

Adapting Computer Aided Parametric Estimating (CAPE) to Process Maturity Levels

Bruce E. Fad

**Lockheed Martin PRICE Systems
700 East Gate Drive, Suite 200
Mt. Laurel, NJ 08054**

Abstract

A current thrust in software engineering economics involves process evaluation and follow-up recommendations for improvements that speed delivery time, improve quality, and save cost; in other words, re-engineering the process. A basis for many organizational evaluations is the Software Engineering Institute (SEI) Capability Maturity Model (CMM). This model defines five levels of increasing process maturity and the characteristics of each. Achieving higher levels of maturity involves improvements in the specific areas of Software Project Planning, Software Product Tracking and Oversight, and Integrated Software Management. Improving these processes requires core measurements to establish performance baselines, benchmark improvement, and predict future organization performance. Computer Aided Parametric Estimating (CAPE) can play a major contributing role in software performance and capability measurement. Reaching higher levels of maturity also involves a cost investment that must be weighed by organizational management for benefit. Key to the process improvement decision is the expected return from the investment - How much time will be saved? How much product quality improvement will there be? What are the savings in resources?

SEI CMM

Computer software is a relatively new technology. Despite the explosion of digital data processing in all facets of everyday life, the first electronic computer (the ENIAC) was built only 46 years ago. When compared to older mechanical and electrical device technology, it is fair to say that we are still in the toddler years of software engineering maturity. Many promises from the 70s and 80s regarding a revolution in development methods have been unfulfilled.

Recognizing that the technological maturation process was to be more evolutionary, the Software Engineering Institute (SEI) at Carnegie Mellon University in Pittsburgh, PA began to establish a process maturity framework in late 1986. With assistance from the Mitre Corporation, SEI established the framework goal to be assisting organizations in improving their software process.

The fundamental problem identified is characteristic of most new technological advances - the inability to manage the process of creating specific applications with the technology. In the 80's, the environment was chaotic and undisciplined. Tools were few and not integrated. For the most part, no organization infrastructure existed to guard against the pitfalls of over cost, behind schedule delivery of poor quality software. In fact, the term "pre-planned product improvement" became commonplace in scheduling for software product delivery. Still, there were enough examples of software products producing excellent results, even within undisciplined organizations, to offer clues about general technology improvement. Though success can result from dumb luck, more often it results from disciplined methods that are consistently applied and continuously improved. This was the common thread among the successful software developments observed. What was missing was organization and widespread adoption of the effective discipline. Discipline of methodology was, and to a large extent still is, an individual issue. Relying on certain individuals for success provides no foundation for widespread quality

and productivity improvements of an organization or the technology as a whole. Clearly, solving the problems of software would require definition and building of the infrastructure to engineer and manage the new technology effectively.

The concept of maturity implies adoption and use of a disciplined process, while immaturity implies an ad-hoc approach. The domain of software people covered by the SEI use of maturity is *organizational* as opposed to *individual*. So, while there may be examples of individual software maturity, there are few that encompass entire organizations. Without organizational maturity, there can be little to further technological maturity. So, the idea is to spread successful practices from individuals through policies, standards, and structures that produce an evolving organization maturity. This is known as institutionalizing the software process.

The Capability Maturity Model (CMM) is intended to provide organizations with guidance needed to gain control of the software development and maintenance processes in order to evolve a culture of software engineering excellence. By focusing on a few of the more critical issues related to software quality and process improvement, organizations can aggressively move towards greater maturity. The road map for progression along the path of greater maturity is the framework of the CMM. Its roots trace back to the 1930s quality control theory promulgated at AT&T Bell Labs and later developed further by others, most notably W. Edwards Deming.

The 5 Levels of Software Process Maturity

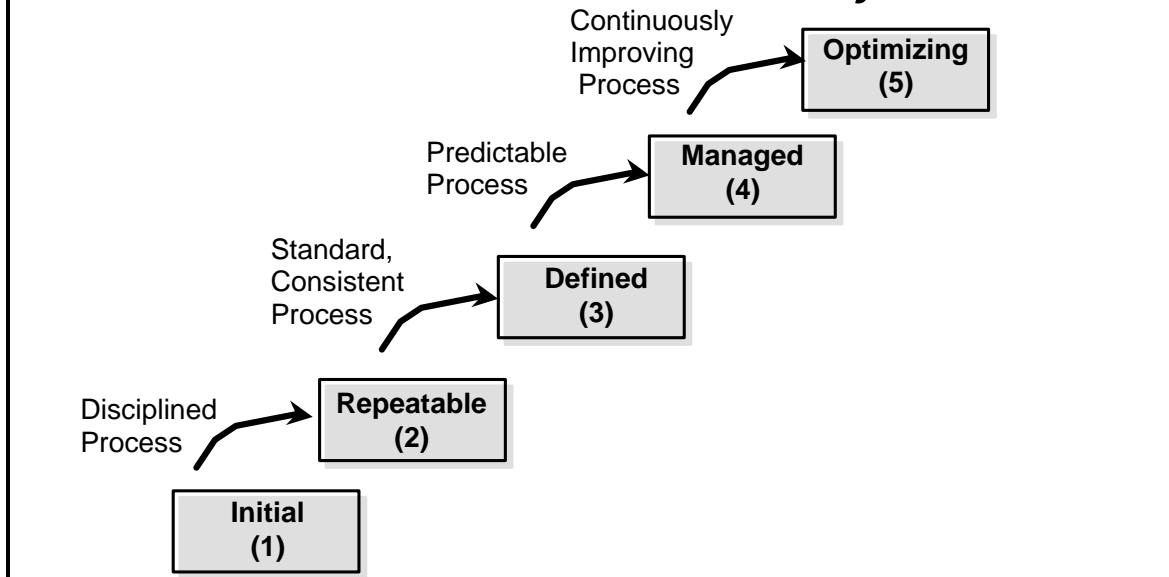
Continuous process improvement is based on many small, evolutionary steps rather than one or two revolutionary innovations. The CMM framework identifies five successive steps and the properties of each in the maturation process. Each level is a well-defined plateau on the path toward maturation. Achieving each level requires that an organization demonstrate the ability to meet the process requirements associated with that level consistently. Notice that the framework does not attempt to quantify the quality or resource requirement goals of a level. This is due to the attention of the model to capability (Capability Maturity Model) and not performance. Software process maturity deals with potential for growth in capability or the expected results from use of a process. Performance at any given time on any given project may be reflective of greater or less maturity than that for which an organization is capable. Over time, the organization's performance will mirror its capability.

The next figure (page 3) illustrates the five SEI maturity levels and the orderly progression from one level to the next. The figure is followed by a brief characterization of each level.

Each maturity level is associated with a more controlled and stable ability to predict organizational performance in developing and/or maintaining a software project. Variations from expectation are confined to a narrow range. The illustration on page 4 depicts the capability norms for Levels 1 and 5. Notice three things from this picture:

1. The expected range of outcomes is much narrower for Level 5.
2. Because of the strength of discipline employed at Level 5, variations from target are truly random. However, at Level 1, variations are more likely to increase in target resources and decrease in target quality due to the greater likelihood of chaos.
3. There is the possibility (however small) that a Level 1 organization can deliver the same or similar quality software with the same expenditure of resources as a Level 5 organization. If this occurs, it will most likely be the result of select individuals at Level 1 performing like most individuals at Level 5.

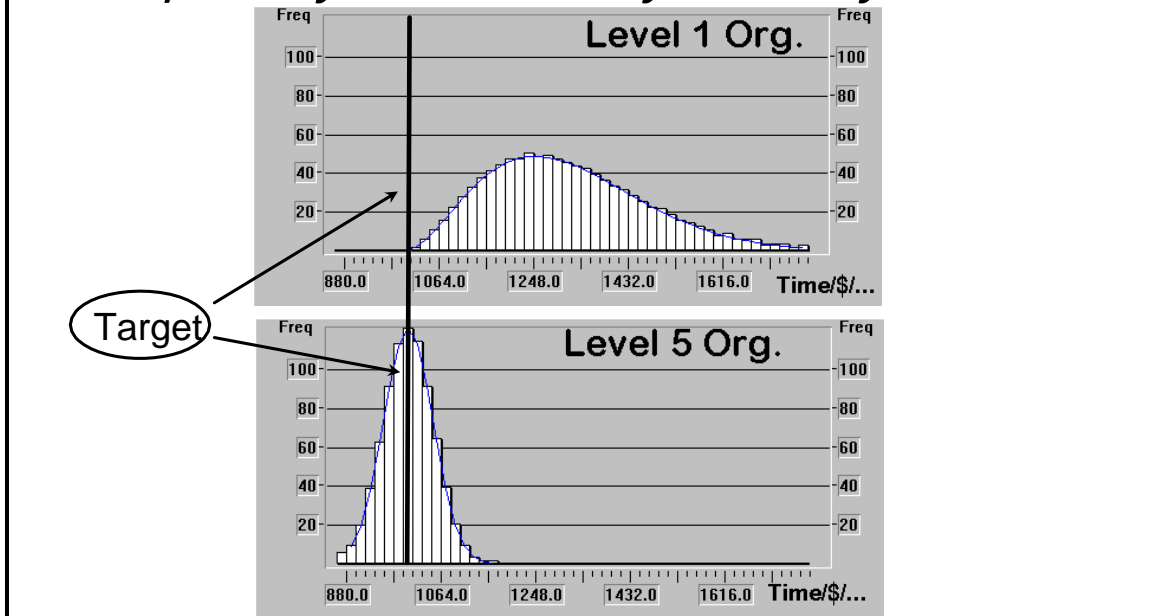
The 5 Software Process Maturity Levels



Characterizations of the 5 Maturity Levels

1). <i>Initial</i>	Ad hoc and occasionally chaotic process with few defined processes. Success depends on individual effort.
2). <i>Repeatable</i>	Uses established basic project management processes to track cost, schedule, and functionality. Contains sufficient process discipline to repeat earlier successes on analogous projects.
3). <i>Defined</i>	Uses a documented, standardized, and integrated (organization wide) software process for both management and engineering activities. All projects will use an approved version of the organization's development and maintenance process.
4). <i>Managed</i>	Detailed measures of the software process and product quality are collected, quantified, understood, and controlled.
5). <i>Optimizing</i>	Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies.

Capability Indicated by Maturity Level



Incentive and Status

The incentives for software organizations to progress through the levels of maturity are self evident. Any business welcomes the opportunity to produce a high quality product quickly and at low cost. This makes for satisfied customers and significant market share. In fact, procuring agencies rely upon levels established through survey in selecting organizations that can meet specific software project reliability requirements. So, there is the added incentive of competing for a larger share of the software market by striving for and achieving high maturity levels. But not all incentives are quantifiable business measures. The harmony associated with smooth operations is an opportunity for all organizations reaching level 5. The damaging effects of stress caused by chaos and confusion are well documented and, unfortunately, well experienced by anyone working in the software field. To a great extent, the productivity improvements expected at each successive maturity level are a result of a working environment that is more harmonious to productivity than the level preceding.

In The Value of Software Metrics for Continuous Process Improvement, 1990 *Software Improvement Conference*, Mr. Herb Krasner of the Lockheed Software Technology Center offered an opinion of motivation for change. Using a 500K Source Lines of Code (SLOC) project as a typical case, Krasner proposes an almost 100 times increase in quality, a twelve fold improvement in productivity, and a three times reduction in development time in moving from Level 1 to Level 5. Contents from a presented table are shown on the next page.

SEI Level	Quality (Defects/KSLOC)	Productivity (SLOC/Hr.)	Total Cost (\$M)	Development Time (Months)
1	9 +	1	33	40
2	3	3	15	32
3	1	5	7	25
4	.3	8	3	19
5	.1 -	12 +	1	16

© 1990 - Herb Krasner

Bold predictions like that above provide the stimuli for doing what is necessary to become a more mature software organization. In order to move forward, we must know where we are starting. A very much debated question is, "Where are we?" Specifically, at which level is our software community as a whole performing today? An answer to this question can only be qualified after defining who "we" are and how "we" are assessed to be capable of performing at a certain SEI level. Still, a few observations from others might help to establish a general state-of-the-art. Data from the June 1989 SEI Assessment Tutorial Workshop studied 167 projects, with the following results:

Workshop Assessments as of 6/89, 167 Projects

SEI Level	Percent Population
1	86
2	13
3	1
4	0
5	0

Data collected since 1989 indicates a trend towards higher SEI levels for the software community as a whole. During its *Applied Software Measurement Course* in Ridgecrest, CA, July 1994, the Software Productivity Consortium instructor cited the following, based on SEI assessments and those done by licensed vendors.

SEI and Licensed Vendor Assessments as of 7/94, Generalized

SEI Level	Percent Population
1	75
2	20 - 23
3	< 5
4	< 1
5	< 1

One company, operating in India was assessed at SEI level 5. But, the data seems to indicate clearly that most organizations continue to employ practices that do not permit repeatable capability. For the immediate future, it would be prudent to concentrate on those processes that will enable an organization to achieve Level 2, so that refinement and practice can lead to the ultimate level.

What kind of investment is required to reach higher maturity levels and what can an organization expect as a return from that investment? The answers will of course depend upon starting position, goal, vehicles selected to reach the goal, and motivation to succeed. The table below summarizes selected improvement experiences of five different organizations.

Summary of Process Improvement Experiences

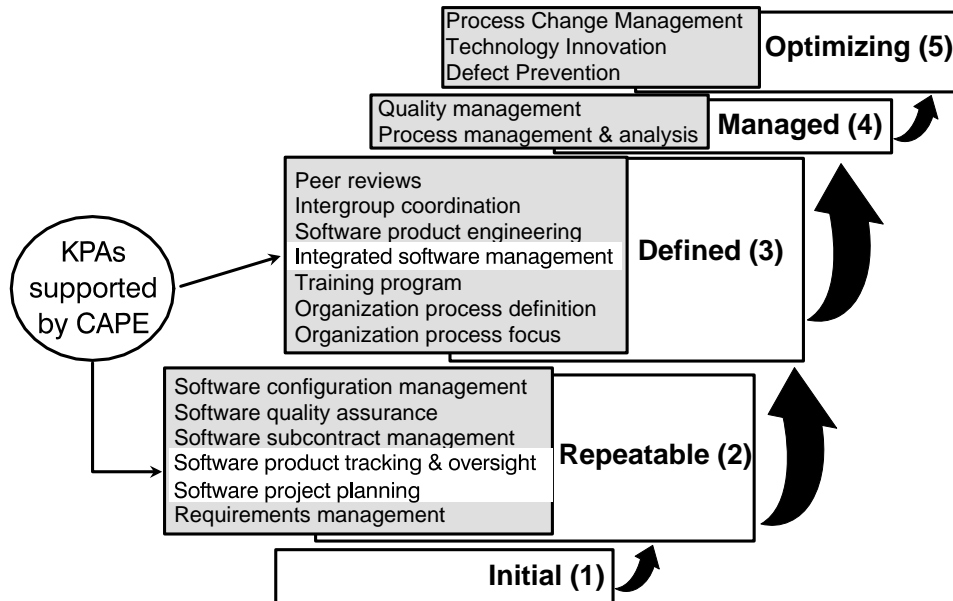
Company	CMM Level	Investment	Benefits/Returns
A Software Systems Lab	1 - '88 2 - '90 3 - '91	\$1M. per year	\$15.8M savings thru '92 2.3x Productivity Improvement 7.3 ROI
B Software Eng. Div., Ground Sys. Group	2 - '87 3 - '91	\$45K. Initial \$400K Maintenance	\$2M per year overrun reduction. 5 ROI
C 20 of 76 Product Units	1 - '90	Not Reported	On-time delivery from 51% to 94% Customer discovered defects from 25% to 10% Validation cycles reduced from 34 to 15
D Gov't. Logistics Cen.	1 - '90	\$462K.	\$2.9M savings 6.35 ROI
E	Not Reported	Not Reported	6x Reduction in customer service requests

Key Process Areas (KPA)

Each maturity level, except for Level 1, is composed of several key process areas (or KPA for short) indicating the areas an organization should concentrate on to improve the software process. They may be viewed as requirements for achieving a specific maturity level. Each level has an associated set of KPA, as illustrated by the figure that follows on page 7.

The specific practices to be executed in each KPA will grow in content as an organization achieves higher levels of process maturity. For example, the scope of estimating done under the Software project planning KPA may evolve from an overall program approach to a more detailed component and phase treatment as the organization moves from Level 2 to Level 3 and higher. This is consistent with the increase in detailed software process understanding as higher maturity is achieved. Each KPA defines a cluster of related activities that collectively strive towards goals that enhance process capability. Though the specific path followed to attain the goals may vary across projects, all goals of the KPA must be satisfied in order to say that the KPA has been met. When the goals of a KPA are accomplished on a continuing basis, the organization has institutionalized the process capability of the KPA. The *Capability Maturity Model for Software* reference to this document defines the goals of each KPA. As an example, a goal of Software project planning for Level 2 is that all affected individuals and groups understand the software estimates and plans and commit to support them. At Level 3, the Integrated Software Management KPA has a goal of technical and management data from past and current projects being available and used to effectively and efficiently estimate, plan, track, and re-plan software projects. These goals will be revisited later in the document when we discuss how CAPE can support achievement of higher maturity levels.

Key Process Areas by Maturity Level



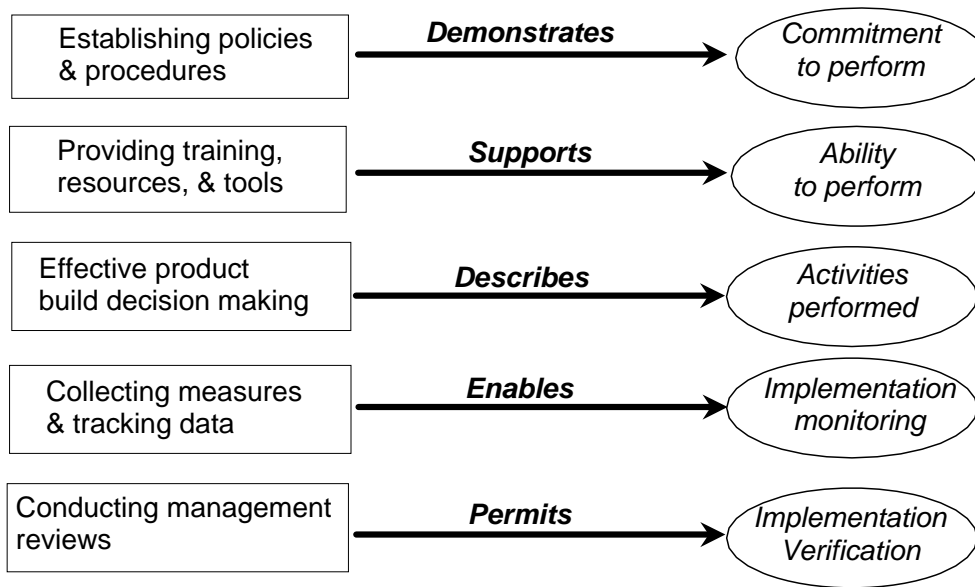
The goals of each KPA will represent the key practices employed by an organization within that process area. The actual practices employed will be those that, for whatever reason, best suit the organization while still meeting KPA goals. Even though there is considerable freedom to adopt best practices locally, there will necessarily be common qualitative features of each successful KPA. These are the established procedures, resource commitments, technical and management execution, and self-governance of excellence and quality (see following page).

The 4 Core Measures

Of all the features discussed above, the one involving the collection of measures and tracking of data more than any other will provide the most objective status of maturity level. It will also be a primary information source for decision making, review, and policy direction. SEI has recommended that an initial set of four core measures be adopted within the US Department of Defense (DoD) to help drive that software community to a Level 2 maturity. Recall that the Software Productivity Consortium reports that at least 75% of those organizations have been professionally assessed as being Level 1 capability.

The core units of measurement recommended by SEI to achieve Level 2 capability are shown on page 8 along with software characteristics addressed by each. These specific measures surfaced from SEI work that initially identified size, effort, schedule, and quality as measures crucial to managing project commitments. They were also identified as foundations to achieving higher maturity levels. A process that collects and uses these four measures under clearly specified controls will satisfy the goals of SEI Level 2. By encompassing the management functions of project planning, project management, and process improvement, these measures form a basis from which to build a comprehensive process improvement program. The referenced document, *The SEI Core Measures: Background Information and Recommendations for Use and Implementation* contains rationale for these specific measures, explanation of their genesis, and definitions and conventions for interpretation and collection of measures.

KPA Common Features



The 4 Core Measures

Measure	Unit of Measure	Characteristics Addressed
Size	Counts of Physical Source Lines of Code (SLOC)	Progress, Reuse, Redesign, Processing function
Effort	Counts of Person-hours expended	Cost, Resource Allocations
Schedule	Milestones - Calendar Dates	Performance
Quality	Counts of Software Errors and Defects	Readiness for Delivery, Improvement Trends

Two cautions about the four measures: 1). They are the preferred measures, but not the only measures of each software characteristic set; and 2). Each has a context that accompanies it and that can affect the value of the measure. Consider the size issue as an example. Logical Source Statements (Instructions), Function Points, Feature Points, and Computer Software Units (CSUs) are also candidates for size measurement. However, these measures are more subject to secondary effects than is SLOC. Still, in an environment where the secondary effects (e.g. software application, design tools, implementation languages) are fairly stable, any of these other size measures can be as useful or even more useful than SLOC. By way of further example, think about the impact of changing requirements on software project milestones. This can stretch the time for completion of design and for conducting test reviews. It can be argued that a mature process is inconsistent with requirements volatility, but remember, we are dealing with movement *to* Level 2 from a level of immaturity. Unfortunately, until we reach Level 2 or higher, changing requirements (of varying degrees by project) will continue to be common. This is an example of context impact on a measure and it must be accounted for qualitatively if it cannot be quantified.

The four core measures are intended to be a means to the goals. Just as a journey can be successfully concluded by use of any number of vehicles, so can the goals of Level 2 KPA be met with measures other than the four listed above, if they are:

- Well understood within the organization;
- Standardized in content for measurement reporting;
- Consistently collected and analyzed;
- Inclusive of all organization groups and projects.

CAPE Utility

Tools are needed to assist in performing the key practices required to achieve higher maturity levels. In the special emphasis areas of the above 4 core measures, Computer Aided Parametric Estimating (CAPE) can be a powerful tool box. But, the particular tools selected from the CAPE box must be ones that are flexible enough to evolve with organization process improvements. The software maturation described by the CMM does not call for change to the production engine - that is, regardless of the maturity level, code is designed, implemented, and tested. What does need to change is the apparatus used to build the engine. In crude terms, it must progress from a three ring circus to a focused assembly line. It is natural then, that any tools employed to measure and predict performance must be able to evolve in the same manner. In CAPE terminology, this is referred to as *adaptation* or *calibration*, and only those tools that can be calibrated to the depth and breadth of measurement requirements will be of value.

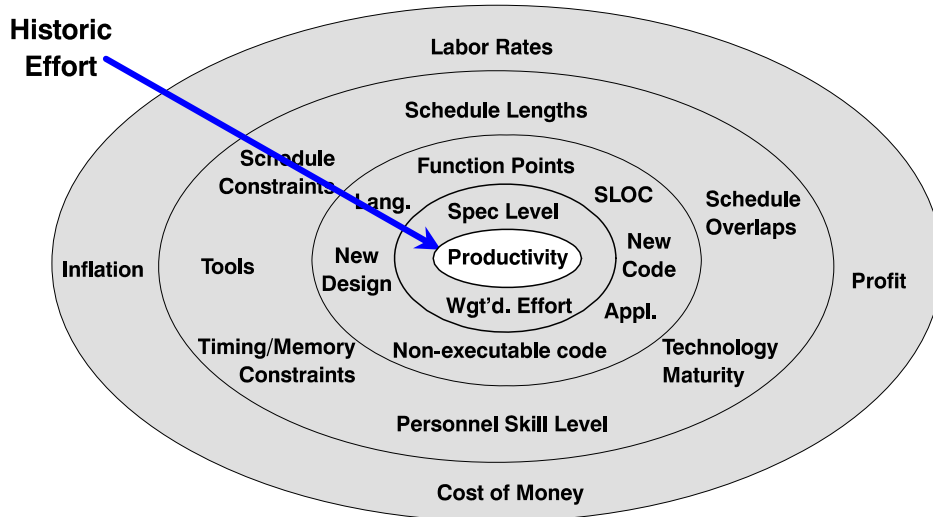
Every software development project encounters complicating factors that impact productivity. Compressed schedules, changing requirements, new languages and tools, are just a few of the real-life factors that cause productivity to deviate from the norm. Performance measurements must be normalized with respect to complicating factors to make comparisons and to accurately predict future organizational performance. CAPE tools can produce normalized productivity measures. The figure on page 10 illustrates the PRICE S method of removing the complicating layers (“peeling the onion”) affecting a historic or on-going software effort to compute a true productivity metric.

CAPE can provide a framework for objective management measurement across the SEI levels by performing many of the measurement and estimating functions of the four core functions. We will now consider a few examples.

Size

Software size is generally accepted as an important measure of the amount of resources needed to make a product. What is not universally accepted is a common denominator for the size measure. In addition to debates of the utility of SLOC, Instructions, Function Points, Modules, Feature Points, etc., there are further issues dealing with origin and class. Does the software need to be built? Does it exist? Is it off-the-shelf? Does any of it need to be modified? What does it do? Is it mission critical? Is it Automated Information System (AIS) software? These are important issues that must be captured along with any raw measure of size. They add depth dimensions necessary in differentiating the 1000 SLOC library Fast Fourier Transform for encryption analysis from the 1000 SLOC Attitude Determination for flight control.

Productivity Measurement



"Peeling the Onion"

The diagram that follows shows some of the size qualification issues that can be addressed with the PRICE S CAPE tool. Each modeled software entity, be it a module, a configuration item, or something between, can be apportioned into classes of function and amount of new (versus re-use) software design and/or code needed by class. The distribution process will provide composite qualified measures of the software size. Notice that the composite measures for generic function (APPL) and amount of new design/code are weighted according to software SLOC distribution into distinct functional classes.

Software Size Qualification

	APPL	Mix	NEWD	NEWC
User Defined	0.00	0.00	0.000	0.000
Store & Retrieve Data	4.10	0.25	0.600	0.600
Online Communications	6.16	0.20	0.750	1.000
Real Time	8.46	0.15	1.000	1.000
Interactive	10.95	0.00	0.000	0.000
Math	0.86	0.00	0.000	0.000
String Manipulation	2.31	0.40	0.250	0.250
Operating System	10.95	0.00	0.000	0.000
Sum	1.00			
	APPL	NEWD	NEWC	
	4.45	0.683	0.752	

- Functions Software Performs

- By Module, Component, or Cfg. Item

User Defined

Store & Retrieve Data

Online Communications

Real Time

Interactive

Math

String Manipulation

Operating System

- Amount of new S/W.

- Design vs. Code

- By function

OK

Cancel

Recall that the SEI recommended size measure is SLOC. The arguments for this measure as the size basis are strong. And, while this is a relatively simple measurement to determine after the fact, it has proven to be difficult to determine for estimating purposes. Function Points and Feature Points are touted as size measures that more directly address the software end product from the users or specifiers perspective. Depending on circumstances, any one of these size metrics may be the preferred measure for an estimator. The situation does not dictate adoption of one size measure over all others. As the figure below illustrates, it is possible to build a correlation between another metric (Function Points in this case) and SLOC.

The translation table shown is the default preset in PRICE S. Though it was built from industry wide data, employing the table as part of a level 2 KPA does require adaptation to organization specific translations.

SLOC and Function Point Correlation

Unadjusted Function Point Calculation				Value Adjustment Factor Calculation	
Function Type	Functional Complexity	Complexity Totals	Function Type Totals	General System Characteristics	Degree of Influence
Internal Logical Files	5	Low × 7 = 35	95	1. Data Communications	3
	3	Avg. × 10 = 30		2. Distributed Processing	0
	2	High × 15 = 30		3. Performance	3
External Interface Files	4	Low × 5 = 20	69		
	10	Avg. × 7 = 70			
	3	High × 10 = 30			
External Inputs	5	Low × 3 = 15	69		
	20	Avg. × 4 = 80			
	4	High × 6 = 24			
External Outputs	0	Low × 4 = 0	69		
	12	Avg. × 5 = 60			
	4	High × 7 = 28			
External Inquires	5	Low × 3 = 15	69		
	3	Avg. × 4 = 12			
	7	High × 6 = 42			
Unadjusted Function Point Count			491		
Function Point Count			477		

Function Point by SLOC Translation Table					
<input checked="" type="radio"/> Ada.....	71	<input type="radio"/> Cobol.....	105	<input type="radio"/> Pascal.....	
<input type="radio"/> Algol.....	105	<input type="radio"/> Compass.....	492	<input type="radio"/> PLI.....	
<input type="radio"/> Apl.....	32	<input type="radio"/> Corral-66.....	64	<input type="radio"/> Prime.....	
<input type="radio"/> Assembly.....	320	<input type="radio"/> Flad.....	32	<input type="radio"/> SPL1.....	
<input type="radio"/> Atlas.....	32	<input type="radio"/> Fortran.....	105	<input type="radio"/> High-1.....	
<input type="radio"/> Basic.....	64	<input type="radio"/> Item.....	25	<input type="radio"/> Mach.....	
<input type="radio"/> C.....	120	<input type="radio"/> Jovial.....	105	<input type="radio"/> 4th G.....	
<input type="radio"/> CMS-2.....	114	<input type="radio"/> Microcode.....	107	<input type="radio"/> Interp.....	

14. Facilitate Change	3
Total Degree of Influence	32
Translation	71.00
SLOC	33867

The important point about the table above is not the values for translation from Function Points to SLOC, but the framework that exists for tool users to record and use organization specific languages and values in managing translation measurement. These are the type of processes that lead to higher levels of maturity. Further illustration of CAPE tool use for adaptive measurement is seen in the PRICE S Feature Point Sizing utility on the next page. Notice the calibration factor. It is used to mold the sizing utility to organization experience in translating from features to SLOC.

Size Measurement And Adaptation

<input checked="" type="checkbox"/> Integration	<input checked="" type="checkbox"/> Design review	<input checked="" type="checkbox"/> Code walk thru
<input checked="" type="checkbox"/> Top down	<input checked="" type="checkbox"/> Module Testing	
OUTP 16	OUTS 8	OUTD 6
INPF 2	OUTF 9	SCRF 4
COPT 14	INPFV 26	COMVA 13
FBULK 1.00		
REQG 0.00		
SICAL 0.90	LANG C	TARSIZ 0

OK

Cancel

Go to Military

SLOC

14362

Effort

Labor hours was selected as the recommended effort measurement unit because it is the most consistently defined standard across the industry. Labor months are usually a function of organization human resource policies and practices and will vary even among divisions of the same corporation. Currency will vary even greater due to differing labor rates, overhead rates, general and administrative mark-ups, and currencies. Yet, within an organization, the relationships between labor hours and labor months and between labor hours and monetary equivalents will be well established and consistent. This is assured by both law and proper business accounting practices. So, once again a useful CAPE tool must be able to operate with any unit of effort measurement in order to deal effectively with measured performance and estimate future requirements.

An example of the effort cross reference table of PRICE S is illustrated on the next page. Referred to as a Financial Factors Table, it will contain the actual or negotiated labor and overhead rates for an organization. A series of tables would be created and used as appropriate to a particular estimating requirement.

Understanding the distribution of effort into project phases and software functions is identified as a KPA of maturity levels 2 and higher. This means that the organization must be able to properly manage each cost account to include all effort required to accomplish a task without overlapping into other accounts. The CAPE tools must also conform to this requirement. Why is this important? It is due to the lack of uniformity of effort across projects and also within project phases and functions. One popular model of the effort distribution for software product development is shown on the following page.

This distribution model is a good tool for general industry prediction. It provides a reasonable expectation for anyone in the software field, but a satisfactory key process for no one. What if requirements analysis has already been done? What if a project also requires effort for system test

Currency - Effort Conversion

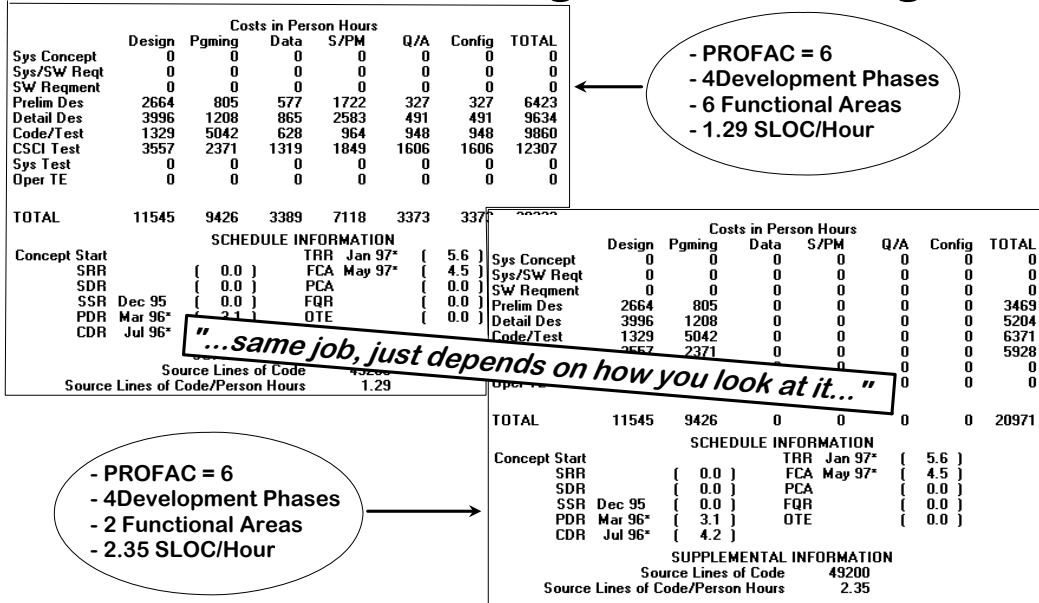
Financial Factors Table			
FELS Communications Division Jan. 1994 Rates			
	Direct	Overhead%	Overtime%
System Engineers	24.45	131.50	1.60
Design Engineers	22.60	131.50	1.60
Programmers	17.25	131.50	1.60
Quality Assurance	18.10	165.20	1.60
Configuration Mgmt	15.98	131.50	1.60
Program Mgmt	21.45	131.50	1.60
Documentation	18.65	131.50	1.60
<div style="display: flex; flex-direction: column; align-items: flex-end; gap: 5px;"> <div style="display: flex; gap: 10px;"> <div style="border: 1px solid gray; padding: 5px; width: 40px; text-align: center;">OK</div> <div style="border: 1px solid gray; padding: 5px; width: 40px; text-align: center;">Cancel</div> </div> <div style="display: flex; gap: 10px;"> <div style="border: 1px solid gray; padding: 5px; width: 40px; text-align: center;">Lock</div> <div style="border: 1px solid gray; padding: 5px; width: 40px; text-align: center;">Esc</div> </div> <div style="display: flex; gap: 10px;"> <div style="border: 1px solid gray; padding: 5px; width: 40px; text-align: center;">lifecycle</div> <div style="border: 1px solid gray; padding: 5px; width: 40px; text-align: center;">Dep</div> </div> <div style="display: flex; gap: 10px;"> <div style="border: 1px solid gray; padding: 5px; width: 40px; text-align: center;">Notepad</div> <div style="border: 1px solid gray; padding: 5px; width: 40px; text-align: center;">Load</div> </div> </div>			
Rate Time Unit	Escalate	Additional Costs (%)	Base Year (MMYY)
<input type="radio"/> Monthly	<input type="radio"/> Yes	Gen & Admin <input style="width: 40px;" type="text" value="11.80"/>	Labor Rate <input style="width: 40px;" type="text" value="194"/>
<input checked="" type="radio"/> Hourly	<input checked="" type="radio"/> No	Fee / Profit <input style="width: 40px;" type="text" value="0.00"/>	Economics <input style="width: 40px;" type="text" value="194"/>
		Cost of Money <input style="width: 40px;" type="text" value="0.00"/>	

and for operational test support? What do we do if our organization is only responsible for software design and implementation? These and many other issues like them require flexibility of tools to match tailored flexing of program plans. There is a secondary effect of flexible program plans that deals with rippling of complications and, fortunately, benefits from phase to phase and function to function. This will further distance the cookie-cutter model below from recorded experience.

Software Development Effort Distribution Model	
Phase	% of Total Effort
<i>Requirements Analysis</i>	15
<i>Preliminary Design</i>	15
<i>Detailed Design</i>	23
<i>Code and Unit Test</i>	22
<i>Component Integration & Test</i>	13
<i>Configuration Item Test</i>	12
<i>Total - All Tasks</i>	100

The following figure will help illustrate the importance of understanding what tasks and time spans are included in the effort recorded for a project. In the first case, the effort is assumed to span four development phases (from preliminary design to configuration item test) and six functional areas (software engineering design and programming, data items preparation, system engineering/program management, quality assurance, and configuration management/control). The second report is for the same program element, but includes only the software engineering and programming functions.

Effort Recording & Estimating



Notice that the conventional productivity measure (SLOC/Hour) is different for these two views of the same project. Though it is obvious that they would be when different categories of work are included, as they are here, how frequently do people define their domain of inclusion when discussing productivity? Not very often. Yet, it is precisely this type of attention that is required to move from Level 1. The PRICE S productivity factor (PROFAC) is determined from past organizational performance and provides a consistent metric, regardless of the domain of inclusion. Earlier, we stated that the Integrated Software Management KPA of Level 3 requires use of data from past and current projects to plan and estimate new projects. This KPA specifically calls for tracking and application to phase and functional breakouts similar to those illustrated in the PRICE S reports above. Once again, the theme is framework. Distribution of effort into the categories shown was done by adapting PRICE S to the past and current projects experience by phase and function. As will be illustrated in the next section, adaptation can be performed to the measure of schedule as well.

Schedule

Due to the high labor content of software projects, schedule has a direct correlation to the amount of effort needed for a project. Scheduling constraints can be anticipated for any software project. On one hand, a need will be critical, causing compression of time allotted for task completion. The next case may require stretching of schedule due to funding constraints or dominance of another system element (such as a processor) that leads scheduling. There is no characteristic of software that indicates what schedule dynamics are likely to be present - this is largely programmatic and need driven. And, ironically, the "hurry up" and "slow down" situations can prevail at different times on the same project. It is not uncommon to see rapid depletion of budget in the early phases of a tightly scheduled project eventually lead to slow down and re-planning in order to conserve funds.

The CAPE tools used for schedule measurement and estimating must be able to accommodate milestones. It is nice to know that a development started in March 1992 and ended July 1994. It is more useful to know that Requirements Analysis was completed May 1992, that preliminary

design started two weeks prior, and so on. This level of schedule treatment provides the means to objectively estimate new projects where phases are tailored to program objectives. The figure below shows the schedule adaptation (calibration) mechanism of PRICE S, followed by an MS-Project© interface display used in support of project tracking and planning.

Schedule Recording & Estimating

MULTIPLIERS	Cost						Schedule			Eliminate Phase	Penalty
	Des	Pgm	Data	SEPM	Q/A	CFM	Mult				
System Concept	1.00	1.00	1.00	1.00	1.00	1.00	1.00	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
System SW Req	1.00	1.00	1.00	1.00	1.00	1.00	1.20	<input type="checkbox"/>	<input type="checkbox"/>		
SW Req Anlys	1.00	1.00	1.00	1.00	1.00	1.00	1.20	<input type="checkbox"/>	<input type="checkbox"/>		
Prelim Design	1.00	1.00	1.00	1.00	1.00	1.00	0.90	<input type="checkbox"/>	<input type="checkbox"/>		
Detail Design	1.00	1.00	1.00	1.00	1.00	1.00	0.90	<input type="checkbox"/>	<input type="checkbox"/>		
Code/Test	1.00	1.00	1.00	1.00	1.00	1.00	0.90	<input type="checkbox"/>	<input type="checkbox"/>		
CSCI Test	1.00	1.00	1.00	1.00	1.00	1.00	1.00	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
System Test	1.00	1.00	1.00	1.00	1.00	1.00	1.00	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
Operational T&E	1.00	1.00	1.00	1.00	1.00	1.00	1.00	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
System I & T	1.00	1.00	1.00	1.00	1.00	1.00	SSR-FCA	<input type="checkbox"/>	<input type="checkbox"/>		

OK	Cancel
Inp	Load
Validate	Esc
lifecycle	Fin
Notepad	Dep
Sens. %	Lock
SMULT	1.00
CMULT	1.00
Hrs/Month	152.00
Decimals	1

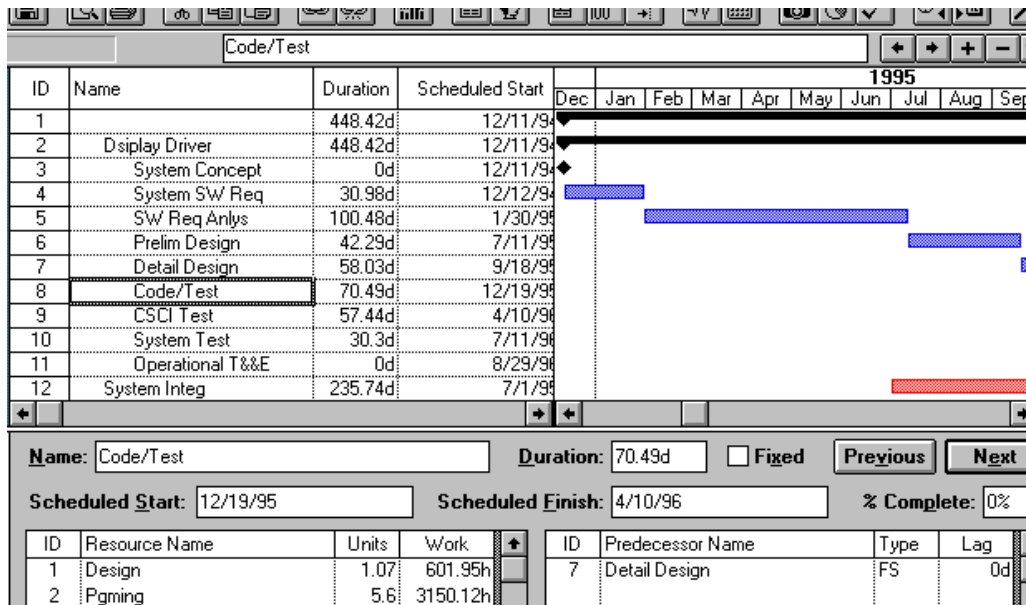
- Tailoring (Phase Elimination)
- Adaptation (Phase Multiplier)
- Calibration (Phase Penalty)
- Includes Integration & Support Phases

With this mechanism, one can adapt PRICE S to perform the following schedule measurement tasks, all of which are processes associated with maturity levels 2 and higher.

- Map recorded schedule performance to an estimating framework (Multiplier);
- Control variations in effort with variations in schedule (Penalty);
- Eliminate unwanted phases from recording or estimating (Phase Elimination);
- Flex System Integration and Test activities;
- Address Operating and Support phases (Lifecycle).

It is important to mention that the schedule interaction among phases is dynamic. That is, a stretch or compression of a phase, as defined by project milestones completion, will ripple into other phases. For instance, increasing requirements definition will certainly add effort and time to that phase, but the net result could be to reduce the expected effort and time for the project overall. In addition, well defined requirements are the first important step towards production of a high quality item. Therefore, it is not enough for a tool to simply conform to project phase break-outs. It must also inherently emulate phase interrelationships.

Project Management Interface



Quality

Quality is the ability of the final software product to perform it's designed mission. Though the measure of defects recommended by SEI fairly well quantifies quality, there is the tendency among users of software products to judge quality on the bases of ease of use and comprehensiveness of application. These issues deal with requirements definition and are not within the quality area addressed by this core measure. Here, we define quality within the scope of the established requirements only. Defects are errors (bugs, inadequate turn-around, etc.) in performance. The useful CAPE tool will provide a means to adapt to observed quality and estimate new project quality based on the adaptation.

PRICE S uses the mechanism of quality level to permit adaptation and influence the estimate of product quality (see figure on following page). This is something that can be applied at the configuration item level. Also, the defect population will be sensitive to the project scheduling. In general, devoting sufficient time to requirements definition will have a positive effect on quality. On the other hand, skimping on requirements and adding to testing will only serve to more fully disclose the number of defects. It is important to plan, as quality cannot be tested into a software product.

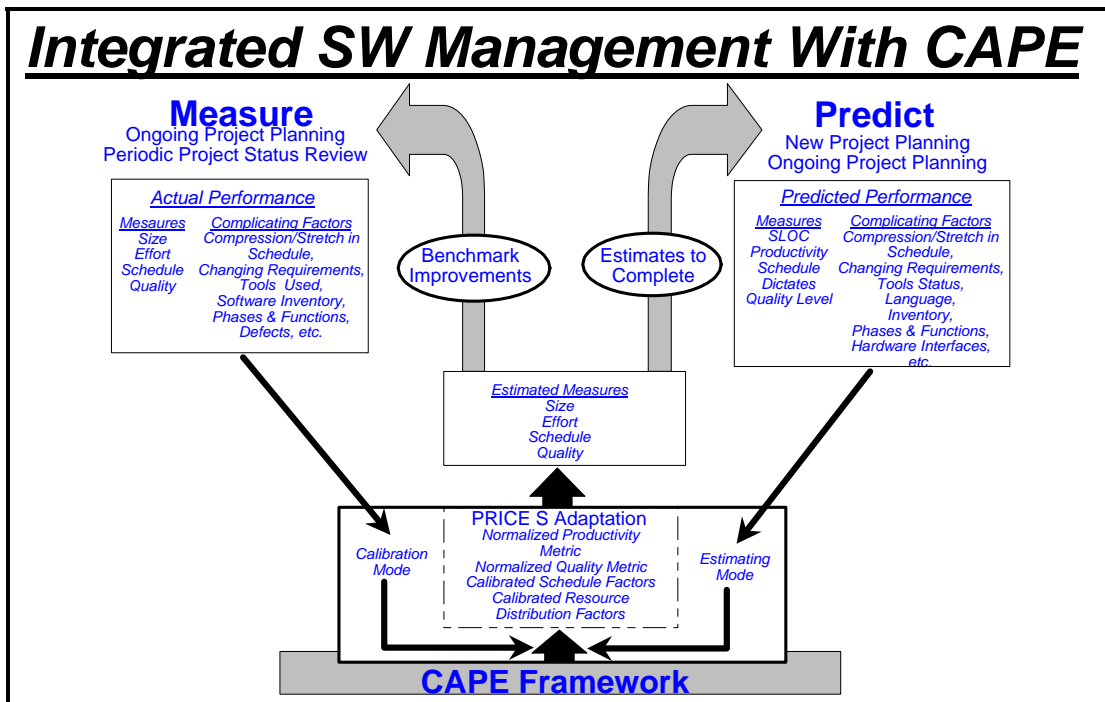
The step-by-step process to higher maturity levels can be categorized as "Measure, Predict, Improve, ...Measure, Predict, Improve, ...Measure, Predict, Improve, ...". Incorporating CAPE tools as part of an overall "Integrated Software Management Plan" facilitates these actions, ensuring objective measures and accurate predictions. The figure on page 17 illustrates the role of PRICE S in an integrated software management environment.

Quality Recording & Estimating

START	END	INSTALL	OK	Cancel	Organization Quality Performance
698	900	1			
GLEVEL	ELEVEL	QLEVEL			
0.00	0.00	0.80	Validate	Esc	

MPROFAC	EPROFAC	Tracker IC2 Dev./ 1.0 Development Item			
5.50	5.50	Costs in Person Hours			
2.3 YEAR OPERATIONAL LIFE					
		Maintenance	Enhancement	Growth	TOTAL
Design		500.5	0.0	0.0	500.5
Pgming		333.6	0.0	0.0	333.6
Data		155.4	0.0	0.0	155.4
S/PM		243.8	0.0	0.0	243.8
Q/A		197.0	0.0	0.0	197.0
Config		197.0	0.0	0.0	197.0
Estimated Defects	TOTAL	1627.4	0.0	0.0	1627.4
- Organization dependent					
- Schedule dependent					
- Product dependent					

		Summary	
		Source Lines of Code	12300
Acquisition Costs	11527.8 *	Initial Defects/KSLOC :	3.79
Support Cost	1627.4		
TOTAL	13155.2		



Measurement Tie-ins

The four core measures of size, effort, schedule, and quality are clearly associated with one another just as they are associated with the contexts presented here. This can create confusion and apparent contradictions among independently derived results of maturity level measurement and prediction. It is important that terms be clearly defined and understood in order to avoid confusion. We have seen a simple example of this with productivity under the topic of effort.

Carrying it a step further, recall the hypothesis of productivity improvement presented by Krasner (page 5). Compare this to a recent aerospace company report that suggests a 1.5 to 2 times productivity increase in transiting from Level 1 to Level 3 maturity. This surfaces at least a two to one difference in conclusion, depending upon source. Or does it? Probably not. One clue lies in the estimated schedule reduction of the Krasner table. Most likely, the Level 3 effort uses more off-the-shelf software than the Level 1 effort could. So, some of the reported productivity improvement and shorter schedule are probably due to less software engineering and programming required at Level 3. Let's test this with an example.

The test case is a 420K SLOC open architecture software project developed in a 4 build phase program over 5 years. Under a Level 1 condition, 40% of the software is off-the-shelf. Under Level 3, the off-the-shelf content is 75% and the project can be completed in one build phase. We use the framework of PRICE S to estimate the schedule and effort expected of development by organizations at each level. To do this, we drive the model by:

- Using a 2 to 1 organization productivity factor for the level 3 organization over the level 1;
- Using a 2 step complexity decrement for the level 3 organization;
- Using design and code inventory measures matching the off-the-shelf estimates;
- Estimating a 4 build versus 1 build trade-off.

The results of the analysis are tabulated below:

	Level 1 Org.	Level 3 Org.
Productivity Factor	X	2X
% COTS	40	75
Effort	518K Hours	152K Hours
Schedule	60 Months	23 Months
Quality	3.72 Defects/KSLOC	1.32 Defects/KSLOC

The analysis for this specific project does not completely align with either of the models represented above, but rather is somewhere between. This simple observation is precisely what should be expected and is the fundamental truth of proper CAPE use in a KPA - the standards proposed by others must be tested against the performance of each organization.

Summary

Simple measures of productivity that do not take into account real-life complicating factors are often misleading, thus hindering software maturity progression. CAPE can play an important role for an organization as it navigates towards higher software maturity levels. The lure of instant answers with canned techniques, though appealing, should be avoided. As the example above illustrates, promoted standards can provide general expectations industry wide. But, the process an organization must pursue requires a CAPE tool that is adaptable and can evolve with the growth of detailed organization knowledge.

References

1. *Applied Software Measurement Course*, SPC-93005-MC, Version 01.02.04, March 1994, Software Productivity Consortium, Inc., SPC Building, 2214 Rock Hill Road, Herndon, VA. 22070.
2. *Software Measurement Guidebook*, SPC-91060-CMC, Version 02.00.02, December 1992, Software Productivity Consortium, Inc., SPC Building, 2214 Rock Hill Road, Herndon, VA. 22070.
3. *The SEI Core Measures: Background Information and Recommendations for Use and Implementation*, Anita D. Carleton, Dr. Robert E. Park, and Wolfhart B. Goethert, Software Engineering Institute - Carnegie Mellon University, Pittsburgh, PA, 15213. 412-268-7718
4. *Capability Maturity Model for Software (CMU/SEI-91-TR-24)*, Mark C. Paulk, Bill Curtis, Mary Beth Chrissis, et. al., Software Engineering Institute - Carnegie Mellon University, Pittsburgh, PA, 15213.
5. *Proceedings of Software Improvement Conference*, December 1990, Education Foundation of the Data Processing Management Association, Washington, DC.
6. *IEEE Software*, July 1993, Dion, Raytheon.
7. *IEEE Software*, July 1991, Humphrey, Snyder, and Willis, Hughes.
8. *International Conference on Software Engineering*, 1993, Wohlwend, Schlumberger.
9. *Crosstalk*, November 1992, Lipke and Butler, USAF.
10. *Practical Software Metrics for Project Management and Process Improvement*, 1992, Grady, Hewlett Packard.

Author

Bruce E. Fad is a customer consultant as well as Business Development Manager for Lockheed Martin PRICE Systems. He has been involved in estimating for 20 years as an analyst, method developer, and group leader. He has performed cost analyses of commercial and military hardware and software systems development, production, and operation from the perspectives of government buyer, industry supplier, independent evaluator, and litigation expert witness. In addition, Bruce has over 10 years hands-on management experience in software development and maintenance, business development, training, and customer support.