**Technical Report**

**CMU/SEI-87-TR-11**

**ESD-TR-87-112**

**June 1987**

# Characterizing the Software Process
# A Maturity Framework

**Watts S. Humphrey**

The Software Process Feasibility Project
The Software Process/External Operations Program



**Software Engineering Institute**
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/XRS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

**Review and Approval**

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Karl Shingler
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

# Characterizing the Software Process
# A Maturity Framework


## Abstract

Improvement in the performance of software development organizations is an essential national need. The improvement process has five basic elements: 1 - an understanding of the current status of the development process, 2 - a vision of the desired process, 3 - a prioritized list of required improvement actions, 4 - a plan to accomplish these actions, and 5 - the resources and commitment to execute the plan. This paper addresses the first three of these elements by providing a model for software organizational improvement. The structure of this model provides five maturity levels, identifies the key improvements required at each level, and establishes a priority order for implementation. This model has been tested with a number of organizations and found to reasonably represent the status and needs of actual software development groups.


## 1. Introduction

The Software Engineering Institute of the Carnegie-Mellon University was established in December 1984 to address the well recognized need for improved software in U. S. Department of Defense operations. Today, software costs are growing at approximately 12% per year and the demand for more function is growing even faster [3]. Software is a major and increasing portion of U.S. DoD procurement costs and software severely affects the schedules and utility of many weapons systems developments.

This paper describes the initial results of an SEI project to provide the U.S. Department of Defense with a means to characterize the capabilities of software development organizations. The software process maturity framework which has been developed can be used both by the Department of Defense and by software organizations to assess their own capabilities and identify the most important areas for improvement.

## 2. Software Development Processes

An important initial step in addressing software problems is to treat the entire development task as a process which can be controlled, measured, and improved. For this purpose, we define a process as that sequence of tasks which, when properly performed, will produce the desired result. Clearly, a fully effective software process must consider the interrelationships of all the required tasks, the tools and methods used, and the skill, training, and motivation of the people involved.

The basic principle of software process management is that if the development process is under statistical control, a consistently better result can only be produced by improving the process. If the process is not under statistical control, no progress is possible until it is [1]. Statistical control

means that if the work is repeated in roughly the same way, it will produce approximately the same result.

To improve their software capabilities, organizations need to take five basic steps:

1. understand the current status of their development process
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
5. commit the resources to execute the plan

This paper addresses these points by providing a framework for characterizing the status of a software process into one of five maturity levels. There are several reasons for using this maturity structure:

1. The maturity levels were selected to reasonably represent the actual historical phases of evolutionary improvement of real software organizations.
2. Each maturity phase should represent a level of software process improvement which is reasonably achievable from the prior level.
3. Each maturity level should suggest interim improvement goals and progress measures.
4. A set of immediate improvement priorities should be readily apparent once an organization's status in this framework is known.

This process maturity structure is intended for use in conjunction with an assessment methodology and a management system [3,4,6]. Assessment provides a way to identify the organization's specific maturity status and the management system establishes a structure for actually implementing the priority actions needed to improve the organization.

Preliminary tests of this methodology have been conducted with some 35 development projects in ten industrial and government software development organizations. These early results indicate that the model reasonably represents the state of the organizations and provides a mechanism to rapidly identify the key improvement issues they face.

## 3. An Ideal Software Process

While there are many potentially useful improvements that one can visualize, it is worthwhile to examine the characteristics to be expected from a truly effective software process. First, it would be predictable. That is, cost estimates and schedule commitments would be met with reasonable consistency and the quality of the resulting products would generally meet the users needs.

W. Edwards Demming, in his work with the Japanese after World War II applied the concepts of statistical process control to industry [1]. While there are important differences, these concepts are, in important ways, just as applicable to software as they are to automobiles, cameras, wrist watches, and steelmaking. A software development process which is under statistical control will, for example, produce the desired results within the anticipated limits of cost, schedule and quality.

The basic principle behind statistical control is measurement. As Lord Kelvin said about 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].

# 4. Process Maturity Levels

The five levels of process maturity which have been defined are:

1. **Initial** - until the process is under statistical control, no orderly progress in process improvement is possible.
2. **Repeatable** - a stable process with a repeatable level of statistical control is achieved by initiating rigorous project management of commitments, cost, schedule, and change.
3. **Defined** - definition of the process is necessary to assure consistent implementation and to provide a basis for better understanding of the process. At this point, it is probable that advanced technology can be usefully introduced.
4. **Managed** - following the defined process, it is possible to initiate process measurements. This is where the most significant quality improvements begin to appear.
5. **Optimized** - with a measured process, the foundation is in place for continuing improvement and optimization of the process.

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.

## 4.1. The Initial Process

The Initial Process could properly be called ad hoc or chaotic. Here, the organization typically operates without formalized procedures, cost estimates, and project plans. Tools are not well integrated with the process or uniformly applied. Change control is lax and there is little senior management exposure or understanding of the problems and issues. Since problems are often deferred or even forgotten rather than solved, software installation and maintenance often present serious problems.

While organizations at the Initial Process may have formal procedures in place for project control, there is no management mechanism to assure that 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. In essence, if the process is appropriate, it must be used in a crisis and if it is not appropriate, it should not be used at all.

One key reason why organizations behave in this chaotic fashion is that they have not gained sufficient experience to understand the consequences of such behavior. Since 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 can advance to the Repeatable Process by instituting basic project controls. The most important are:

1. **Project Management**. The fundamental role of a project management system is to insure effective control of commitments. This requires adequate preparation, clear responsibility, a public declaration, and a dedication to performance [4]. For software, this starts with an understanding of the magnitude of the job to be done. In any but the simplest projects, a plan must then be developed to determine the best schedule which can be met and the anticipated resources required. In the absence of such an orderly plan, no commitment can be better than an educated guess.

2. **Management Oversight**. A suitably disciplined software development organization must have corporate oversight. This includes review and approval of all major development plans prior to their official commitment. A quarterly review is also conducted of facility-wide process compliance, field 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 frequent over commitments and cost surprises.

3. **Product Assurance**. A product 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 which is between 5% and 10% of the size of the development organization.

4. **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 changes are essential, historical evidence demonstrates that the vast bulk of changes can be deferred and phased in at a subsequent point. If change is not controlled, orderly testing is impossible and no quality plan can be effective.

## 4.2. The Repeatable Process

The Repeatable Process has one important strength over the Initial Process: it provides a reasonable measure of commitment control. This is such an enormous advance over performance at 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 doing similar work. Organizations at the Repeatable Process thus face major risks when they are presented with new challenges. Examples of the changes that represent the highest risk at this level are the following:

1. Every new technology provides a mix of risks and benefits. Unless the risks are understood and addressed in an orderly way, they will likely cause serious and unanticipated problems. In the Repeatable Process, new tools and methods will likely impact the way 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.

2. When the Repeatable Process 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 serious 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 larger scale projects. These changes again destroy the relevance of the intuitive historical basis on which the organization relies.

3. Major organization changes can also 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 to the Defined Process are the following:

1. Establish a process group. This is a technical group with exclusive focus on improving the software development process. In most software organizations the 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 specific 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 2% the size of the development organization. Because of the need for a nucleus of skills, groups smaller than about four professional are unlikely to be fully effective.

2. Establish a software development process architecture which describes the technical and management activities required for proper execution of the development process [5]. The architecture is a structural decomposition into tasks, which each have 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 together with the adoption of modern implementation languages.

## 4.3. The 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. That is, 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 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, one can focus the measurements on specific tasks and items. The process architecture is thus an essential prerequisite to effective measurement.

In advancing from the Defined to the Managed Process, the key steps are the following:

1. Establish a minimum basic set of process measurements to identify the quality and cost parameters of each process step.
2. Establish a process data base with the resources to manage and maintain it.
3. Provide sufficient process resources to analyze this data and advise project members on the data's meaning and use.
4. Assess the relative quality of each product and inform management where quality targets are not being met. This is typically done by the assurance organization.

## 4.4. The Managed Process

In advancing all the way from the Initial to the Managed Process, software organizations will typically 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.

Data gathering should be approached with care and each piece of data carefully defined in advance. Productivity data is generally meaningless unless explicitly defined. For example, the simple measure of lines of source code per expended development month can vary by over two orders of magnitude 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 factor of ten variation. Similarly, counted lines may be used directly or converted to equivalent assembler code, varying again by factors of up to seven [8]. Management, test, documentation, and support personnel may or may not be counted when calculating labor months expended. Again, the variation can run at least as high as seven [9].

When the different groups gathering data do not use identical definitions, the results are not comparable, even if comparing them made sense. The tendency with such data is to use it to compare several groups and put pressure on those with the lowest ranking. This is an unfortunate 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 fifth amendment is based on sound principles since few people can be counted on to provide reliable data on their own performance.

The two fundamental requirements for advancing from the Managed to the Optimized Process are:

1. Provide automatic support for gathering process data. Some data can not be gathered by hand and all manually gathered data is subject to error and omission.
2. Turn the management focus from the product to the process.

### 4.5. The Optimized Process

The step from the Managed to the Optimized Process involves a paradigm shift. Up to this point, software development managers are largely focused on their product efforts and will typically only gather and analyze data that directly relates to product improvement. With the Optimized 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.

As an example of what can be done to optimize the process, errors can be identified and fixed far more economically by using code inspections than through testing. While there is little published data, a useful rule of thumb is that it takes about one to four working hours to find and fix a bug through inspections while typical function or system test requires about 15 to 20 labor hours per bug fixed [7]. It is thus clear that testing is not a good way to find and fix bugs. On the other hand, it would be unwise to eliminate testing completely since it provides a useful check against human frailties.

With the data that is available with the Optimized Process, new perspectives are apparent regarding testing. For most projects, a little analysis will make it clear that there are two distinctly different activities involved. The first is the removal of bugs. To reduce this cost, inspections should be emphasized together with any other techniques which are found to be cost effective. The role of functional and system testing should then be changed to one of finding symptoms that are further explored with more economical methods.

With the Optimized 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.

At this point, one no longer needs reams of paper to describe what is happening since simple yield curves and statistical plots provide clear and concise indicators. It is now possible to assure the process and, by using it, have confidence in the quality of the resulting products.

## 5. People in the Optimized Process

Clearly, any software development process is dependent on the quality of the people that implement it. Even with the best people, however, there is always a limit to what they can accomplish. When they 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 Optimized Process helps in several ways:

1. It provides management with the means to understand where help is needed and how to best provide the people with the support they require.

2. With an Optimized Process, the professionals can communicate in concise quantitative terms. This facilitates the transfer of knowledge and minimizes the likelihood of reinventing the wheel.

3. An enormous amount of effort is generally expended in fixing and patching other peoples' mistakes. The Optimized Process provides the framework for the professionals to understand their work performance and to understand how to improve it. This results in a highly professional environment and substantial productivity benefits.

The Optimized Process provides a disciplined environment for professional work. In spite of the undoubted advantages, many programmers are naturally nervous about the impact it will have on them. There is considerable comfort in informal processes and a natural reluctance to change to a more structured environment. While this is understandable, a formal process should not be seen as threatening. There is an enormous difference between a disciplined environment and a regimented one. Discipline is normal and natural in high technology. Semiconductors can not be designed without precise instrumentation and environmental control. This discipline enables creativity, for it is difficult for semiconductor designers to be creative when using private tool kits or working in a sloppy environment. The same is true for bioengineering, high energy physics, medical research, astronomy, and meteorology, to name a few. Process discipline provides the freedom for the most talented software professionals to be creative by freeing them from the many crises that others have created.

# 6. The Need for the Optimized Process

There are many examples of disasters which have been caused by software bugs. They range from expensive missile aborts to enormous financial losses. As the computerization of our society continues, the public risks of poor quality code will become untenable unless orderly steps are taken to improve our software processes.

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. While many of the currently promising technologies will undoubtedly help, there is an enormous backlog of needed function which 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.

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

To meet the needs of society for increased system functions while simultaneously addressing these risks, we must move rapidly to the Optimized Process. There is no other way.

# 7. Conclusion

This software development process maturity model has been found to reasonably represent the actual ways in which software development organizations improve. It provides a framework for assessing such organizations and identifying the priority areas for immediate improvement. It also assists in identifying where advanced technology can be of most value in improving the software development process.

The Software Engineering Institute is using this model as a foundation for a continuing program of assessments and software process development efforts. This work is being updated and will be made available for public comment and use.

# Acknowledgment

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

## References

[1] W. Edwards Demming, Quality, Productivity, and Competitive Position, Cambridge, MA: Massachusetts Institute of Technology Center for Advanced Engineering Study, 1982.

[2] J. R. Dunham and E. Kruesi, "The Measurement Task Area," *IEEE Computer*, vol. 16, no. 11, November 1983.

[3] W. S. Humphrey, "The IBM Large-Systems Software Development Process: Objectives and Direction," *IBM Systems Journal*, vol. 24, no. 2, 1985.

[4] W. S. Humphrey, *Managing for Innovation - Leading Technical People*, Englewood Cliffs, NJ: Prentice-Hall, 1987.

[5] R. A. Radice, N. K. Roth, A. C. O'Hara, Jr., and W. A. Ciarfella, "A Programming Process Architecture," *IBM Systems Journal*, vol. 24, no. 2, 1985.

[6] R. A. Radice, J. T. Harding, P. E. Munnis, and R. W. Phillips, "A Programming Process Study," *IBM Systems Journal*, vol. 24, no. 2, 1985.

[7] M. L. Shooman and M. I. Bolsky, "Types, Distribution and Test and Correction Times for Programming Errors," *Proc. of the 1975 International Conference of Reliable Software*, New York, IEEE, 1975, pp347-357.

[8] M. L. Shooman, *Software Engineering: Design, Reliability, and Management*, McGraw-Hill, New York, 1983.

[9] R. W. Wolverton. "The Cost of Developing Large-Scale Software," *IEEE Transactions on Computers*, June, 1974.

# Table of Contents