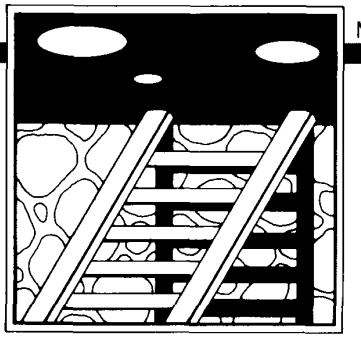

Characterizing the Software Process: A Maturity Framework

Watts S. Humphrey, *Software Engineering Institute*

Software quality and productivity must improve. But where to start? This model helps organizations identify their highest priority problems and start making improvements.

The amount of money spent on software in the US grows approximately 12 percent each year, and the demand for added software functions grows even faster. Software is a major and increasing portion of US Defense Dept. procurement costs, and software often adversely affects the schedules and effectiveness of weapons systems.

In recognition of the need to improve the development of military software, the Defense Dept. has launched several initiatives on software reliability, maintainability, and testing, including the Ada Joint Program Office and the STARS program. The Defense Dept. formed the Software Engineering Institute at Carnegie Mellon University in 1984 to establish standards of excellence for software engineering and to accelerate the transition of advanced technology and methods into practice.

One SEI project is to provide the Defense Dept. with some way to characterize the capabilities of software-development organizations. The result is this software-process maturity framework, which can be used by any software organization to assess its own capabilities and identify the most important areas for improvement.

Ideal software process

It is worthwhile to examine the characteristics of a truly effective software process. First, it is predictable: Cost estimates and schedule commitments are met with reasonable consistency and the quality of the resulting products generally meet user needs.

Statistical control. The basic principle of software process management is that if

0740-7459/88/0300/0073/$01.00 ©1988 IEEE

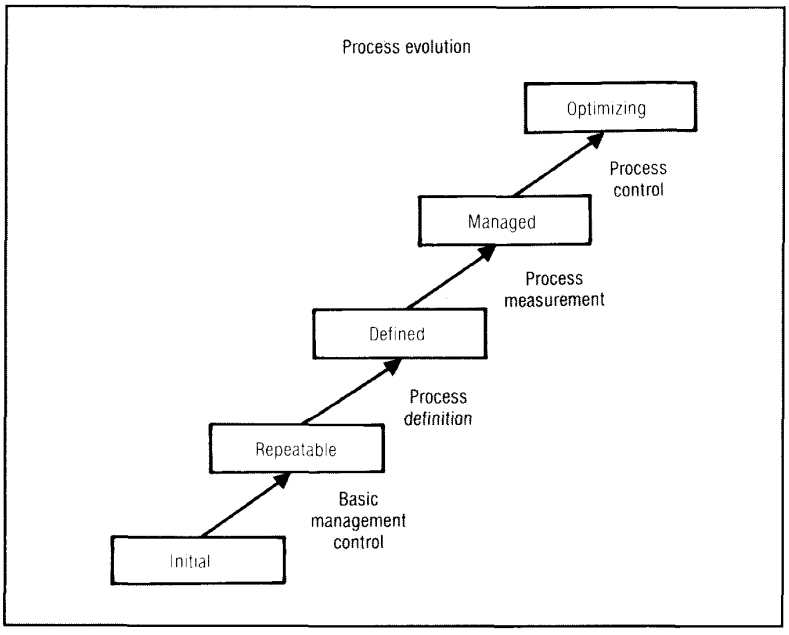

Figure 1. The five levels of process maturity.

the development process is under statistical control, a consistently better result can be achieved only by improving the process. If the process is not under statistical control, sustained progress is not possible until it is.[1]

When a process is under statistical control, repeating the work in roughly the same way will produce roughly the same result.

W.E. Deming, in his work with the Japanese industry after World War II, applied the concepts of statistical process control to industry.[1] While there are important differences, these concepts are just as applicable to software as they are to automobiles, cameras, wristwatches, and steel. A software-development process that is under statistical control will produce the desired results within the anticipated limits of cost, schedule, and quality.

Measurement. The basic principle behind statistical control is measurement. As Lord Kelvin said a century ago, " . . . when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science. . . ."[2]

There are several factors to consider in measuring the programming process. Perhaps most important is that the mere act of measuring human processes changes them. Since people's fears and motivations are involved, the results must be viewed in a different light than data on natural phenomena.

It is also essential to limit the measurements to those few items that will really be used. Measurements are both expensive and disruptive; overzealous measuring can degrade the processes we are trying to improve.

Development-process improvement

An important first step in addressing software problems is to treat the entire development task as a process that can be controlled, measured, and improved. We define a process as a sequence of tasks that, when properly performed, produces the desired result. Clearly, a fully effective software process must consider the relationships of all the required tasks, the tools and methods used, and the skill, training, and motivation of the people involved.

To improve their software capabilities, organizations must take five steps:

(1) understand the current status of their development process or processes,

(2) develop a vision of the desired process,

(3) establish a list of required process improvement actions in order of priority,

(4) produce a plan to accomplish these actions, and

(5) commit the resources to execute the plan.

The maturity framework developed at the SEI addresses these five steps by characterizing a software process into one of five maturity levels. By establishing their organization's position in this maturity structure, software professionals and management can more readily identify those areas where improvement actions are most likely to produce results.

Process maturity levels

As Figure 1 shows, the five levels of process maturity are:

1. Initial. Until the process is under statistical control, no orderly progress in process improvement is possible.

2. Repeatable. The organization has achieved a stable process with a repeatable level of statistical control by initiating rigorous project management of commitments, cost, schedule, and changes.

3. Defined. The organization has defined the process, to ensure consistent implementation and provide a basis for better understanding of the process. At this point, advanced technology can usefully be introduced.

4. Managed. The organization has initiated comprehensive process measurements, beyond those of cost and schedule performance. This is when the most significant quality improvements begin.

5. Optimizing. The organization now has a foundation for continued improvement and optimization of the process.

These levels have been selected because they

- reasonably represent the actual historical phases of evolutionary improvement of real software organizations,
- represent a measure of improvement that is reasonable to achieve from the prior level,
- suggest interim improvement goals and progress measures, and
- make obvious a set of immediate improvement priorities, once an organization's status in this framework is known.

While there are many other elements to these maturity-level transitions, the basic

objective is to achieve a controlled and measured process as the scientific foundation for continuous improvement. This structure is intended to be used with an assessment and management methodology, as outlined in the box on pp. 76-77.

Initial Process

The Initial Process could properly be called ad hoc, and it is often even chaotic. Here, the organization typically operates without formalized procedures, cost estimates, and project plans. Tools are neither well integrated with the process nor uniformly applied. Change control is lax and there is little senior management exposure to or understanding of the problems and issues. Since problems are often deferred or even forgotten, software installation and maintenance often present serious problems.

While organizations at this level may have formal procedures for project control, there is no management mechanism to ensure they are used. The best test is to observe how such an organization behaves in a crisis. If it abandons established procedures and reverts to merely coding and testing, it is likely to be at the Initial Process level. After all, if the techniques and methods are appropriate, they must be used in a crisis and if they are not appropriate, they should not be used at all.

One reason organizations behave chaotically is that they have not gained sufficient experience to understand the consequences of such behavior. Because many effective software actions such as design and code reviews or test data analysis do not appear to directly support shipping the product, they seem expendable.

It is much like driving an automobile. Few drivers with any experience will continue driving for very long when the engine warning light comes on, regardless of their rush. Similarly, most drivers starting on a new journey will, regardless of their hurry, pause to consult a map. They have learned the difference between speed and progress.

In software, coding and testing seem like progress, but they are often only wheel-spinning. While they must be done, there is always the danger of going in the wrong direction. Without a sound plan and a thoughtful analysis of the problems, there is no way to know.

Organizations at the Initial Process level can improve their performance by instituting basic project controls. The most important are:

• Project management. The fundamental role of a project-management system is to ensure effective control of commitments. This requires adequate preparation, clear responsibility, a public declaration, and a dedication to performance.[3]

For software, this starts with an understanding of the job's magnitude. In any but the simplest projects, a plan must then be developed to determine the best schedule and the resources required. In the absence of such an orderly plan, no commitment can be better than an educated guess.

A disciplined software-development organization must have senior management oversight.

• Management oversight. A disciplined software-development organization must have senior management oversight. This includes review and approval of all major development plans before official commitment.

Also, a quarterly review should be conducted of facility-wide process compliance, installed-quality performance, schedule tracking, cost trends, computing service, and quality and productivity goals by project. The lack of such reviews typically results in uneven and generally inadequate implementation of the process as well as in frequent overcommitments and cost surprises.

• Quality assurance. A quality-assurance group is charged with assuring management that the software-development work is actually done the way it is supposed to be done. To be effective, the assurance organization must have an independent reporting line to senior management and sufficient resources to monitor performance of all key planning, implementation, and verification activities. This generally requires an organization of about 5 to 6 percent the size of the development organization.

• Change control. Control of changes in software development is fundamental to business and financial control as well as to technical stability. To develop quality software on a predictable schedule, the requirements must be established and maintained with reasonable stability throughout the development cycle. Changes will have to be made, but they must be managed and introduced in an orderly way.

While occasional requirements changes are needed, historical evidence demonstrates that many of them can be deferred and phased in later. If all changes are not controlled, orderly design, implementation, and testing is impossible and no quality plan can be effective.

Repeatable Process

The Repeatable Process has one important strength over the Initial Process: It provides commitment control.

This is such an enormous advance over the Initial Process that the people in the organization tend to believe they have mastered the software problem. They do not realize that their strength stems from their prior experience at similar work. Organizations at the Repeatable Process level thus face major risks when they are presented with new challenges.

Examples of the changes that represent the highest risk at this level are:

• New tools and methods will likely affect how the process is performed, thus destroying the relevance of the intuitive historical base on which the organization relies. Without a defined process framework in which to address these risks, it is even possible for a new technology to do more harm than good.

• When the organization must develop a new kind of product, it is entering new territory. For example, a software group that has experience developing compilers will likely have design, scheduling, and estimating problems if assigned to write a control program. Similarly, a group that has developed small, self-contained programs will not understand the interface and integration issues involved in large-scale projects. These changes again

destroy the relevance of the intuitive historical basis for the organization's work.

• Major organization changes can be highly disruptive. In the Repeatable Process organization, a new manager has no orderly basis for understanding what is going on and new team members must learn the ropes through word of mouth.

The key actions required to advance from the Repeatable Process to the Defined Process are:

1. Establish a process group. A process group is a technical group that focuses exclusively on improving the software-development process. In most software organizations, people are entirely devoted to product work. Until someone is given a full-time assignment to work on the process, little orderly progress can be made in improving it.

The responsibilities of process groups include defining the development process, identifying technology needs and opportunities, advising the projects, and conducting quarterly management reviews of process status and performance. Typically, the process group should be about 1 to 3 percent the size of the development organization. Because of the need for a nucleus of skills, groups smaller than about four are unlikely to be fully effective. Small organizations that lack the experience base to form a process group should address these issues through specially formed committees of experienced professionals or by retaining consultants.

2. Establish a software-development process architecture that describes the technical and management activities required for proper execution of the development process.[4] The architecture is a structural decomposition of the development cycle into tasks, each of which has entry criteria, functional descriptions, verification procedures, and exit criteria. The decomposition continues until each defined task is performed by an individual or single management unit.

3. If they are not already in place, introduce a family of software-engineering methods and technologies. These include design and code inspections, formal design methods, library-control systems, and comprehensive testing methods. Prototyping should also be considered, along with the adoption of modern implementation languages.

Defined Process

With the Defined Process, the organization has achieved the foundation for major and continuing progress. For example, the development group, when faced with a crisis, will likely continue to use the Defined Process. The foundation has now been established for examining the process and deciding how to improve it.

As powerful as the Defined Process is, it is still only qualitative: There is little data to indicate what is going on or how effective the process really is. There is considerable debate about the value of software-process measurements and the best ones to use. This uncertainty generally stems from a lack of process definition and the consequent confusion about the specific items to be measured. With a defined process, we can focus the measurements on specific tasks. The process architecture is thus an essential prerequisite to effective measurement.

The key steps[3,4] to advance to the Managed Process are:

1. Establish a minimum, basic set of process measurements to identify the quality and cost parameters of each process step. The objective is to quantify the relative costs and benefits of each major pro-

How to use this framework

This process-maturity structure is intended to be used with an assessment methodology and a management system.[1-3]

Assessment lets you identify the organization's specific maturity status. A management system establishes a structure for actually implementing the priority actions necessary to improve the organization. Once its position in this maturity structure is defined, the organization can concentrate on those items that will let it advance to the next level.

When, for example, a software organization does not have an effective project-planning system, it may be difficult or even impossible to introduce advanced methods and technology. Poor project planning generally leads to unrealistic schedules, inadequate resources, and frequent crises. In such circumstances, new methods are usually ignored, and the focus is on coding and testing.

Using this maturity framework, the SEI has developed an assessment questionnaire and methodology, a portion of which is shown in Figure A.[4,5] The questionnaire has been reviewed by more than 400 governmental and industrial organizations. Also, it has been completed by more than 50 programming professionals from nearly as many software organizations. A section of our questionsionals from nearly as many software organizations.

The SEI has also used the assessment methodology to conduct in-depth technical reviews of 25 programming projects in four large programming organizations.

Through this work, the assessment methodology and questionnaire have evolved, but the five-level maturity framework has remained essentially unchanged. We have found that it portrays, with reasonable accuracy, the status and problems as seen by the managers and professionals in the organizations reviewed.

These early results indicate that the model reasonably represents the state of such organizations and provides a mechanism to rapidly identify the key improvement issues they face. At this time, the data is too limited to provide any more detailed information as to maturity distribution by industry, organization size, or type of work.

References

1. W.S. Humphrey, *Managing for Innovation - Leading Technical People*, Prentice-Hall, Englewood Cliffs, N.J., 1987.
2. R.A. Radice et al., "A Programming Process Study," *IBM Systems J.*, Vol. 24, No. 2, 1985, pp. 91-101.
3. R.A. Radice et al., "A Programming Process Architecture," *IBM Systems J.*, Vol. 24, No. 2, 1985, pp. 79-90.

cess activity, such as the cost and yield of error detection and correction methods.

2. Establish a process database with the resources to manage and maintain it. Cost and yield data should be maintained centrally to guard against loss, to make it available for all projects, and to facilitate process quality and productivity analysis.

3. Provide sufficient process resources to gather and maintain this data and to advise project members on its use. Assign skilled professionals to monitor the quality of the data before entry in the database and to provide guidance on analysis methods and interpretation.

4. Assess the relative quality of each product and inform management where quality targets are not being met. An independent quality-assurance group should assess the quality actions of each project and track its progress against its quality plan. When this progress is compared with the historical experience on similar projects, an informed assessment generally can be made.

Managed Process

In advancing from the Initial Process via the Repeatable and Defined Processes to the Managed Process, software organizations typically will experience substantial quality improvements. The greatest potential problem with the Managed Process is the cost of gathering data. There are an enormous number of potentially valuable measures of software development and support, but such data is expensive to gather and maintain.

Therefore, approach data gathering with care and precisely define each piece of data in advance. Productivity data is generally meaningless unless explicitly defined. For example, the simple measure of lines of source code per development month can vary by 100 times of more, depending on the interpretation of the parameters. The code count could include only new and changed code or all shipped instructions. For modified programs, this can cause a ten-times variation. Similarly, you can use noncomment, nonblank lines, executable instructions, or equivalent assembler instructions, with variations again of up to seven times.[5] Management, test, documentation, and support personnel may or may not be counted when calculating labor months expended. Again, the variations can run at least as high as seven times.[6]

When different groups gather data but do not use identical definitions, the results are not comparable, even if it made sense to compare them. The tendency with such data is to use it to compare several groups and put pressure on those with the lowest ranking. This is a misapplication of process data.

First, it is rare that two projects are comparable by any simple measures. The variations in task complexity caused by different product types can exceed five to one. Similarly, the cost per line of code of small modifications is often two to three times that for new programs. The degree of requirements change can make an enormous difference, as can the design status of the base program in the case of enhancements.

Process data must not be used to compare projects or individuals. Its purpose is to illuminate the product being developed and to provide an informed basis for improving the process. When such data is used by management to evaluate individuals or teams, the reliability of the data itself will deteriorate. The US Constitution's Fifth Amendment, which protects against self-incrimination, is based on sound principles: Few people can be counted on to provide reliable data on

4. W.S. Humphrey and D.H. Kitson, "Preliminary Report on Conducting SEI-Assisted Assessments of Software Engineering Capability," Tech. Report SEI-87-TR-16, Software Eng. Inst., Pittsburgh, July 1987.

5. W.S. Humphrey and W.L. Sweet, "A Method for Assessing the Software Engineering Capability of Contractors," Tech. Report SEI-87-TR-23, Software Eng. Inst., Pittsburgh, Sept. 1987.

2.3. Data Management and Analysis

Data management deals with the gathering and retention of process metrics. Data management requires standardized data definitions, data management facilities, and a staff to ensure that data is promptly obtained, properly checked, accurately entered into the database, and effectively managed.

Analysis deals with the subsequent manipulation of the process data to answer questions such as, "Is there is a relatively high correlation between error densities found in test and those found in use?" Other types of analyses can assist in determining the optimum use of reviews and resources, the tools most needed, testing priorities, and needed education.

2.3.1. Has a managed and controlled process database been established for process metrics data across all projects?

2.3.2. Are the review data gathered during design reviews analyzed?

2.3.3. Is the error data from code reviews and tests analyzed to determine the likely distribution and characteristics of the errors remaining in the product?

2.3.4. Are analyses of errors conducted to determine their process related causes?

2.3.5. Is a mechanism used for error cause analysis?

2.3.6. Are the error causes reviewed to determine the process changes required to prevent them?

2.3.7. Is a mechanism used for initiating error prevention actions?

2.3.8. Is review efficiency analyzed for each project?

2.3.9. Is software productivity analyzed for major process steps?

Figure A. A portion of the SEI's assessment questionnaire.

their own performance.

The two fundamental requirements to advance from the Managed Process to the Optimizing Process are:

1. Support automatic gathering of process data. Some data cannot be gathered by hand, and all manually gathered data is subject to error and omission.
2. Use this data to both analyze and modify the process to prevent problems and improve efficiency.

Optimizing Process

In varying degrees, process optimization goes on at all levels of process maturity. With the step from the Managed to the Optimizing Process, however, there is a paradigm shift. Up to this point, software-development managers have largely focused on their products and will typically only gather and analyze data that directly relates to product improvement. In the Optimizing Process, the data is available to actually tune the process itself. With a little experience, management will soon see that process optimization can produce major quality and productivity improvements.

For example, many errors can be identified and fixed far more economically by code inspections than through testing. Unfortunately, there is little published data on the costs of finding and fixing errors.[7] However, I have developed a useful rule of thumb from experience: It takes about one to four working hours to find and fix a bug through inspections and about 15 to 20 working hours to find and fix a bug in function or system test. It is thus clear that testing is not a cost-effective way to find and fix most bugs.

However, some kinds of errors are either uneconomical or almost impossible to find except by machine. Examples are errors involving spelling and syntax, interfaces, performance, human factors, and error recovery. It would thus be unwise to eliminate testing completely because it provides a useful check against human frailties.

The data that is available with the Optimizing Process gives us a new perspective on testing. For most projects, a little analysis shows that there are two distinct activities involved. The first is the removal of bugs. To reduce this cost, inspections should be emphasized together with any other cost-effective techniques. The role of functional and system testing should then be changed to one of finding symptoms that are further explored to see if the bug is an isolated problem or if it indicates design problems that require more comprehensive analysis.

In the Optimizing Process, the organization has the means to identify the weakest elements of the process and fix them. At this point in process improvement, data is available to justify the application of technology to various critical tasks and numerical evidence is available on the effectiveness with which the process has been applied to any given product. We no longer need reams of paper to describe what is happening because simple yield curves and statistical plots provide clear and concise indicators. It is now possible to assure the process and hence have confidence in the quality of the resulting products.

People in the process. Any software-development process is dependent on the quality of the people who implement it. Even with the best people, however, there is always a limit to what they can accomplish. When engineers are already working 50 to 60 hours a week, it is hard to see how they could handle the vastly greater challenges of the future.

The Optimizing Process helps in several ways:

- It helps managers understand where help is needed and how best to provide the people with the support they require.
- It lets professionals communicate in concise, quantitative terms. This facilitates the transfer of knowledge and minimizes the likelihood of their wasting time on problems that have already been solved.
- It provides the framework for the professionals to understand their work performance and to see how to improve it. This results in a highly professional environment and substantial productivity benefits, and it avoids the enormous amount of effort that is generally expended in fixing and patching other people's mistakes.

The Optimizing Process provides a disciplined environment for professional work. Process discipline must be handled with care, however, for it can easily become regimentation. The difference between a disciplined environment and a regimented one is that discipline controls the environment and methods to specific standards while regimentation defines the actual conduct of the work.

Discipline is required in large software projects to ensure, for example, that the people involved use the same conventions, don't damage each other's products, and properly synchronize their work. Discipline thus enables creativity by freeing the most talented software professionals from the many crises that others have created.

The need. There are many examples of disasters caused by software problems, ranging from expensive missile aborts to enormous financial losses. As the computerization of our society continues, the public risks due to poor-quality code will become untenable. Not only are our systems being used in increasingly sensitive applications, but they are also becoming much larger and more complex.

While proper questions can be raised about the size and complexity of current systems, they are human creations and they will, alas, continue to be produced by humans — with all their failings and creative talents. While many of the currently promising technologies will undoubtedly help, there is an enormous backlog of needed functions that will inevitably translate into vast amounts of code.

More code means increased risk of error and, when coupled with more complexity, these systems will become progressively less testable. The risks will thus increase astronomically as we become more efficient at producing prodigious amounts of new code.

As well as being a management issue, quality is an economic one. It is always possible to do more inspections or to run more tests, but it costs time and money to do so. It is only with the Optimizing Process that the data is available to understand the costs and benefits of such work. The Optimizing Process thus provides the foundation for significant advances in software quality and simultaneous improvements in productivity.

There is little data on how long it takes for software organizations to advance through these maturity levels toward the Optimizing Process. Based on my experience, transition from level 1 to level 2 or from level 2 to level 3 take from one to three years, even with a dedicated management commitment to process improvement. To date, no complete organizations have been observed at levels 4 or 5.

To meet society's needs for increased system functions while simultaneously addressing the problems of quality and productivity, software managers and professionals must establish the goal of moving to the Optimizing Process.

This software-development process-maturity model reasonably represents the actual ways in which software-development organizations improve. It provides a framework for assessing these organizations and identifying the priority areas for immediate improvement. It also helps identify those places where advanced technology can be most valuable in improving the software-development process.

The SEI is using this model as a foundation for a continuing program of assessments and software process development. These assessment methods have been made public,[8,9] and preliminary data is now available from several dozen software organizations.

Figure 2 shows the maturity distribution of these organizations and the three leading problems faced at each level. At level one, the distribution is shown by quartile. There is not yet sufficient data to provide this detail for levels 2 or 3. As further data is gathered, additional reports will be published on the results obtained. ⊹

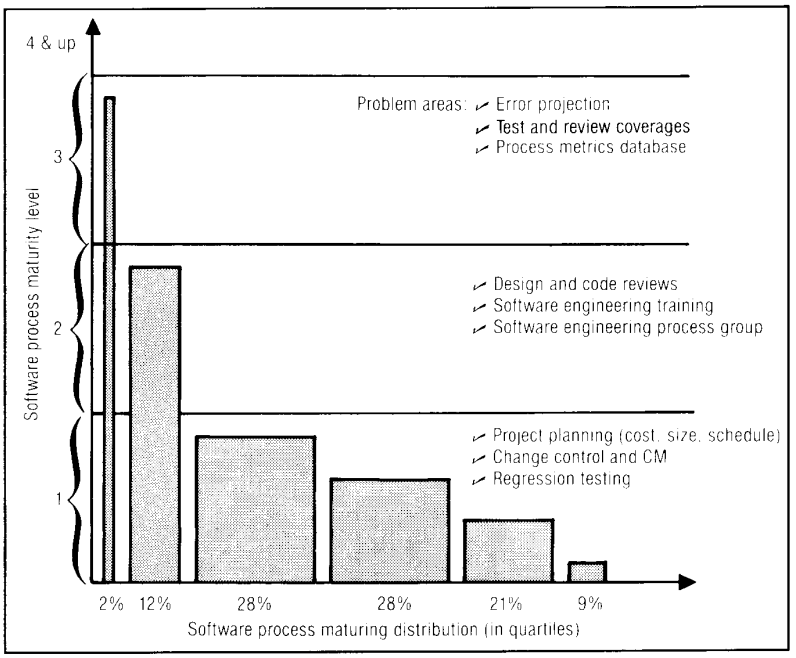

Figure 2. Early results from several dozen software organizations queried by the SEI shows the maturity distribution and the three leading problems faced at each level. At level one, the distribution is shown by quartile. There is not yet sufficient data to provide this detail for levels 2 or 3. To date, no complete organizations have been observed at levels 4 or 5.

Acknowledgments

Much of the early work on software process maturity was suggested by my former colleagues at IBM. I am particularly indebted to Ron Radice and Jack Harding for their insights and support. In addition, William Sweet of the SEI and Martin Owens and Herman Schultz of Mitre Corp. have made valuable contributions to this work. I am also indebted to my colleagues at the SEI, particularly Rodger Blair, Larry Druffel, and Greg Hansen, for their helpful comments and suggestions. This work was supported by the Defense Dept.

References

1. W.E. Deming, "Quality, Productivity, and Competitive Position," tech. report, MIT Center for Advanced Eng. Study, Cambridge, Mass., 1982.
2. J.R. Dunham and E. Kruesi, "The Measurement Task Area," *Computer*, Nov. 1983, pp. 47-54.
3. W.S. Humphrey, *Managing for Innovation: Leading Technical People*, Prentice-Hall, Englewood Cliffs, N.J., 1987.
4. R.A. Radice et al., "A Programming Process Architecture," *IBM Systems J.*, Vol. 24, No. 2, 1985, pp. 79-90.
5. M.L. Shooman, *Software Engineering: Design, Reliability, and Management*, McGraw-Hill, New York, 1983.
6. R.W. Wolverton. "The Cost of Developing Large-Scale Software," *IEEE Trans. Computers*, June 1974, pp 615-636.
7. M.L. Shooman and M.I. Bolsky, "Types, Distribution, and Test and Correction Times for Programming Errors," *Proc. Int'l Conf. Reliable Software*, IEEE, New York, 1975, pp. 347-357.
8. W.S. Humphrey and D.H. Kitson, "Preliminary Report on Conducting SEI-Assisted Assessments of Software-Engineering Capability," Tech. Report SEI-87-TR-16, Software Eng. Inst., Pittsburgh, July 1987.
9. W.S. Humphrey and W.L. Sweet, "A Method for Assessing the Software Engineering Capability of Contractors," Tech. Report SEI-87-TR-23, Software Eng. Inst., Pittsburgh, Sept. 1987.

Watts S. Humphrey is director of the software process program for the Software Engineering Institute. This group provides leadership in establishing advanced software engineering processes, metrics, methods, and quality programs for the US government and its contractors.

He worked at IBM from 1959 to 1986, where he was director of programming quality and process. Humphrey has written two books, *Managing for Innovation: Leading Technical People* and *Switching Circuits with Computer Applications*.

Humphrey received a BS in physics from the University of Chicago, an MS in physics from the Illinois Institute of Technology, and an MBA from the University of Chicago. He has taught graduate electrical engineering at Northeastern University. An IEEE Fellow, he is also a member of the ACM.

Questions about this article can be addressed to Humphrey at the SEI, Carnegie Mellon University, Pittsburgh, PA 15213.