided into numerous subtypes as shown in Figure 4. In the figure, each of these subtypes has various attributes listed with it, but in practice most of these attributes would be promoted to further subtypes with unique attributes of their own. Notice that Figure 4 makes no assumptions...
Configuration Management (CMM level 2)

The essence of configuration management (CM) is the precise definition of reconstructable baselines throughout the project. The baselines must include all parts of the development: hardware, software, firmware, development equipment and environment, test equipment and environment, and documentation. Automated support is invaluable for CM, and numerous software packages exist to provide it.

Parallel development makes the need for intermediate baselines very clear. The recognition that everything is in a state of flux throughout development underscores the need for baselines other than the obvious ones for deliverable configurations. The planning process should include definitions of these intermediate baselines, which will typically coincide with the intermediate milestones.

Quality Assurance (CMM level 2)

Quality is the responsibility of everyone on the project, not just the QA group. It is not something that can be added later, but must be built into every detail of the development. A well-defined process provides a framework for quality to be built around. If everyone knows what is expected of them, then they can concentrate on performing those tasks to high standards of quality. Without clear task definitions it is hard to know what constitutes good quality and what does not.

The process must include written standards, and the task of the QA group is to monitor that the process and the standards really are being followed.

Requirements Management (CMM level 2)

Earlier, we described the pervasive nature of requirements, and this nature underscores the need for requirements to be well defined and controlled. It is customary to think of system requirements, hardware requirements, and software requirements as just the incoming requirements to those three areas, nicely and cleanly bundled. Figure 1 makes it clear that there are similar sets of requirements surrounding every one of the development layers, and all these sets are equally important.

Figure 4 is an entity-relationship diagram illustrating all the different types of requirements and their relationships. We are used to seeing requirements expressed as Required Features—collections of statements that have some common theme, which will usually be implemented as a single function. These features can be broken down into Primitive Requirements—indivisible statements that define the fundamental components of a required feature. In the DOD paradigm, each primitive statement contains a single shall. Required features also include Descriptive Narrative—statements that do not impose requirements on the system, but nevertheless contain important background information on the rationale for the requirements, the environment in which the system is to operate, and other very valuable

Hatley ©1992, Derek J. Hatley, System Strategies

326 System Development
about the sources of the requirements: they can arise from within or outside the project, and those from outside can come from many sources as well as the customer.

The Requirement Deriving and Requirement Decomposing relationships reflect some of the design activities. Requirement deriving takes place when a design decision is made that a primitive requirement will be allocated to more than one design module. For this to take place, an algorithm must be chosen to implement the primitive requirement, and this creates a new set of primitive requirements—Derived Requirements. Requirement decomposing takes place when a required feature is allocated to more than one design module, and it is therefore divided into two or more subfeatures and/or primitives. In this case, no new requirements are generated: the original requirements are simply separated from each other.

It is clear from this synopsis of requirements that requirements management is no small task. Every one of the requirements types and their relationships must be recorded, and in a typical modern project there will be several thousand primitive requirements. To handle this size and complexity it is virtually essential that the requirements (and everything else in the development) should be captured in a project database. A diagram such as Figure 4 can provide the starting point for developing such a database.

In the parallel development environment a project database is particularly appropriate. Like the environment, the database is inherently non-sequential, and it can be populated in any convenient sequence. Progress can be measured in terms of how completely the database is populated. The non-sequential nature of the project database also lends itself to change management, and to determining change impact. Further benefits of the project database are that it supports traceability, and it can provide an integrating foundation for CASE tools.

Subcontract Management (CMM level 2)

The usual criteria for subcontract management, as stated in the CMM, apply in the parallel development environment: appropriate subcontractor qualifications; clearly defined standards, procedures and requirements for the subcontract; clearly understood commitments between the contractor and subcontractor; tracking of actual results against the commitments. It is very helpful if the subcontractor too follows a parallel development process, and while this cannot usually be mandated, this size and complexity is virtually essential that the requirements (and everything else in the development) should be captured in a project database. A diagram such as Figure 4 can provide the starting point for developing such a database.

Peer Reviews (CMM level 3)

Peer reviews are widely recognized as one of the most effective techniques for improving development. Peer reviews are unlike most other kinds of reviews in that they seek to discover errors, but not to find solutions for them: testing is based on exactly the same principle, and in fact, peer reviews are best thought of as part of the testing process. Their great strength as a testing tool is that they are applied to items that are not otherwise amenable to testing, such as requirements statements and analyses. They therefore facilitate finding development errors early, with the corresponding dramatic reductions in the cost of fixing the errors.

Peer reviews are eminently compatible with the parallel, cooperative development process, and they encourage participation by multiple disciplines. When selecting the participants for a peer review, it is important to include the "suppliers" and the "customers" of the item under review: for example, a review of top level software requirements (prepared by the Detailed System Design layer) should include representation from the higher system design layer (supplier) and the Top Level Software Design layer (customer).

Intergroup Coordination (CMM level 3)

This feature is so intrinsic to parallel cooperative development that it almost seems redundant to mention it. Its principles are that the project must be planned and conducted from a total systems perspective by one or more interdisciplinary teams. Not only must the responsibilities of each layer be well defined and understood, but the working interfaces between the layers must be equally well defined and understood. Teamwork is the essence of this feature.

Product Engineering (CMM level 3)

Development should have a product focus, whereby everyone involved is aware of the needs of the end user, and of the total system life cycle. This implies that the process and all the procedures will be defined and applied with this end user and life cycle focus in mind. User friendliness and maintainability will be high priorities, and state-of-the-art tools and methods will be used to ensure product longevity.

Again, this feature is a natural by-product of the parallel, cooperative development approach.

Integrated System Management (CMM level 3)

To lay the foundation for continuous process improvement and growth, it is essential that system management should have a perspective beyond individual projects. This means that, even though each project should be free to tailor the process to meet its own needs, this tailoring should be done within the bounds of organization-wide guidelines. It also means that project records are properly maintained, not just for the project itself, but as a source of information for future projects and for process analysis and improvement.

Training (CMM level 3)

Historically, training has been one of the most neglected areas in system development organizations, yet effective training can produce more immediate benefits than most other measures. The problem is that the costs of training, in both time and money, are very visible, while the costs of not training are buried in day-to-day inefficiencies, and can easily be overlooked or ignored. The sad fact is that the cost of not training is usually far greater than the cost of training.

There are many training options available: internal, external, seminars, video courses, special reading materials, local universities, consultants, and so on. A training program should be
about the sources of the requirements: they can arise from within or outside the project, and those from outside can come from many sources as well as the customer.

The Requirement Deriving and Requirement Decomposing relationships reflect some of the design activities. Requirement deriving takes place when a design decision is made that a primitive requirement will be allocated to more than one design module. For this to take place, an algorithm must be chosen to implement the primitive requirement, and this creates a new set of primitive requirements—Derived Requirements. Requirement decomposing takes place when a required feature is allocated to more than one design module, and it is therefore divided into two or more subfeatures and/or primitives. In this case, no new requirements are generated: the original requirements are simply separated from each other.

It is clear from this synopsis of requirements that requirements management is no small task. Every one of the requirements types and their relationships must be recorded, and in a typical modern project there will be several thousand primitive requirements. To handle this size and complexity it is virtually essential that the requirements (and everything else in the development) should be captured in a project database. A diagram such as Figure 4 can provide the starting point for developing such a database.

In the parallel development environment a project database is particularly appropriate. Like the environment, the database is inherently non-sequential, and it can be populated in any convenient sequence. Progress can be measured in terms of how completely the database is populated. The non-sequential nature of the project database also lends itself to change management, and to determining change impact. Further benefits of the project database are that it supports traceability, and it can provide an integrating foundation for CASE tools.

Subcontract Management (CMM level 2)

The usual criteria for subcontract management, as stated in the CMM, apply in the parallel development environment: appropriate subcontractor qualifications; clearly defined standards, procedures and requirements for the subcontract; clearly understood commitments between the contractor and subcontractor; tracking of actual results against the commitments. It is very helpful if the subcontractor too follows a parallel development process, and while this cannot usually be mandated, it is therefore divided into two or more subfeatures and/or primitives. In this case, no new requirements are generated: the original requirements are simply separated from each other.

Peer Reviews (CMM level 3)

Peer reviews are widely recognized as one of the most effective techniques for improving development. Peer reviews are unlike most other kinds of reviews in that they seek to discover errors, but not to find solutions for them: testing is based on exactly the same principle, and in fact, peer reviews are best thought of as part of the testing process. Their great strength as a testing tool is that they are applied to items that are not otherwise amenable to testing, such as requirements statements and analyses. They therefore facilitate finding development errors early, with the corresponding dramatic reductions in the cost of fixing the errors.

Peer reviews are eminently compatible with the parallel, cooperative development process, and they encourage participation by multiple disciplines. When selecting the participants for a peer review, it is important to include the "suppliers" and the "customers" of the item under review: for example, a review of top level software requirements (prepared by the Detailed System Design layer) should include representation from the higher system design layer (supplier) and the Top Level Software Design layer (customer).

Intergroup Coordination (CMM level 3)

This feature is so intrinsic to parallel cooperative development that it almost seems redundant to mention it. Its principles are that the project must be planned and conducted from a total systems perspective by one or more interdisciplinary teams. Not only must the responsibilities of each layer be well defined and understood, but the working interfaces between the layers must be equally well defined and understood. Teamwork is the essence of this feature.

Product Engineering (CMM level 3)

Development should have a product focus, whereby everyone involved is aware of the needs of the end user, and of the total system life cycle. This implies that the process and all the procedures will be defined and applied with this end user and life cycle focus in mind. User friendliness and maintainability will be high priorities, and state-of-the-art tools and methods will be used to ensure product longevity.

Again, this feature is a natural by-product of the parallel, cooperative development approach.

Integrated System Management (CMM level 3)

To lay the foundation for continuous process improvement and growth, it is essential that system management should have a perspective beyond individual projects. This means that, even though each project should be free to tailor the process to meet its own needs, this tailoring should be done within the bounds of organization-wide guidelines. It also means that project records are properly maintained, not just for the project itself, but as a source of information for future projects and for process analysis and improvement.

Training (CMM level 3)

Historically, training has been one of the most neglected areas in system development organizations, yet effective training can produce more immediate benefits than most other measures. The problem is that the costs of training, in both time and money, are very visible, while the costs of not training are buried in day-to-day inefficiencies, and can easily be overlooked or ignored. The sad fact is that the cost of not training is usually far greater than the cost of training.

There are many training options available: internal, external, seminars, video courses, special reading materials, local universities, consultants, and so on. A training program should be...
put together that covers both the long term and short term needs, and provides both for the needs of the organization and for the professional growth of the individual.

Since the development process itself should be customized to the needs of the organization, training in the process should likewise be customized and provided in-house (there are several consulting and training organizations that offer packaged training of their packaged development process and tool set—beware!). Other more specialized subjects, such as specific programming languages, can readily be taught by outside sources. The important principles are that a rational training program should be developed that is highly focused on the specific needs of the organization and the individual, and that the training sources should be highly qualified in their subjects.

Organization Process Definition and Focus (CMM level 3)

It should be clear from reading this paper so far that a process such as the one described does not happen by accident. It requires careful definition, planning, training, and support. It is highly recommended that a process team, not necessarily full-time, be established, that has the clearly defined responsibility for the care and feeding of the process. Like quality, implementing the process is everyone's responsibility, but the process group oversees it, ensures that it is running effectively, gathers data on its performance, and plans and facilitates process improvements. Also like quality, there must be a commitment to the development process from the highest levels of management right on down, and this commitment must comprise more than just lip service: it requires the allocation of real resources such as training, the time of the process group, and automated tools where the process calls for them.

Quality Management (CMM level 4)

Although, as we have said several times, quality is everyone's responsibility, it does need nurturing, especially when there are quality problems. The foundation for effective quality management is measurement, against which quality can be assessed. These measurements should be centered on deviations from customer requirements—both external and internal customers. Dimensional variations in mechanical parts, and residual errors in delivered software are examples of appropriate quality measurements.

The ideas of quality as conformance with requirements, and of in-house customers and suppliers, are central to Total Quality Management (TQM) (which has become something of a fad in recent years, but which nevertheless has great merit). For example, if you are a systems engineer writing a software requirements specification, then the software group is your customer, and you should make sure that you understand and comply with their requirements for a requirements specification. Similarly, if you are a software engineer developing a software package to be delivered to the systems group for integration into a larger system, then the systems group is your customer, and you should be aware of and conform with their requirements on the software package.

Process Measurement and Analysis (CMM level 4)

Having established measurements such as those mentioned above, they should be analyzed and correlated with productivity and development cycle time. It is important to understand that the measurements should not and need not be elaborate. If peer reviews are included as a part of the testing process, as discussed earlier, then errors discovered in testing, correlated with type and size of item tested, resources applied to development of the item, and resources applied to testing the item can provide most of the measurements and analysis needed. Once a historical record of such measurements and analyses has been established, it can be used to investigate the effects of changes in the process.

Process Change Management (CMM level 5)

This and the rest of the CMM level 5 KPAs are areas that few organizations have yet achieved, except in sporadic bursts of energy. Like everything else in the CMM, it is dependent on all the preceding KPAs being in place and working effectively. The parallel cooperative development approach tends to promote an effective change process. By definition, everyone is involved in and aware of the process, so everyone will be naturally inclined to look for improvements in the process. Ongoing process improvement cannot be decreed from above: it requires "buy-in" by everyone involved. The role of management is to establish and maintain the means for process improvement suggestions to be channeled into the system. The suggestions should be evaluated (preferably with the involvement of the originator(s)), and then either incorporated (with or without change), or deferred, or rejected—all according to the results of the evaluation. All of this must be based on quantitative measurements whose results are irrefutable. Only at this level of process maturity is all this likely to be achievable.

Technology Innovation (CMM level 5)

Notice that this KPA does not appear until level 5. Contrast this with the tendency of many organizations to decide, as a first step, to "bring in some CASE tools to fix our problems". If you review all the preceding KPAs, you will see that they provide a myriad applications for new technology, many of which are far beyond the reach of existing CASE tools.

Technology innovation goes hand in hand with the previous KPA. As the process is originally being developed, technology can be progressively introduced to support it. As process changes are evaluated, a part of the evaluations should be compatibility with existing technology, and as they are introduced, new technology should be investigated to support them. In addition, regardless of process improvements, the evolution of technology must be monitored to always take advantage of the best that is available.

Defect Prevention (CMM level 5)

This KPA too goes hand in hand with process change management. Analysis of the process measurements should routinely look for repeated errors, and correlate them with the process. Process changes can then be investigated that will prevent such errors before they occur.
put together that covers both the long term and short term needs, and provides both for the needs of the organization and for the professional growth of the individual.

Since the development process itself should be customized to the needs of the organization, training in the process should likewise be customized and provided in-house (there are several consulting and training organizations that offer packaged training of their packaged development process and tool set—beware!) Other more standardized subjects, such as specific programming languages, can readily be taught by outside sources. The important principles are that a rational training program should be developed that is highly focused on the specific needs of the organization and the individual, and that the training sources should be highly qualified in their subjects.

Organization Process Definition and Focus (CMM level 3)

It should be clear from reading this paper so far that a process such as the one described does not happen by accident. It requires careful definition, planning, training, and support. It is highly recommended that a process team, not necessarily full-time, should be established, that has the clearly defined responsibility for the care and feeding of the process. Like quality, implementing the process is everyone's responsibility, but the process group oversees it, ensures that it is running effectively, gathers data on its performance, and plans and facilitates process improvements. Also like quality, there must be a commitment to the development process from the highest levels of management right on down, and this commitment must comprise more than just lip service: it requires the allocation of real resources such as training, the time of the process group, and automated tools where the process calls for them.

Quality Management (CMM level 4)

Although, as we have said several times, quality is everyone's responsibility, it does need nurturing, especially when there are quality problems. The foundation for effective quality management is measurement, against which quality can be assessed. These measurements should be centered on deviations from customer requirements—both external and internal customers. Dimensional variations in mechanical parts, and residual errors in delivered software are examples of appropriate quality measurements.

The ideas of quality as conformance with requirements, and of in-house customers and suppliers, are central to Total Quality Management (TQM) (which has become something of a fad in recent years, but which nevertheless has great merit). For example, if you are a systems engineer writing a software requirements specification, then the software group is your customer, and you should make sure that you understand and comply with their requirements for a requirements specification. Similarly, if you are a software engineer developing a software package to be delivered to the systems group for integration into a larger system, then the systems group is your customer, and you should be aware of and conform with their requirements on the software package.

Process Measurement and Analysis (CMM level 4)

Having established measurements such as those mentioned above, they should be analyzed and correlated with productivity and development cycle time. It is important to understand that the measurements should not and need not be elaborate. If peer reviews are included as a part of the testing process, as discussed earlier, then errors discovered in testing, correlated with type and size of item tested, resources applied to development of the item, and resources applied to testing the item can provide most of the measurements and analysis needed. Once a historical record of such measurements and analyses has been established, it can be used to investigate the effects of changes in the process.

Process Change Management (CMM level 5)

This and the rest of the CMM level 5 KPAs are areas that few organizations have yet achieved, except in sporadic bursts of energy. Like everything else in the CMM, it is dependent on all the preceding KPAs being in place and working effectively. The parallel cooperative development approach tends to promote an effective change process. By definition, everyone is involved in and aware of the process, so everyone will be naturally inclined to look for improvements in the process. Ongoing process improvement cannot be decreed from above: it requires “buy-in” by everyone involved. The role of management is to establish and maintain the means for process improvement suggestions to be channeled into the system. The suggestions should be evaluated (preferably with the involvement of the originator(s)), and then either incorporated (with or without change), or deferred, or rejected—all according to the results of the evaluation. All of this must be based on quantitative measurements whose results are irrefutable. Only at this level of process maturity is all this likely to be achievable.

Technology Innovation (CMM level 5)

Notice that this KPA does not appear until level 5. Contrast this with the tendency of many organizations to decide, as a first step, to “bring in some CASE tools to fix our problems”. If you review all the preceding KPAs, you will see that they provide a myriad applications for new technology, many of which are far beyond the reach of existing CASE tools.

Technology innovation goes hand in hand with the previous KPA. As the process is originally being developed, technology can be progressively introduced to support it. As process changes are evaluated, a part of the evaluations should be compatibility with existing technology, and as they are introduced, new technology should be investigated to support them. In addition, regardless of process improvements, the evolution of technology must be monitored to always take advantage of the best that is available.

Defect Prevention (CMM level 5)

This KPA too goes hand in hand with process change management. Analysis of the process measurements should routinely look for repeated errors, and correlate them with the process. Process changes can then be investigated that will prevent such errors before they occur.
PROCESS DEFINITION AND IMPROVEMENT

Figure 1 illustrates the general principles of the parallel, cooperative development process; reference [1] describes it in more detail; and this paper, using the SEI CMM as a guide, puts it into the broader perspective of all the factors involved—organization and management, as well as technical. It remains, however, for each development organization to define its own specific version of the process, adapted to its own products, organization, size, and culture.

The definition process itself is no small task. General “arm-waving” definitions will not suffice. The process should be written in enough detail that a newly hired individual, with appropriate qualifications, and a few years of development experience, would clearly understand how he or she is expected to carry out any of the tasks that are likely to be assigned. If there is a wide variety of sizes and types of development, then several versions of the process might be necessary.

It is useful to view process definition itself as a system development activity: all the principles that apply to developing deliverable systems also apply to developing the development process. It has requirements (to meet the needs of management, development staff, customers, cost, quality…), requirements analysis (to determine if the requirements are complete, consistent, unambiguous, reasonable, compatible with available resources…), design (defining tasks, information flows, and material flows; reusing existing procedures…), implementation (applying the proposed processes to pilot projects), test (evaluating the results of the pilot projects against expectations, and correcting deficiencies), deployment (launching the process into the development organization, formalizing the procedures…) and maintenance (ongoing monitoring, corrections, and improvements).

Another useful principle to follow in constructing the development process is the “black box” design approach, whereby each layer in the process defines only the external inputs and outputs as requirements for the modules in the layer below. Exceptions to this principle should only be made for legitimate and well-justified design constraints—these are “white box” requirements. Following these principles forces cooperative development, because black box requirements cannot be defined without consulting the recipients to determine if the requirements are reasonable. The approach to follow is that a given requirement should be passed through without change from layer to layer until it must be assigned to more than one module, at which point some derived requirements must be produced from it. In this way, a requirement is passed down to the lowest layer possible before it is implemented.

It is essential that, whatever the variations on the process, traceability is well supported. Traceability is needed between requirements and requirements, between requirements and design, and between design and design. External requirements must be traced through to the design modules to which they are allocated and to any associated derived requirements; design modules must be traced to all of their descendent modules. All of these traces must be bi-directional, so that both sources and destinations can be traced. The requirements and architecture methods described in [5] are unique in their coverage of both requirements and design traceability throughout system, software, and hardware development.

Structured analysis works well as a tool for defining the process in detail. Its inherent non-sequential, data-triggered paradigm fits the parallel cooperative development model, in that all processes and sub-processes are always active and waiting for inputs to arrive. The level of detail illustrated in this paper typically needs to be taken two or three levels deeper to fully define a specific organization’s process. Remember, when using SA, the diagrams alone are not sufficient: they must be backed up by detailed specifications of the flows and processes. Note too that (contrary to traditional guidelines) it is beneficial to write process specifications at all levels, not just at the primitive level. There is usually much more to say about a given process than is contained in its child diagram: for example, why that particular overall decomposition of the process was chosen.

The foundation for effective process improvement lies in starting with a well-defined and documented process, and following the steps outlined above will provide this. Beyond this, the principles of Integrated System Management, discussed earlier, are necessary to discover weaknesses in the process, and to evaluate changes. One further piece of advice: a development process should never be allowed to rest on its laurels. Remember that process maturity can regress as well as progress, and it will if left to its own devices. An active process improvement program is needed just to ensure that the status quo is maintained.

CONCLUSIONS

In the past, the system development process has been treated, at best, in a piecemeal way, with each discipline making some effort to cover its own needs, but little being done to integrate the whole process. With the increasing functional size and complexity of modern systems, it is imperative that this integrated approach is taken. The SEI CMM provides a framework for a sound software process, and it can be extended to cover the whole system process. The parallel, cooperative development approach recognizes the reality that sequential approaches to development are counter-productive, and that multi-disciplinary development teams are essential. Methods and tools serve to support the process, they do not provide a process in themselves: they should therefore be used with caution, and a clear understanding of what they can and cannot do. The requirements and architecture methods described in [5] are unique in their coverage of the total system process, and some CASE tools are now starting to support them completely.
PROCESS DEFINITION AND IMPROVEMENT

Figure 1 illustrates the general principles of the parallel, cooperative development process; reference [1] describes it in more detail; and this paper, using the SEI CMM as a guide, puts it into the broader perspective of all the factors involved—organization and management, as well as technical. It remains, however, for each development organization to define its own specific version of the process, adapted to its own products, organization, size, and culture.

The definition process itself is no small task. General “arm-waving” definitions will not suffice. The process should be written in enough detail that a newly hired individual, with appropriate qualifications, and a few years of development experience, would clearly understand how he or she is expected to carry out any of the tasks that are likely to be assigned. If there is a wide variety of sizes and types of development, then several versions of the process might be necessary.

It is useful to view process definition itself as a system development activity: all the principles that apply to developing deliverable systems also apply to developing the development process. It has requirements (to meet the needs of management, development staff, customers, cost, quality...), requirements analysis (to determine if the requirements are complete, consistent, unambiguous, reasonable, compatible with available resources...), design (defining tasks, information flows, and material flows; reusing existing procedures...), implementation (applying the proposed processes to pilot projects), test (evaluating the results of the pilot projects against expectations, and correcting deficiencies), deployment (launching the process into the development organization, formalizing the procedures...) and maintenance (ongoing monitoring, corrections, and improvements).

Another useful principle to follow in constructing the development process is the “black box” design approach, whereby each layer in the process defines only the external inputs and outputs as requirements for the modules in the layer below. Exceptions to this principle should only be made for legitimate and well-justified design constraints—these are “white box” requirements. Following these principles forces cooperative development, because black box requirements cannot be defined without consulting the recipients to determine if the requirements are reasonable. The approach to follow is that a given requirement should be passed through without change from layer to layer until it must be assigned to more than one module, at which point some derived requirements must be produced from it. In this way, a requirement is passed down to the lowest layer possible before it is implemented.

It is essential that, whatever the variations on the process, traceability is well supported. Traceability is needed between requirements and requirements, between requirements and design, and between design and design. External requirements must be traced through to the design modules to which they are allocated and to any associated derived requirements; design modules must be traced to all of their descendent modules. All of these traces must be bi-directional, so that both sources and destinations can be traced. The requirements and architecture methods described in [5] are unique in their coverage of both requirements and design traceability throughout system, software, and hardware development.

Structured analysis works well as a tool for defining the process in detail. Its inherent non-sequential, data-triggered paradigm fits the parallel cooperative development model, in that all processes and sub-processes are always active and waiting for inputs to arrive. The level of detail illustrated in this paper typically needs to be taken two or three levels deeper to fully define a specific organization’s process. Remember, when using SA, the diagrams alone are not sufficient: they must be backed up by detailed specifications of the flows and processes. Note too that (contrary to traditional guidelines) it is beneficial to write process specifications at all levels, not just at the primitive level. There is usually much more to say about a given process than is contained in its child diagram: for example, why that particular overall decomposition of the process was chosen.

The foundation for effective process improvement lies in starting with a well-defined and documented process, and following the steps outlined above will provide this. Beyond this, the principles of Integrated System Management, discussed earlier, are necessary to discover weaknesses in the process, and to evaluate changes. One further piece of advice: a development process should never be allowed to rest on its laurels. Remember that process maturity can regress as well as progress, and it will if left tots own devices. An active process improvement program is needed just to ensure that the status quo is maintained.

CONCLUSIONS

In the past, the system development process has been treated, at best, in a piecemeal way, with each discipline making some effort to cover its own needs, but little being done to integrate the whole process. With the increasing functional size and complexity of modern systems, it is imperative that this integrated approach is taken. The SEI CMM provides a framework for a sound software process, and it can be extended to cover the whole system process. The parallel, cooperative development approach recognizes the reality that sequential approaches to development are counter-productive, and that multi-disciplinary development teams are essential. Methods and tools serve to support the process, they do not provide a process in themselves: they should therefore be used with caution, and a clear understanding of what they can and cannot do. The requirements and architecture methods described in [5] are unique in their coverage of the total system process, and some CASE tools are now starting to support them completely.
REFERENCES


DSP Techniques for Real-Time Applications

Patrick Heath
Motorola
Austin, Texas

Patrick Heath is Motorola's Advanced MCU technical marketing manager. Among other tasks, he writes software demonstrating how to take advantage of new MCU features. He is also responsible for Motorola's strategy of low cost evaluation boards. After receiving separate Bachelor's degrees in computer science, mathematics, and business administration from Graceland College, and a Master's degree in computer science from the University of Missouri, Rolla, Heath worked for IBM in Lexington, Kent, before joining Motorola in Austin, Texas.