The Effectiveness Formula[1] Key to Productivity Improvement

Dr. Randall W. Jensen, Consultant

Abstract. The Effectiveness Formula is based on three development attributes of success: communication, management effectiveness, and technology. The formula in its generic form can be applied to any organization in any industry.

"Our job is to escape the cave, look around, then come back and tell others what we have seen…Of course, they won't believe us." Daniel K. McKiernan

Introduction

The logical place to begin a discussion about software engineering is the "software crisis" at the 1968 NATO conference in Munich, Germany. A list of software problems was presented as the major development concerns at the NATO Conference. The problem list included software that was:

- Unreliable
- Delivered late
- Prohibitive in terms of modification costs
- Impossible to maintain
- Performed at an inadequate level
- Exceeded budget costs

This list of problems still persists in much of the software development industry some 40 years later.

The term "software engineering" was first used in 1968 as a title for the world's first conference on software engineering, sponsored and facilitated by NATO[2]. The conference was attended by international experts on software who agreed that the discipline of software engineering was needed to address poor quality of software, get projects exceeding time and budget under control, and ensure that software is built systematically, rigorously, measurably, on time, on budget, and within specification. One of the significant conference outputs was a software engineering college curriculum that just happened to be identical to the standard computer science curriculum.

Software engineering is defined as (1) the systematic design and development of software products and (2) the management of the software process. Software engineering has as its primary objectives the production of programs that meet specifications, and are demonstrably accurate, produced on time, and within budget.[3]

All organizations can be represented by the people-process-project triad shown in Figure 1. The projects, which vary from industry to industry, are represented by the Project node. All organizations contain a People node representing the physical environments the people are part of, and the management environments of those individuals. The Process node of the triad represents the development processes used to produce the organization products and the management of those processes. Organization capability and development productivity are largely driven by the communication and management attributes of the People node of the triad model.

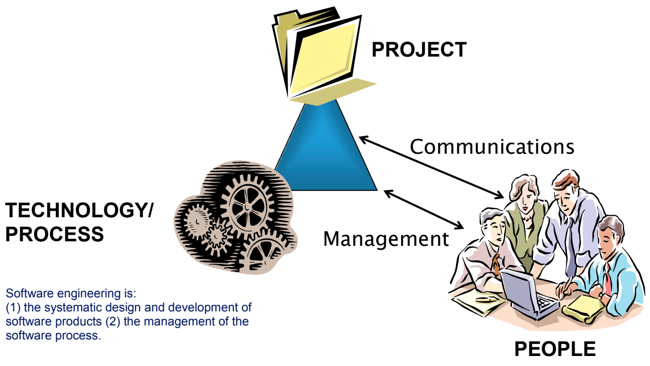
PROJECT
Communications
TECHNOLOGY/ PROCESS
Management
Software engineering is: (1) the systematic design and development of software products (2) the management of the software process.
PEOPLE

Figure 1 Software development triad: Projects, Process, People

For the purpose of this presentation I am going to divide the processes into two camps. The first camp includes "traditional" processes represented by the classic waterfall and spiral[4] development and other variations of the waterfall process. The second camp is comprised of the varied agile software development processes. Agile methodologies include Pair Programming[5] (1975), Scrum[6] (1995), Crystal Clear[7] (1996), Extreme Programming[8] (1996), among others. A major agile characteristic is stated in the Agile Manifesto[9] as "We value individuals and interactions over processes and tools." The traditional camp can be described as those who value processes and tools over individuals and interactions.

The people node is common to both traditional and iterative Agile process camps. For example, pair programming, is very dependent on communications and management support to function, and the pair programming concept works well in a traditional environment. Barry Boehm wrote in 1981 that: "Poor management can increase software costs more rapidly than any other factor…"[10]

Weinberg's Second Law of Consulting[11] added a supporting observation: "No matter how it looks at first, it's always a people problem."

Looking deeper into the list of software problems, we find that the perceived solution to the software development problem is software engineering, or technology. According to the results from the 2013 Standish Chaos Manifesto1[12], technology has not been the total solution to project success. The Chaos report divides projects into three classes: successful, challenged, and failed. About 39 percent of the 2012 projects evaluated were successful. Forty three percent were delivered, but with significant average overruns of near 59 percent of cost and 74 percent of schedule while delivering only 69 percent of the original requirements (challenged). Still about 18 percent were cancelled before delivery (failed).

The Effectiveness Formula

Thirty-five years ago, Chuck Tonies and I wrote about the state of software engineering as a basis for what we called the "Effectiveness Formula." As I reviewed the introduction to our 1979 book Software Engineering for this presentation, I was struck by the similarity between our description of software engineering in 1975 and the state of the profession today. The lessons we learned in our software development work are directly applicable to both engineering and development in any industry.

Most development activities, including software system development, are dynamic. No matter how effective the development methods and the process, and no matter how stable the project staff, some degree of rethinking, replanning, redefining, and redirection are necessary as a project proceeds. Unfortunately, communication among team members is not always perfect. An incomplete or incorrect understanding of requirements, designs, and interfaces is inevitable. Frequent communication among all participants on a development project is the only way to prevent or correct such misunderstandings.

If people are not capable of participating (or are not motivated to participate) in the inevitable ebb and flow of management decisions on a project, or if they can't communicate daily with members of the team, the value of their contributions (no matter how brilliant) is diminished, because those contributions probably don't match the real product requirements.

We developed a conceptual model of the software development organization and its embedded interactions to illustrate the software development process. The simple model highlights the interactions that are always present in any development organization. A person's value to an organization operating in the industrial environment is dependent on three attributes: (1) available technology, (2) the ability to understand management concepts, and (3) the ability to communicate. The model shows these qualities are so intimately involved in the development process that the net effect of an individual's effort is best represented by the product of the Effectiveness Formula attribute values:

$$E=C[M(T)] \qquad (1)$$

where

E = Net effectiveness (0–1)

C = Communication ability and skills (0–1)

M = Management concept awareness (0–1)

T = Technical ability (0–1)

Our experience in the software industry, and especially in product-oriented environments, has shown the formula to be a realistic model of software engineering performance. This effectiveness (performance) model can be applied equally well to any industrial development endeavor. If any of the attributes have a value of zero, the net effectiveness of the organization is also zero. If all of the attributes are 0.5, the effectiveness of the organization is only 0.125. Software engineering, by definition, does not include either management or communication.

We are still in an age of technical specialization; however, software development work is by its very nature a complex interactive process that involves much more than technology. It requires careful, intense management, and even the most specialized of the contributors must act in concert with his colleagues and the management plan if the development process is to be efficient.

The software development industry, especially within the U.S. Department of Defense (DoD), also collects enormous amounts of data to improve development estimates, and that stockpile provided much of the data behind the refinement of the Effectiveness Formula today. The organization-related parameters of both the Seer and COCOMO software cost estimating models can be mapped directly into this conceptual model.

I have been collecting software cost-analysis data from across the Department of Defense community since 1972, to calibrate cost- and schedule-estimating models. Within these records, dating back to the 1960s, I noted a constancy of the capability data for each of the DoD contractors, changing very little over the 50-year period continuing into the 2000s.

As an example of the use of the Effectiveness Formula as a

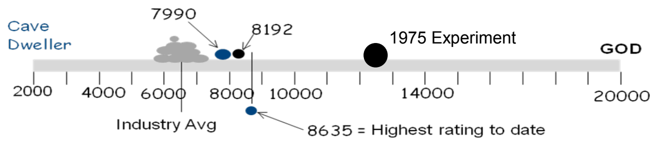

Figure 2 Capability Measures for traditional and agile environments

capability measure, the data in Figure 2 represents the distribution of Seer-based developer basic technology constant[13] values across the range of 2000 to 20000. The clustering of the project data around the 6500 value shows the measured consistency of the organization capabilities across the industry. The data cluster at 6500 is also related to traditional developments with low communication and management model values. The data points between 7990 and 8635 represented projects with higher management ratings. The 1975 data point represents the first measured pair programming project.

The first rather obvious observation in the data is that most software organizations are using similar or identical development technologies. The organizations adopted similar methods and tools at roughly the same time. Each technology improvement offered some capability improvement as shown in Figure 2, even though the capability gains are not strikingly significant.

The second observation is that the management approaches across the clustered organizations are almost identical, all being slight variations of classical management best described as Theory X[14]. The five management functions (planning, organizing, commanding, coordinating and controlling) introduced in 1916 by Fayol[15] are still the focus of management training today. The development environment for these organizations all provided limited communications.

Table 1 shows the relationship between the Effectiveness Formula values from the completed project data used to generate Figure 3 to the relative organization capability ratings (percentile) of the software development organization database.

Basic Technology Constant (C_{tb})	Effectiveness Value	Relative Percentage
2000	0.11	10
6500	0.25	50
8192	0.38	70
8635	0.40	78
12500	0.5	88

Table 1 Comparison of Basic Technology Constant (Ctb), Effectiveness value and relative ranking of project data

The basic technology constant data in the table can be related directly to the Effectiveness Formula values.

The traditional software development organization capability ratings are remarkably clustered around an Effectiveness Formula value of approximately 0.25, or about 6500 on the familiar basic technology constant Ctb value. The relative percentile ranking of such an organization is the industry average (50%). Assuming a traditional organization utilizes a fairly advanced technology attribute value of 0.8, the combined impact of communication and management of that organization must be a low combined value of approximately 0.3 to achieve and effectiveness value of 0.25.

The Attributes

Success in software development productivity improvement is not a matter of concentrating one's effort in any one of the three attributes that comprise the Effectiveness Formula. A narrow view of the importance of any one of the attributes spells failure for an organization attempting to make improvements.

Technology

The first part of the software engineering definition consists of the set of methods and tools we use to develop the ultimate product. Technology has existed since the time of the agrarian plow, and it is constantly evolving to support the needs of the developer. Better tools and methods contribute to improving effectiveness and efficiency as one of the three primary attributes of the Effectiveness Formula.

There have been several development technology breakthroughs during the past forty years that have significantly decreased the cost of software products. For example, the introduction of FORTRAN and COBOL decreased the cost of a given product functionality to one-third of the cost achievable when implemented in Assembler due to the decrease in the source lines of code to required achieve the product functionality. The transitions from C++ to the newer visual languages, and the advent of object-oriented structures, created significant software cost savings because the required number of source lines have continually decreased. However, when we look at the effort required to produce a single line of source code in any given programming language, we see that traditional software development productivity (measured from start of development through delivery or software-system integration) has increased, with little blips and dips, almost linearly at the rate of less than two source lines per person-month (lppm) per year as shown in Figure 3.

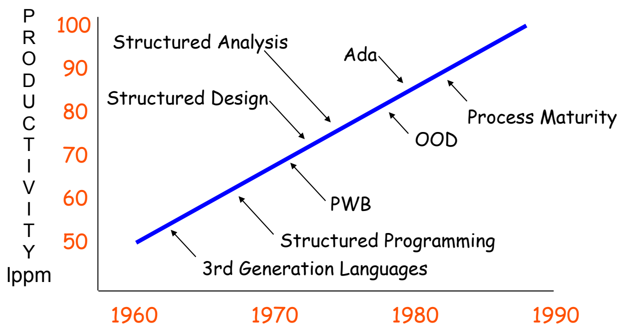

Figure 3 Traditional software development productivity gains -- 1960 to 1990

We have been learned new things about software development during this period. The development environment focus was almost entirely on the product during the 1960s and early 1970s. The principle activity, once the requirements were established was programming. Software development technology, namely programming languages, improved as the system requirements grew to manage the size and complexity of the tasks increased. Development platforms improved to support the ever increasing size of software systems.

The second part of the software engineering definition formalizes the development process. Technology was the primary focus in the 1960s, when the software development discipline was new. The early 1970s brought a shift in focus to the development process. In the mid-1980s, the Software Engineering Institute introduced the CMM® as an approach to stabilize the development process, improving quality and productivity by focusing energy on process improvement. The CMM concept was introduced in 1987 by Watts Humphrey[16]. Humphrey stated: "While there are many unique characteristics to software, they all require more management discipline, not less. Managers should thus demand detailed plans, tracking systems, and periodic technical and management reviews. Software management should be entirely traditional, only more so. Unfortunately, many managers who insist on these items for hardware let their software teams get by without them."

Humphrey's remark clearly conflicts with the agile focus on individuals and interactions over processes and tools. The CMMI® Guidelines for Process Integration and Product Improvement place a rigid outline for managing the development of software products without filling in the details of how the process is implemented. Process provides no support for the second and third attributes of the Effectiveness Formula; that is communications and management.

Management of the process, or Process maturity, is not a programming language, not a development method or approach, nor is it a tool set that supports the development, but it is a major factor in any engineering approach. CMMI[17] (Capability Maturity Model Integration) is a process improvement model for the development of products and services consisting of best practices that address development and maintenance activities covering the product lifecycle from conception through delivery and maintenance.

Communication

Broadly defined, communication is the act or process of exchanging information between individuals, using a common system of symbols, signs, or behavior. The related definition of collaboration is working jointly with others, especially in an intellectual endeavor. Both elements, communication and collaboration, are necessary to produce a software product effectively and efficiently.

The most important observation is the norm for physical environments in current traditional development. The most common modern large-scale software development environment is the "cube farm," in which all communication between developers is carried through the organization's computer network. The cubicle development environment is not designed, by intent or by chance, to foster interactive communication among the project participants. Instead, the cube farm's purpose appears to even the casual observer as a means to prevent communication. The lack of effective communication blocks any attempt to motivate the staff, prevents collaboration within the staff, and eliminates the potential for forming teams.

The least common environment, the "skunk works," is typically defined as a small group of experts who move outside an organization's mainstream operations in order to develop a new technology or application as quickly and efficiently as possible, without the burden of the organization's bureaucracy or strict process application. The skunk works workspace is a physically open environment that encourages intra-team access and communications. Tools and processes are tailored and adapted to the project's requirements. An Agile environment fits into this type of structure.

The effectiveness of voice or visual communications is supported by a well-known research study by Mehrabian and Ferris[18] shown in Figure 4. The information transfer effectiveness, however, is diminished when we remove any source of information. For example, we can remove the visual part of the information transfer by forcing the communicators to use a telephone. This eliminates all of the gestures, body language, and eye contact from the conversation and lowers the information transfer by over 50 percent. This important information source is no longer available to reinforce understanding between the two individuals and leads to gaps in communication, as well as misunderstandings.

Removing the visual and vocal information elements of the discussions leaves us with only 7 percent of the information content. Information transfer is significantly degraded when we rely solely on paper because we remove the ability to ask and respond to clarifying questions. We lose not only the subtle elements of voice communication, but also the real-time elements necessary for feedback between one another.

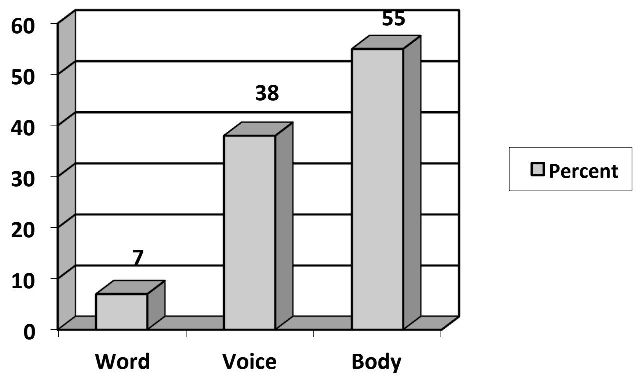

Figure 4 Components of communication.

By placing each software engineer contributing to a product development in a 36 to 64 square foot cubicle lowers the communication attribute of the net Effectiveness Formula to less than 0.07. Some argue that by connecting the software engineer's boxes by a high-speed network connection restores the information content to a level approaching 100 percent in spite of removing the visual and vocal communication elements. The argument is unsupportable. To further degrade the effective communication, consider removing the real-time feedback element as well. Communication between workstations is little more effective than communication by written documents.

The solution to communication barriers is not modern technology, such as the use of e-mail and high-speed network communications. These solutions are often proposed for local communication support and to justify remote software development teams. Ironically, this technology solution raises even greater barriers than the cubicle example. At least people in adjacent cubicles have some physical contact. Remote locations are sometimes separated by thousands of miles.

Communication, which has a range of values between zero and (ideally) unity, has an effective value of only 0.07 in a cube farm. With a value that low, even a management effectiveness value of 1.0 and a perfect technology value (1.0) cannot provide much of a contribution to the person's effectiveness.

Management

Project management theories have been around since long before the beginning of the 20th century. In the early 1900s, the focus of management studies centered on the most effective methods to organize and structure the industrial organization; that is, ways to organize, delegate, and coordinate work efficiently. Five basic functions of management were identified: planning, organizing, commanding, coordinating, and controlling. These five functions, which were defined in 1911, appear in almost all the current management literature

The well-known Hawthorne experiment conducted in the 1920s showed that the solution to the productivity dilemma was not found in the physical working conditions, but in the human aspects. The most significant factor affecting productivity in an organization was found to be interpersonal relationships developed on the job, not just pay and working conditions. This factor is widely referred to as the Hawthorne effect. When informal groups identified with management, productivity rose. The increased productivity reflected the workers' feeling of competence—a sense of mastery over the job and the environment. The study also showed that when the workers' goals were in opposition to those of management as often happens with micro-management, productivity remained at only marginally acceptable levels, or decreased from the norm.

The Hawthorne experiment was a forerunner of the development of Douglas McGregor's classic Theory X-Theory Y view of management. McGregor proposed that there are two primary categories of organizational management thinking[19], each with pronounced impacts on the way organizations function. Theory X assumes that most people prefer to be directed, are not interested in assuming responsibility, and want safety (job security) above all. Theory X corresponds to the belief that most people are motivated by money, fringe benefits, and the threat of punishment.

Managers who follow Theory X assumptions attempt to structure, control, and closely supervise their workers. These managers believe that external control is clearly appropriate for dealing with unreliable, irresponsible, and immature people. The mode is consistent with the five advertised functions of management.

McGregor's alternate theory of basic human behavior, Theory Y, assumed that people are not lazy and unreliable by nature. They can be self-directed and creative if properly motivated. McGregor concluded that it is management's responsibility to free the potential of the workers so that they can achieve their own goals. Supportive Theory Y managers provide the means to achieve organizational goals, as opposed to Theory X managers, who control and closely supervise workers. I find the X-Y comparison to sheepherders and shepherds.

In spite of the work by these behavioral pioneers and many others, software management remains primarily a Theory X culture. I am frequently reminded of Weinberg's Second Law of Consulting[20]: No matter how it looks at first, it's always a people problem.

Summary

Software engineering is, by definition, contained in the Process node of the triad depicted in Figure 1. It is also totally contained in the technology attribute of the Effectiveness Formula. Looking back as far as the 1968 conference on software engineering, it has offered little or no support for capability and productivity improvement. If improved productivity is our goal, software or otherwise, effective leadership should be our focus. The Effectiveness Formula is based on three development attributes of success: communication, management effectiveness, and technology. The formula in its generic form can be applied to any organization in any industry. Fortunately, the software industry has collected sufficient data to apply a quantitative measure to organization productivity and capability.

One important feature of management that is not discussed by McGregor and others is that people must have contact with each other in order to achieve associated social, esteem, and self-actualization goals. Isolating people, such as is accomplished with cubicles, negates the positive aspects of Theory Y. Leadership attitude toward employees allows the organization's communication to grow, thus increasing productivity. You can only imagine a manager trying to lead a group of engineers, none of whom know how their work fits into the system concept or direction being followed by the other engineers. Without communication between employees, productivity improvement is difficult or even impossible to achieve.

A final point to be made in the Effectiveness Formula is the quantitative support of the measure. The conceptual model attribute ranges were simply 0 to 1. A quantitative implementation of the model shown in Table 1underlies both the proven COCOMO and Seer (SEER-SEM) software cost estimating models.

Disclaimer:

CMM® and CMMI® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ABOUT THE AUTHOR

Randall W. Jensen, Ph.D., is an independent software acquisition consultant with more than 50 years of practical experience as a computer professional in hardware and software development. For the past 40 years, he has actively engaged in software engineering methods, tools, quality software management methods, software schedule and cost estimation, and management metrics. He retired as chief scientist of the Software Engineering Division of Hughes Aircraft Company's Ground Systems Group (1993) and a subject matter expert from the USAF Software Technology Support Center (2011). He developed the model that underlies the Sage and the Galorath, Inc.'s SEER-SEM software cost and schedule estimating systems. Jensen received the International Society of Parametric Analysts Freiman Award for Outstanding Contributions to Parametric Estimating in 1984. He has published several computer-related texts, including "Software Engineering" (1979), "Improving Software Development Productivity: Effective Leadership and Quantitative Methods in Software Management" (2014), and numerous software and hardware analysis papers. He has a BS, an MS, and a Ph.D. in electrical engineering from Utah State University.

E-mail: costsage@comcast.net

NOTES

1. Jensen, R. W., Improving Software Development Productivity: Effective Leadership and Quantitative Management, (Englewood Cliffs, NJ: Prentice Hall, 2014)
2. Naur, P., and Brian Randell, Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmish, Germany, October 7-11, 1968
3. Mills, H.D., "The management of software engineering, Part I: Principles of software engineering," IBM Systems Journal , vol.19, no.4, pp.414,420, 1980
4. Boehm, B. W., "A Spiral Model for Software Development and Enhancement, IEEE Computer, May 1988, pp. 61-72
5. Jensen, R. W., "A Pair Programming Experience," CrossTalk , March 2003)
6. Schwaber, K.. and M. Beedle, Scrum: Agile Software Development, (Upper Saddle River NJ: Prentice-Hall), 2002.
7. Cockburn, A., Crystal Clear: A Human-Powered Methodology for Small Teams, (Reading, MA: Addison Wesley), 2004
8. Beck, K., Extreme Programming Explained, (Boston MA: Addison-Wesley), 2001
9. Fowler, M. and J. Highsmith, "The Agile Manifesto," Software Development, August, 2001
10. Boehm, B. W., Software Engineering Economics, (Englewood Cliffs, NJ: Prentice-Hall, Inc.) 1981, pg. 486
11. Weinberg, G. W., The Secrets of Consulting, (New York, NY: Dorset House Publishing), 1985, p. 5.
12. Standish Group International, The Chaos Manifesto, (West Yarmouth, MA: Standish Group International), 2013
13. Jensen, R. W., "A Macrolevel Software Development Cost Estimation Methodology," Proceedings of the Fourteenth Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, CA, November 17-19¸1980
14. Hersey, P. and K. Blanchard, Management of Organizational Behavior, (Englewood Cliffs, NJ.: Prentice-Hall, Inc.), 1977, P.55
15. Fayol, H., General and Industrial Management (London: Pitman, 1916
16. Watts S. Humphrey, Managing the Software Process,(Addison-Wesley, Boston, MA), 1989
17. Chrissis, M. B, M. Konrad and S. Shrum, CMMI: Guidelines for Process Integration and Product Improvement, 2nd Ed. ,(Upper Saddle River, NJ: Addison-Wesley), 2007
18. Mehdrabian, A. and S. R. Ferris, "Inference of Attitudes from Nonverbal Communication in Two Channels," Journal of Counseling Psychology, Vol. 31, 1967
19. Hersey, P. and K. Blanchard, Management of Organizational Behavior, (Englewood Cliffs, NJ.: Prentice-Hall, Inc.), 1977, P.55
20. Weinberg, G. M., The Secrets of Consulting: A Guide to Giving & Getting Advice Successfully. New York, NY: Dorset House, 1986, p. 5.