

Realizing the Benefits of the CMMISM with the CeBASE Method

Barry Boehm and Dan Port, USC

Victor Basili, U. of Maryland

Abstract

Future systems will be increasingly software-intensive, but the type of software development they will need is not well covered by current development and maturity models such as the waterfall model and Software Capability Maturity Model[®] (CMM[®]). Future development of software-intensive systems will need situation-specific balancing of discipline and flexibility to address such issues as COTS, open source, distribution, mobility rapid change, agents, collaboration support, and simultaneous achievement of rapid development and high dependability.

This paper shows how the CMMI's integration of modern systems engineering, software engineering, and integrated process and product development concepts provides a framework for redressing the shortfalls of the Software CMM, and for enabling projects and organizations to achieve the right balance of discipline and flexibility for their particular situations.

But the CMMI has shortfalls of its own. It provides little guidance on how to define and execute specific processes for a specific project or organization. The paper summarizes various process model generators for software intensive systems such as the spiral, the Rational Unified Process (RUP), MBASE, and the CeBASE Method. It concludes that the CeBASE Method best covers the full range of concerns in the CMMI, resolves its practice-focus shortfalls, and covers additional best practices not in the CMMI, such as business case analysis, requirements prioritization, and evolution requirements.

1. How the CMMI Redresses Shortfalls in the Software CMM

The current Capability Maturity Model Integration (CMMI) model [SEI, 2000; Ahern et al., 2001] is intended to replace the Software CMM Versions 1.1, [Paulk et al., 1994] and 2.0 draft C, [SEI, 1997a], the Systems Engineering Capability Model, [EIA, 1998], and the Integrated Product Development CMM draft Version 0.98, [SEI, 1997b].

As such, the CMMI has arrived just in time to redress some serious shortfalls in the Software CMM. We say "just in time," because

- More and more systems are becoming software-intensive; even bullets, [Etter, 2001].
- More and more systems are being developed in an environment of rapid change, requiring integrated development of software, hardware, and human elements of systems.

SM CMMI and Capability Maturity Model Integration are service marks of Carnegie Mellon University.

[®] Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office.

- The Software CMM, particularly the currently-used Version 1.1, has several shortfalls which push software engineering toward a reactive and sequential role in systems engineering.

For example, the first Ability to Perform in the first Key Process Area of the Software CMM v. 1.1 states, “Analysis and allocation of the system requirements is not the responsibility of the software engineering group but is a prerequisite for their work.” This and other similar Software CMM elements have reinforced several negative trends, [Boehm, 2000]:

1. A preference among many software people and organizations to practice software engineering as an abstract logical exercise independent of system and value concerns. For example, in a recent survey of 16 books on object-oriented design, only 6 had the word “performance” in their index, and only 2 had the word “cost.”
2. A system engineering social structure in which the hardware, human factors, and system engineers sit at the center table working out the system definition, while the software engineers sit along the wall waiting for someone to give them requirements to turn in to code [Boehm, 1994].
3. Many system disasters or near-disasters where software engineers were presented with requirements that were unnecessarily difficult, expensive, or time-consuming. An example from [Boehm, 2000] was a large TRW-government project that locked itself into a 1-second response time requirement. Two years later, it found that the best software solution would cost \$100 M; that \$30 M software solutions with 4-second response times were available; and that most users could do their job quite well with a 4-second response time. Of course, by this time, the project was well up the Degree of Commitment Curve in [Blanchard and Fabrycky, 1998:37], and well down the Ease of Change curve, so the change of architecture and of hardware commitments was quite expensive.

The CMMI uses several process areas to redress this shortfall. Its Integrated Project Management and Integrated Teaming process areas emphasize software experts’ participation in system engineering via “Identifying the knowledge, skills, and functional expertise required to perform team tasks” and “Resolving issues with the relevant stakeholders.” Its Requirements Development process area emphasizes “Validating requirements to ensure the resulting product will perform appropriately in its intended use environment.” Thus, the CMMI provides the software engineers a seat at the center table. It also identifies the additional activities necessary to integrate system and software engineering, and to avoid the past problems caused by decoupling them.

2. How New Process Generators Redress CMMI Shortfalls

The CMMI provides an appraisal framework and criteria for determining if a project’s or organization’s processes satisfy the critical success factors identified in the CMMI process areas. Also, in its Integrated Product Management process area, its goal that “The project is conducted using a defined process that is tailored from the organization’s set of standard processes,” emphasizes that there is no “one-size-fits-all” process that fits all situations.

The CMMI’s main shortfalls (by intent and not omission) are that it does not provide guidelines for tailoring an appropriate process, or for integrating process considerations with product and product-line considerations.

At the project level, these shortfalls are being redressed by new process generator approaches such as the Rational Unified Process (RUP) [Royce, 1998; Kruchten, 1999; Jacobson et al., 1999], and Model-Based (System) Architecting and Software Engineering (MBASE) [Boehm and Port, 1999; Boehm, 2001]. Both RUP and MBASE build on the risk-driven Spiral Model process generator [Boehm, 1988; Boehm and Hansen, 2001], and use risk considerations to tailor a project's process and product definitions to its particular objectives and constraints. RUP is strongest in the areas of guidance for object-oriented software analysis and design processes using the Unified Modeling Language. MBASE builds on these strengths to provide extensions addressing software-intensive system engineering, Integrated Product and Process Development, stakeholder cost-benefit and value considerations, and avoidance of model clashes among the project's process, product, property, and success models.

3. An Overview of MBASE

This section discusses the main features of MBASE as an extension of RUP: model clash avoidance; model integration and process framework; stakeholder win-win requirements negotiation and project management; and life cycle anchor point milestones and pass-fail criteria. It also summarizes MBASE usage and experience over the last five years on over 100 real-client projects [Boehm and Port, 2000].

3.1 MBASE and Model Clash Avoidance

Particularly for an invisible product such as software, projects make use of various process, product, property, and success models to guide its progress. *Process models* can include the waterfall model (sequential determination of the system's requirements, design, and code); evolutionary development (development of an initial core capability, with full definition of future increments deferred); incremental development; spiral development; rapid application development; adaptive development; and many others.

Product models can include various ways of specifying operational concepts, requirements, architectures, designs, and code, along with their interrelationships; for example the object-oriented design models specified within the Rational Unified Process, e.g. class and sequence diagrams, use-cases, etc.

Property models can include models of desired or acceptable cost, schedule, performance, reliability, security, portability, evolvability, reusability, etc., and their tradeoffs, e.g., Constructive Cost Model II (COCOMO II).

Success Models can include correctness (satisfying the specified requirement), organization and project goals, stakeholder win-win, business-case, Goal-Question-Metric (GQM), or others such as IKIWISI (I'll know it when I see it: a frequent response when users are asked to specify user-interface requirements).

Software architects and product developers are generally familiar with the concept that trying to integrate two or more arbitrarily selected products or product models can lead to serious conflicts and disasters. Some examples are mixing functional and object-oriented components, or the architectural style clashes you can encounter when integrating commercial off-the-shelf (COTS) products (see [Garlan et al., 1995] for a well-described case study involving factors of four and five overruns in schedule and effort). Software process people are similarly familiar

with the serious effects of trying to coordinate organizations with clashing process models such as top-down/bottom-up or CMM Level 1/Level 5.

However, relatively few people are aware of the extent to which projects can run into trouble because they choose incompatible combinations of software process, product, property, and success models. Such model clashes are not only frequent and severe, but they are also hard to diagnose because they derive from different sources, and involve mismatched assumptions lying deep below the project's written plans and specifications.

A good Department of Defense (DoD) example was a deeply troubled project encountered during the National Research Council *Ada and Beyond* study [Boehm et al., 1997]. This project inherited a commitment to a waterfall process model via its organization's commitment to DoD-STD-2167. It also inherited a commitment to COTS-based product model through the need to comply with a Secretary of Defense mandate.

The waterfall model assumes that the requirements determine the capabilities, and the project had contracted for a two-second response time requirement. However, the contractor found that none of the available COTS capabilities could process the system workload with a two-second response time, and was proceeding to develop an expensive custom software system to meet the two-second requirement. The customer wanted the COTS-directive product model to take precedence, and to have the available COTS capabilities determine the performance requirements.

Besides this difficult model clash, the project had also inherited an Ada-based product model via the DoD Ada mandate. This clashed with the COTS model, since some attractive COTS solutions did not have adequate Ada bindings. Further, none of the approaches were compatible with the project's success model of producing an initial operational capability in 36 months, or with an additional property model, which was being adopted by the organization. The model was *Cost As Independent Variable*, which would have been difficult to satisfy when the project already had at least four existing independent variables.

This example covers only a small subset of the model clashes a project can encounter. Further discussions, examples, and case studies can be found in [Boehm and Port, 1999; Boehm, Port, and AlSaid, 2000]. We have been able to use MBASE to help other large government and commercial projects to diagnose and avoid serious model clashes, and have worked with smaller companies to develop lightweight versions of MBASE to balance discipline and flexibility on rapid-development projects. After describing MBASE in sections 3.2-3.4, we will summarize how its use could have avoided the DoD project model clash dilemmas noted, followed by a summary of MBASE usage to date.

3.2 MBASE Model Integration Framework and Process Framework

Figure 1 summarizes the overall model integration framework used in the MBASE approach to ensure that a project's success, product, process, and property models are consistent and well integrated. At the top of Figure 1 are success models, illustrated with several examples, whose priorities and consistency should be considered first as they tend to drive the selection and use of other models (including other success models).

Thus, if the overriding top-priority success model is to “demonstrate a competitive agent-based electronic commerce system on the floor of the COMDEX trade show in nine months,” this constrains the ambition level of other success models. It would be a major schedule risk to insist on provably correct code or a fully documented system. The nine-month schedule constraint is most critical because the system will lose most of its value if it is not available to compete for early market share at COMDEX.

The risk of schedule overrun also determines many aspects of the product model (architecture designed to easily shed lower-priority features if necessary to meet schedule), the process model (Schedule as Independent Variable), and various property models (portable and reliable enough to achieve a successful demonstration). The achievability of the 9-month success model needs to be evaluated with respect to the other models. Figure 1 shows that the choices of process and product models need to be evaluated by having the success model provide evaluation criteria for the product milestone artifacts, and provide preconditions and post-conditions (entry and exit criteria) for the process milestones.

In the 9-month COMDEX demonstration example, a cost-schedule estimation model would relate various product characteristics (sizing of components, reuse, product complexity), process characteristics (staff capabilities and experience, tool support, process maturity), and property characteristics (required reliability, cost constraints) to determine whether the product capabilities achievable in 9 months would enable competitive success. Thus, as shown at the bottom of Figure 1, a cost and schedule property model would be used for the evaluation and analysis of the consistency of the system’s product, process, and success models.

In other cases, the success model would make a process model or a product model the primary driver for model integration. An IKIWISI (I’ll know it when I see it) success model might initially establish a prototyping and evolutionary development process model, with most of the product features and property levels left to be determined by the process. A success model focused on developing a product line of similar products would initially focus on product models (domain models, product line architectures), with process models and property models subsequently explored to perform a business-case analysis of the most appropriate breadth of the product line and the timing for introducing individual products (in Figure 1, using a business-case success model as the primary evaluation criterion for the domain-breadth of the product model).

Figure 2 provides an overall process framework for the MBASE approach. The primary drivers for any system’s (or product line’s) characteristics are the value propositions of its key stakeholders. These generally include the system (taken below to mean “system or product-line”) users, customers, developers, and maintainers. Key stakeholders can also include strategic partners, marketers, operators of closely coupled systems, and the general public for such issues as safety, security, privacy, or fairness.

The critical interests of these stakeholders determine the priorities, desired levels, and acceptable levels of various system success criteria. These are reflected in the success models for the system, such as stakeholder win-win, business case, organization and project goals, operational effectiveness models, or IKIWISI. These in turn determine which portions of an applications domain and its environment are relevant to consider in specifying the system and its development and evolution process. The particular objective is to determine a system boundary,

within which the system is to be developed and evolved; and outside of which is the system environment (and context).

For example, in our COMDEX electronic commerce demo application, the driving success model is the 9-month schedule. The system boundary or domain-breadth would be determined by the most cost-effective set of capabilities that could be developed in 9 months. Thus, a credit card verification capability might be considered outside the initial system boundary, although it would be needed later.

This latter point illustrates how boundaries might (and will likely) change over time, particularly in the face of evolving success models (e.g., the nature of a competitive e-commerce system). For example, if a compatible COTS credit card verification capability became available and easy to integrate, it could be added within the system boundary. Thus the domain-breadth for the demo system would be very much determined by the available COTS products that could be tailored, integrated, and built upon.

Determining the appropriate combination of COTS products and extensions could take several win-win spiral cycles of experimental prototyping and risk resolution, in concert with cost-schedule modeling to determine how much capability would be feasible to develop in nine months. The appropriate process model would be Schedule As Independent Variable (SAIV) [Boehm et al., 2002], which adds further product model constraints, such as the need to prioritize features and to design the system architecture for ease of adding or dropping marginal-priority features in order to minimize the risk of not meeting the nine-month schedule.

3.2.1 A Different Balance of Discipline and Flexibility: Safe Air Traffic Control

With a different set of stakeholders and success models, the same MBASE process framework in Figure 2 will produce a different balance of discipline and flexibility. In an air traffic control system, for example, the key stakeholders will include the airplane passengers and various regulatory bodies whose success models involve a very high level of system safety. In this case, the success models will reject high-risk product models including unreliable COTS products. And the process models will include considerably more discipline to eliminate safety risks at the requirements, architecture, design, and code levels. The key property model will focus on safety rather than schedule, although schedule considerations might still affect the timing of various increments of system capability. Thus, the different risk patterns imposed by the stakeholders and their success models will produce different sequences of product capabilities, and processes with different but situation-appropriate balances of discipline and flexibility.

3.3 MBASE and Stakeholder WinWin

A common element in the e-commerce example and the air traffic control example is the need to reconcile the key stakeholders' success models. To achieve this, a stakeholder win-win negotiation process becomes a key step in each spiral cycle of the MBASE approach, as shown in Figure 3.

In the COMDEX application, for example, the initial spiral cycle would focus on evaluating COTS products and scoping the overall system to be buildable in 9 months. In a subsequent spiral cycle, the next-level stakeholders would include representative users of the e-commerce system, and the reconciliation of their win conditions would include prototyping of the user

interface to eliminate the risk of showing up at COMDEX with an unfriendly user interface. The MBASE tool support includes a groupware system called Easy WinWin, which enables distributed stakeholders to enter their win conditions and to negotiate mutually satisfactory (win-win) agreements with other stakeholders, [Boehm et al., 2001].

The Win-Win spiral model in Figure 3 provides another view of how risk considerations are used to reconcile stakeholder success conditions in terms of product, process, and property models. A complementary view was shown in Figure 2, which also identifies the WinWin Spiral Model's role in guiding the early feedback cycles involved in defining and reconciling the system's domain, product, process, and property models.

3.4 MBASE and Life-Cycle Anchor Points

The stakeholder win-win approach helped resolve one of the problems with the original spiral model (the white portion of Figure 3): the problem of how to determine the next-level objectives, constraints, and alternatives that start each spiral cycle. We found through experience that the win-win agreements that reconcile stakeholder win conditions (determined in the gray portion of Figure 3) generally take the form of objectives, constraints and alternatives.

Another problem with the original spiral model was its lack of intermediate milestones for assessing progress and ratifying stakeholder commitments to proceed. This problem was satisfactorily resolved by the definition of three key life cycle “anchor point” milestones [Boehm, 1996].

The specific content of the first two anchor point milestones is summarized in Table 1. It includes increasingly detailed, risk-driven definitions of the system’s operational concept, prototypes, requirements, architectures, life cycle plan, and feasibility rationale. For the feasibility rationale, property models are invoked to help verify that the project’s success models, product models, process models, and property levels or models are acceptably consistent.

The first milestone is the Life Cycle Objectives (LCO) milestone, at which management verifies the basis for a business commitment to proceed at least through an architecting stage. This involves verifying that there is at least one system architecture and choice of COTS/reuse components which is shown to be feasible to implement within budget and schedule constraints, while satisfying key stakeholder win conditions and generating a viable business case.

The second milestone is the Life Cycle Architecture (LCA) milestone, at which management verifies the basis for a sound commitment to product development and evolution (a particular system architecture with specific COTS and reuse commitments which is shown to be feasible with respect to budget, schedule, requirements, operations concept and business case; identification and commitment of all key life-cycle stakeholders; and elimination of all critical risk items). The AT&T/Lucent Architecture Review Board technique [Marenzano, 1995], is an excellent management reviews approach involving the LCO and LCA milestones. It is similar to the highly successful recent DoD best practice of software Independent Expert Program Reviews [DSB, 2000].

The third anchor point milestone is the system’s Initial Operational Capability (IOC), defined further in Boehm [1996]. The LCO, LCA, and IOC have become the key milestones in the RUP, while MBASE has adopted the RUP’s phase terminology (Inception, Elaboration, Construction,

Transition). There are many possible minor milestones (adjusted to the particular project as needed) that may lie between LCO and IOC and several post-deployment milestones beyond IOC. In general, though, the LCO, LCA, and IOC anchor points work quite well as the intermediate milestones of an annual or 18-month post-deployment upgrade. Table 2 summarizes the pass/fail criteria for the LCO, LCA, and IOC anchor points:

As seen in Table 2, the focus of the LCO review is to ensure that *at least one* architecture choice is viable from a business perspective. The focus of the LCA review is to commit to *a single* detailed definition of the architecture and the other review artifacts. The project must have either eliminated all significant risks or put in place an acceptable risk-management plan. The focus of the IOC review, also called the Transition Readiness Review, is to ensure that the initial users, operators, and maintainers (generally equivalent to beta-testers) are fully prepared to successfully operate the delivered system. If the pass/fail criteria for any review are not satisfied, the package should be reworked (or, if the problems are intractable, the project should be cancelled).

We determined these anchor point milestones as common commitment points across commercial, aerospace, and government organizations when searching with our USC Center for Software Engineering Affiliates for a set of common milestones for referencing COCOMO II cost and schedule estimates. They work well as common commitment points across a variety of process model variants because they reflect similar commitment points during one's lifetime.

The LCO milestone is the equivalent of getting engaged, and the LCA milestone is the equivalent of getting married. As in life, if you marry your architecture in haste, you and your stakeholders will repent at leisure (if, in Internet time, any leisure time is available). The third anchor point milestone, the Initial Operational Capability (IOC), constitutes an even larger commitment: It is the equivalent of having your first child, with all the associated commitments of care and feeding of a legacy system.

To return to our DoD-2167/COTS/Ada/deadline/CAIV model clash example in section 3.1, at the latest, the project would have failed its LCO milestone review by being unable to demonstrate that a COTS-based architecture could satisfy the 2-second response time requirement. Even earlier, through, this model clash would have been picked up by the MBASE process framework in Figure 2, in feeding back to the stakeholders the need to revise their success models (COTS-based product and 2-second response time) to permit a clash-free solution. This would involve additional Win Win spiral cycles to determine a mutually satisfactory (win-win) combination of features, budgets, schedules, increments, and COTS choices.

3.5 MBASE Usage Experience

For the past five years, USC has used and refined MBASE extensively within its two semester graduate software engineering course. The students work on web-based electronic services projects for real USC clients (frequently a digital library application for the university information services division) from initial system definition through transition, utilizing a specialized form of MBASE. This specialization includes particular tools and models such as Easy WinWin, Rational Rose, MSPProject, and elements of the Rational Unified Process. Over 100 real-client projects have used MBASE, and over 90% have delivered highly satisfactory

products on very short fixed schedules. The annual lessons learned have been organized into an extensive set of usage guidelines and an Electronic Process Guide [USC-CSE, 2001], all accessible at <http://sunset.usc.edu/research/MBASE>. In the spring of 1999, MBASE was used in both the undergraduate and graduate software engineering courses at Columbia University. Although these were single semester courses, MBASE was successfully adapted to help student teams complete a full project lifecycle for real clients.

Within industry, Xerox has adopted many elements of MBASE to form its “time to market” process, including the use of the LCO and LCA anchor points as synchronization points for the hardware and software portions of their printer product definitions. The Xerox approach is similar to that suggested by [Rechtin and Maier, 1997] in synchronizing software spirals with waterfall hardware increments.

As mentioned previously, Rational has adopted the LCO, LCA, and IOC anchor points within their Rational Unified Process (RUP); while MBASE adopted Rational’s Inception-Elaboration-Construction-Transition phase definitions. Also, the software company C-Bridge has mapped their “define, design, develop, deploy” rapid development methodology for e-commerce systems to the MBASE spiral model.

The Internet startup company Media Connex adopted MBASE and used Easy WinWin to establish win-win relationships among their key stakeholders. Each of these companies converged on different balances of discipline and flexibility to satisfy their stakeholders' success models. Additionally, there are numerous companies and organizations directly making use of MBASE elements within their project development efforts. For example, the U.S. Army Tank and Automotive Command has used Easy WinWin and other MBASE elements to reconcile its software technology organizations' process and product strategies [Saboe, 2002].

4. The CeBASE Method and How it Redresses MBASE/RUP Shortfalls

MBASE and RUP provide strong guidance for satisfying the project-oriented parts of the CMMI. Their main shortfall with respect to CMMI is that, as project-level methods, they do not address the six organization level process areas in the CMMI (Organizational Process Focus, Definition, Performance, et al.).

However, as USC and the University of Maryland have been collaborating within our Center for Empirically-Based Software Engineering (CeBASE) on a unified approach to empirical software engineering, we have found that USC’s project-level MBASE method and Maryland’s organization-level Experience Factory and Goal-Question-Metric method [Basili-Caldeira-Rombach, 1994a; 1994b; van Solingen-Berghout, 1999] are highly compatible. The resulting unified CeBASE Method currently comes close to implementing both the project and organizational levels of the CMMI. We are currently extending it to cover the full CMMI.

As we explored the details of Maryland’s Experience Factory (EF) and Goal-Question-Metric (GQM) approaches and USC’s MBASE approach, we found that they were expressing very similar principles and practices. The Spiral Model’s initial focus on system objectives was essentially the same as GQM’s initial focus on organizational goals and MBASE’s focus on tracing system goals back to organizational goals. The Experience Factory’s focus on organizational learning to understand a system’s operational stakeholders and their goals

corresponds strongly with MBASE's stakeholder win-win approach to mutual stakeholder understanding and development of a shared system vision.

MBASE's focus on determining and reconciling a project's process, product, property, and success models corresponded strongly with EF-GQM's focus on using models to capture experience and to understand multiple-goal relationships. The Experience Factory's strong focus on continuous monitoring and control corresponded strongly with MBASE's focus on continuous monitoring and control as a way for projects to cope with rapid changes in their environment, technology options, and competitive marketplace.

4.1 The CeBASE Method Framework

Overall, then, we found that both EF-GQM and MBASE could be integrated into a common CeBASE Method. Its framework is organized around a trio of common strategic themes, as shown in Figure 4. These three themes are the stakeholders' *shared vision* for the organization or project; risk-driven *plans* for process, product and people; and continuous *monitoring and control*. As seen in Figure 4, these themes express both the operation of EF-GQM at the organizational level and the operation of MBASE-GQM at the project level. Figure 4 also shows how progress/plan/goal mismatches feed continuous learning and improvement at both the organizational and project levels; and how learning at each level is used to condition learning at the level above or below.

4.1.1 Example of CeBASE Guidelines: Shared Vision

The CeBASE project-level and organization-level Shared Vision guidelines are quite similar. Their main difference is one of context: the project-level Shared Vision has the organization-level Shared Vision as context and shows traceability to it, but not vice versa. Figure 5 shows the Table of Contents and example text from the project-level Shared Vision. In the CeBASE Method, it is the first item to be drafted. It sets the stage for subsequent Inception Phase prototyping and stakeholder win-win requirements negotiation. The Benefits Realized and Results Chain sections are adapted from the DMR Consulting Group's excellent Benefits Realization Approach [Thorp, 1998].

4.2 CeBASE Method Coverage of the CMMI

4.2.1 Example Mapping: Requirements Development

To test its coverage of critical issues, we have done a mapping of the CeBASE Method onto the CMMI's 24 Process Areas, using the CMMI summary tables in [Ahern et al., 2001]. Figure 6 shows the mapping for one of the key process areas discussed above, Requirements Development.

The acronyms refer to elements of the CeBASE framework shown in Figure 4: OSV is Organization Shared Vision; PSV is Project Shared Vision; PP-1 is item 1 of the Project Plan, the Life Cycle Objectives/Life Cycle Architecture Package; OCD is Operational Concept Description; SSRD is System and Software Requirements Definition; SSAD is System and Software Architecture Description; LCP is Life Cycle Plan; FRD is Feasibility

Rationale Description; TP is Test Plan; TRP is Transition Plan. In addition, there are two CMMI acronyms: SG is a Specific Goal and SP is a Specific Practice for the Requirements Development process area.

Overall, the mapping indicated that the CeBASE Method covered the CMMI goals and practices well. It provided the CeBASE team with some action items, such as restoring the Operational Scenarios section of the OCD and elaborating the Transition Plan guidelines. Most significantly, though, it identified items that we have found important for Requirements Development that were not in the CMMI: a Business Case justifying the need for required features; prioritization of requirements; and coverage of project, level of service, and evolution requirements (to avoid point-solution architectures).

4.2.2. Overall CeBASE Method Coverage of the CMMI

Overall, we found not only a strong correspondence, but also an almost complete coverage of the CMMI's practices by the organizational and project components of the CeBASE Method. We are extending the CeBASE Method to cover the specific CMMI processes not currently covered. A summary of the percentage coverage of the CMMI process areas by the CeBASE method is shown in Table 3. The "+" annotations in Table 3 indicate that the CeBASE method's coverage goes considerably beyond that of the CMMI. For example, it covers not just an organizational process focus but also an organizational product and people focus. The "-" annotations in Table 3 indicate that some areas in the CeBASE Method still remain to be fleshed out, such as detailed guidelines for organizational training plans, although they are covered in principle.

The CeBASE Method also provides a prescriptive approach for an organization to use in tailoring the CMMI's generic practices to its particular culture, environment, and value propositions. Thus, an e-commerce organization's value propositions (rapid time to market, rapid adaptation to change) will cause it to adopt more flexible processes, but such elements as the Anchor Point milestones will balance this flexibility with sufficient discipline to keep the overall process under control. And the value propositions of an organization developing safety-critical products or services will cause it to emphasize more rigorous specifications, processes, and practices, but in ways that enable it to cope with rapid change (e.g., by capturing evolution requirements; by architecting systems to accommodate future change; by building in buffer periods to synchronize and stabilize processes [Cusumano and Selby, 1996]; or to adapt to potential schedule or budget slips by dropping lower-priority product features [Boehm et al., 2002].)

Another point worth emphasizing is that the Experience Factory component of the CeBASE Method supports a continuous vs. staged approach to process improvement. You don't need to be a level 4 organization to begin realizing significant benefits from organizational innovation or causal analysis.

5. Conclusions

Figure 7 summarizes the distinctions among maturity models such as the Software CMM and the CMMI; process models such as the waterfall model; and process model generators such as MBASE, RUP, and the CeBASE Method. It shows where each model fits with respect to

organizational focus (project vs. organization) application focus (software vs. system), and operational focus (practice vs. assessment).

From Figure 7, we can see that the Software CMM had shortfalls in both applications focus (software, not systems) and operational focus (assessment, not practice). We can also see that solutions focused on redressing one of the two shortfall dimensions will still have shortfalls of its own in the other dimension.

In terms of future software intensive system challenges, the ability to balance discipline and flexibility is critically important to the development of highly dependable software-intensive systems in an environment of rapid change. The CMMI's risk-management orientation enables its users to apply risk considerations to determine how much discipline and how much flexibility is enough in a given situation. The risk-driven nature of the spiral model and MBASE enables them to achieve a similar balance of discipline and flexibility. When these project-level approaches are combined with the organization-level approaches in the Experience Factory, the result is a unified CeBASE Method which currently implements most of the CMMI, which is being extended cover the full CMMI, and which has a strong track record of continuous process improvement at USC's and UMD's Software Engineering Laboratories and industry adapters elsewhere.

6. Acknowledgements

We would like to acknowledge the support of the National Science Foundation in establishing CeBASE, the DoD Software Intensive Systems Directorate in supporting its application to DoD projects and organizations, and the affiliates of the USC Center for Software Engineering and the University of Maryland's software engineering program for their contributions to MBASE and CeBASE.

7. References

D. Ahern, A. Clouse, and R. Turner, CMMI Distilled, Addison Wesley, 2001.

V. Basili, G. Caldeira, and H. D. Rombach, The Experience Factory, in J. Marciniak (ed.), Encyclopedia of Software Engineering, Wiley, 1994a.

V. Basili, G. Caldeira, and H. D. Rombach, The Goal Question Metric Approach, in J. Marciniak (ed.), Encyclopedia of Software Engineering, Wiley, 1994b.

B. Blanchard and W. Fabrycky, Systems Engineering and Analysis (3rd Ed.), Prentice Hall, 1998.

B. Boehm, A Spiral Model of Software Development and Enhancement, Computer, May 1988, pp. 61-72.

B. Boehm, Integrating Software Engineering and Systems Engineering, Journal of INCOSE, January 1994.

B. Boehm, Anchoring the Software Process, IEEE Software, July 1996, pp. 73-82.

- B. Boehm, Unifying Software Engineering and Systems Engineering, Computer, March 2000, pp. 114-116.
- B. Boehm et al., Ada and Beyond: Software Policies for the DoD, National Academy Press, 1997.
- B. Boehm and W. Hansen, The Spiral Model as a Tool for Evolutionary Acquisition, Cross Talk, May 2001.
- B. Boehm, D. Port, Escaping the Software Tar Pit: Model Clashes and How to Avoid Them, ACM Software Engineering Notes, January 1999, pp. 36-48.
- B. Boehm, D. Port, Balancing Discipline and Flexibility with the Spiral Model and MBASE, Cross Talk, December 2000, pp. 23-28.
- B. Boehm, D. Port, et al., Guidelines for Model-Based (System) Architecting and Software Engineering (MBASE), Version 2.3, 2001 (<http://sunset.usc.edu/research/MBASE>)
- B. Boehm, D. Port, and M. AlSaid, Avoiding the Software Model Clash Spider Web IEEE Computer, November 2000, pp. 120-122.
- B. Boehm, D. Port, L. Huang, and A. W. Brown, Using the Spiral Model and MBASE to Generate New Acquisition Process Models: SAIV, CAIV, and SCQAIV, Cross Talk, January 2002, pp. 20-25.
- B. Boehm, P. Gruenbacher, and R. Briggs, Developing Groupware for Requirements Negotiation: Lessons Learned, IEEE Software, May/June 2001.
- Defense Science Board, Report of the Defense Science Board Task Force on Defense Software, OUSD (AT&T), November 2000.
- Electronic Industries Association, Systems Engineering Capability Maturity Model (EIA-IS 731), 1998.
- D. Etter, Keynote Address: National Academy of Sciences Workshop on Statistical Methods for Defense Software, July 2001.
- D. Garlan, R. Allen, and J. Ockerbloom, Architectural Mismatch: Why Reuse Is So Hard, IEEE Software, November 1995, pp. 17-26.
- I. Jacobsen, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison Wesley, 1999.
- P. Kruchten, The Rational Unified Process, Addison-Wesley, 1999 (2nd Ed., 2001).
- J. Marenzano, System Architecture Validation Review Findings, in D. Garlan (ed.), ICSE-17 Architecture Workshop Proceedings. CMU, Pittsburgh, PA, 1995.
- G. Moore, Crossing the Chasm, Harper Business, 1991.

M. Paulk, C. Weber, B. Curtis, and M. Chrissis, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1998.

E. Rechtin and M. Maier, *The Art of Systems Architecting*, CRC Press, 1997.

W.E. Royce, *Software Project Management: A Unified Framework*, Addison-Wesley, 1998.

M. Saboe, P. Gruenbacher, and P. Kloska, *Experience with the WinWin Process for Planning Software Infrastructure and Technology Proceedings*, Intl. Conf. Group Systems, 2002.

Software Engineering Institute, *Software Capability Maturity Model Version 2.0 (draft)*, 1997a.

Software Engineering Institute, *Integrated Product Development Capability Maturity Model Version 0.98 (draft)*, 1997b.

Software Engineering Institute, *CMMI for Systems Engineering/ Software Engineering/ Integrated Product and Process Development*, CMU-SEI-TR-030,031-2000, November 2000.

J. Thorp and DMR, *The Information Paradox*, McGraw Hill, 1998.

R. Van Solingen and E. Berghout, *The Goal/Question/Metric Method*, McGraw Hill, 1999.

Figures and Tables:

Figure 1:

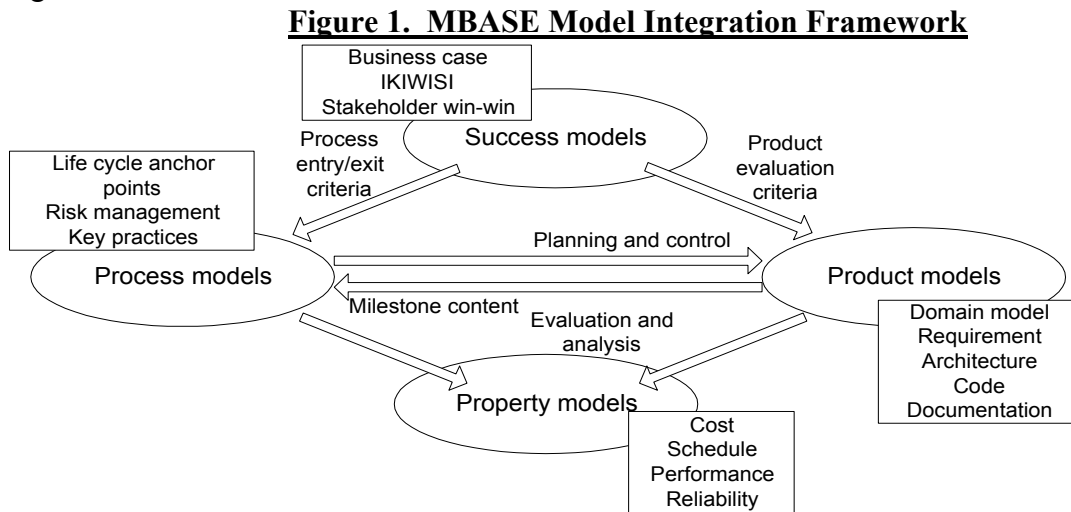


Figure 2:

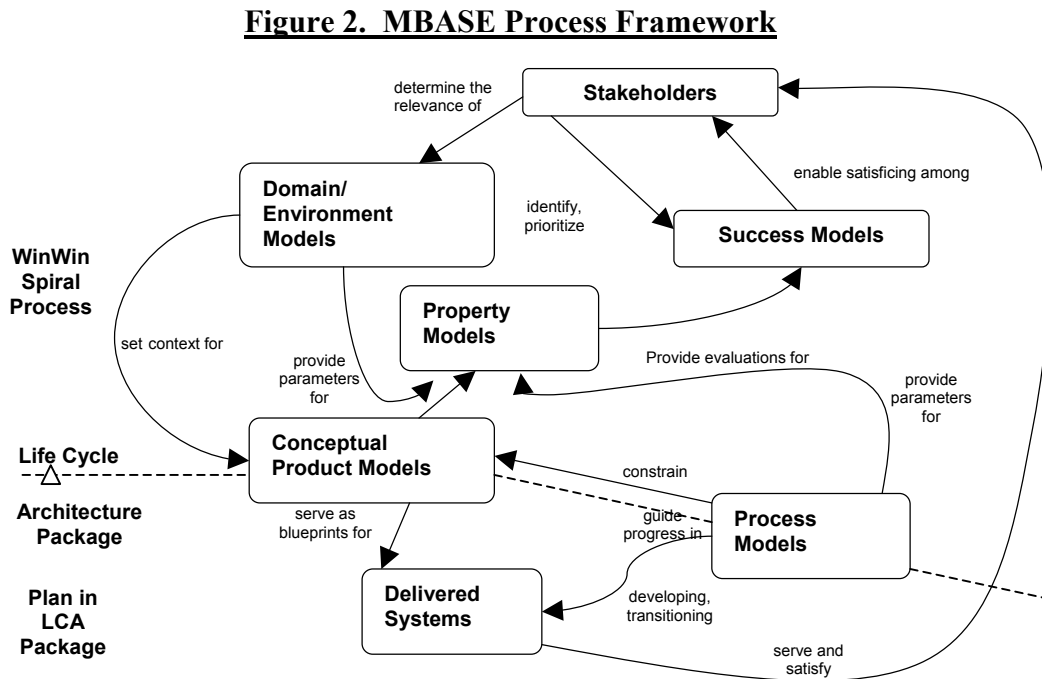


Figure 3:

Figure 3. The Win-Win Spiral Model

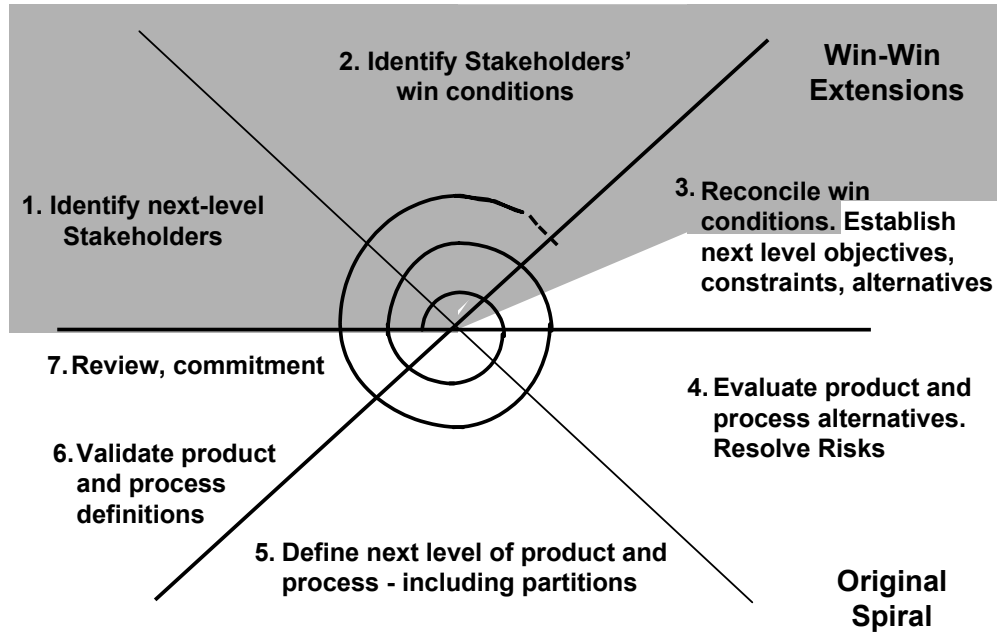


Figure 4:

Figure 4. The CeBASE Method Framework

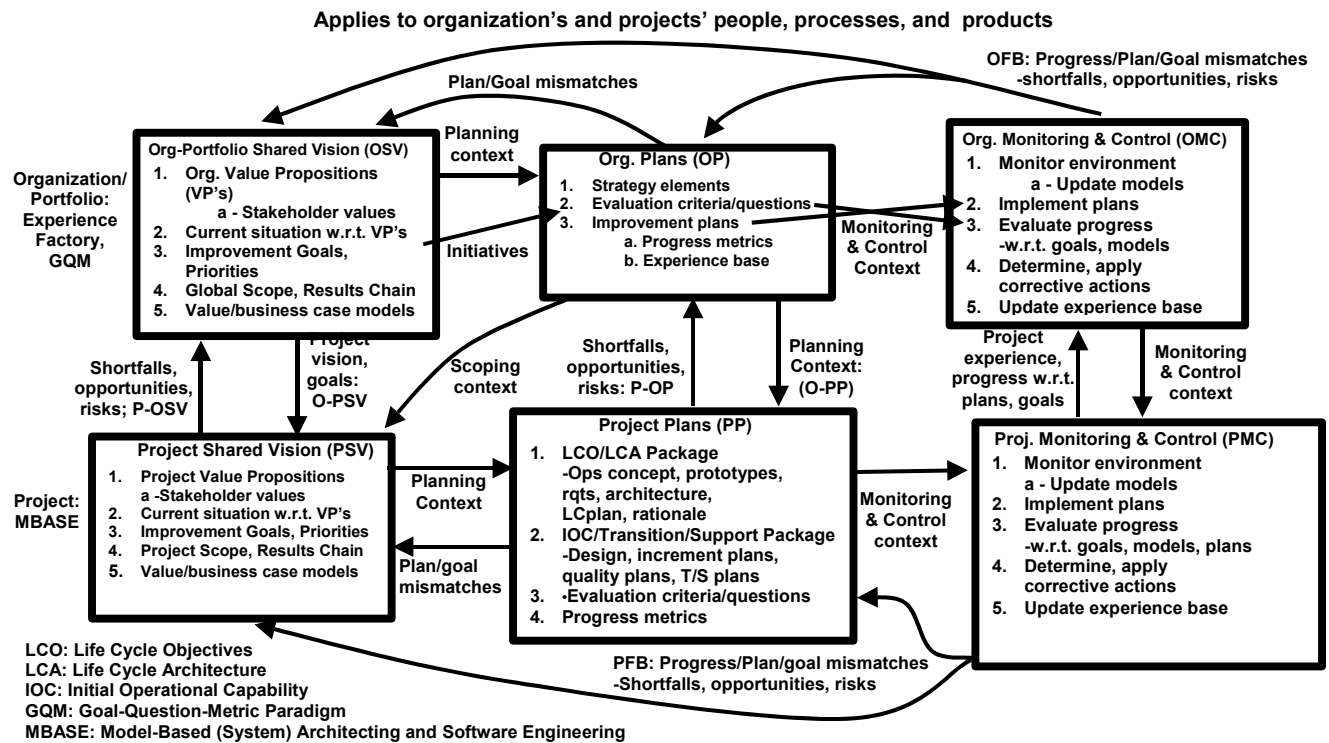


Figure 5:

Figure 5. Example CeBASE System-Level Shared Vision Content	
Table of Contents	
2.	Shared Vision
2.1	System Capability Description
2.1.1	Benefits Realized
2.1.2	Results Chain
2.2	Key Stakeholders
2.3	System Boundary and Environment
2.4	Major Project Constraints
2.5	Top-Level Business Case
2.6	Inception Phase Plan and Required Resources
2.7	Initial Spiral Objectives, Constraints, Alternatives, and Risks
2.1 System Capability Description	
A concise description of the system that can pass the “elevator test” described in Geoffrey Moore’s <u>Crossing the Chasm</u> (Harper Business, 1991, p.161). This would enable you to explain why the system should be built to an executive while riding up or down an elevator. It should take the following form:	
<ul style="list-style-type: none">• For (target customer)• Who (statement of the need or opportunity)• The (product name) is a (product category)• That (statement of key benefit-that is, compelling reason to buy)• Unlike (primary competitive alternative)• Our product (statement of primary differentiation)	
Here is an example for a corporate order-entry system: “Our sales people need a faster, more integrated order entry system to increase sales and customer satisfaction. Our proposed Web Order system would give us an e-commerce order entry system similar to Amazon.com’s that will fit the special needs of ordering mobile homes and their aftermarket components. Unlike the template-based system our main competitor bought, ours would be faster, more user friendly, and better integrated with our order fulfillment system.”	
Common Pitfalls:	
<ul style="list-style-type: none">• Not relating the need or opportunity to the goals in the organization’s Shared Vision.• Being too verbose about “Our product” or its key benefits. <p style="text-align: center;">* * *</p>	
2.2 Key Stakeholders	
Identify each stakeholder by their <u>home organization</u> , their <u>authorized representative for project activities</u> , and their <u>relation to the Results Chain</u> . The four classic stakeholders are the software/IT system’s users, customers, developers and maintainers. Additional stakeholders may be system interfacers, subcontractors, suppliers, venture capitalists, independent testers and the general public (where safety or information protection issues may be involved).	
Common Pitfalls:	
<ul style="list-style-type: none">• Being too pushy or not pushy enough in getting your immediate clients to involve the other success-critical stakeholders. Often, this involves fairly delicate negotiations among operational organizations. If things are going slowly and you are on a tight schedule, seek the help of your higher-level managers.• Accepting just anybody as an authorized stakeholder representative. You don’t want the stakeholder organization to give you somebody they feel they can live without. Some good criteria for effective stakeholders are that they be empowered, representative, knowledgeable, collaborative and committed.	

Figure 6:

Figure 6. CeBASE-CMMI Mapping: Requirements Development

Requirements Development

Produce and analyze customer, product, and product component requirements.

- CeBASE: Done via shared vision, value propositions, stakeholder negotiation. OSV-PSV, PP-1 (OCD, SSRD, FRD)

Stakeholder needs, expectations, constraints, and interfaces are collected and translated into customer requirements. SG 1

Identify and collect stakeholder needs, expectations, constraints, and interfaces for all phases of the product's life cycle.

SP 1.1-1

- CeBASE: Shared vision, current system shortfalls, proposed system. OCD 2, 3.7, 4
Elicit stakeholder needs, expectations, constraints, and interfaces for all phases of the product's life cycle. SP 1.1-2
- CeBASE: Stakeholder value propositions, negotiations, priorities; business case. OCD 2, 3.7, 4; FRD 3.3
Transform stakeholder needs, expectations, constraints, and interfaces into customer requirements. SP 1.2-1
- CeBASE: OCD-SSRD traceability and feasibility rationale. OCD- SSRD, FRD 2,3

Customer requirements are refined and elaborated to develop product and product component requirements for the product life cycle. SG 2

Establish and maintain, from the customer requirements, product and product component requirements essential to product and product component effectiveness and affordability. SP 2.1-1

- CeBASE: OCD-SSRD traceability and feasibility rationale. OCD, SSRD, FRD 2.2
Allocate the requirements for each product component. SP 2.2-1
- CeBASE: SSRD-SSAD traceability. SSRD, SSAD, FRD 2.2
Identify interface requirements. SP 2.3-1
- CeBASE: SSRD 4

The requirements are analyzed and validated, and a definition of required functionality is developed. SG 3

Establish and maintain operational concepts and scenarios. SP 3.1-1

- CeBASE: Done via OCD, prototypes. OCD 4.7, 5. Need to put Scenarios section back.
Establish and maintain a definition of required functionality. SP 3.2-1
- CeBASE: Done via OCD-SSRD traceability. OCD 4.3, SSRD 3
Analyze derived requirements to ensure that they are necessary and sufficient. SP 3.3-1
- CeBASE: Done via Feasibility Rationale. FRD 2.2, 5
Analyze requirements with the purpose of reducing the life-cycle cost, schedule and risk of product development. SP 3.4-3
- CeBASE: Done in terms of stakeholder value propositions, business case. OCD 2, FRD 2.1
Validate requirements to ensure the resulting product will perform appropriately in its intended use environment. SP 3.5-1
- CeBASE: Done via Feasibility Rationale. FRD 2.2, 5
Validate requirements to ensure the resulting product will perform as intended in the user's environment using multiple techniques as appropriate. SP 3.5-2
- CeBASE: Done via Feasibility Rationale, Verification and Validation, Test and Transition Plans. FRD 2.2, 5; LCP 4.4.2, TP, TRP (needs some elaboration).
- In CeBASE but missing here: business case; priorities; project, level of service, and evolution requirements

Figure 7:

Process Model Distinctions

Ops. Focus: <u>Assessment</u> Practice		Organizational Focus	
		Project	Organization
Application focus	Software	<u>Software CMM</u> Waterfall	<u>Software CMM</u> early EF, GQM
	Systems	<u>CMMI</u> Spiral, MBASE, RUP	<u>CMMI</u> CeBASE Method

Table 1. The MBASE/RUP Initial Anchor Point Milestones

Milestone Element	Life Cycle Objectives (LCO)	Life Cycle Architecture (LCA)
Definition of Operational Concept	<ul style="list-style-type: none"> • Top-level system objectives and scope <ul style="list-style-type: none"> - System boundary - Environment parameters and assumptions - Evolution parameters • Operational concept 	<ul style="list-style-type: none"> • Elaboration of system objectives and scope by increment • Elaboration of operational concept by increment
System Prototype(s)	<ul style="list-style-type: none"> • Exercise key usage scenarios • Resolve critical risks 	<ul style="list-style-type: none"> • Exercise range of usage scenarios • Resolve major outstanding risks
Definition of System Requirements	<ul style="list-style-type: none"> • Top-level functions, interfaces, quality attribute levels, including: <ul style="list-style-type: none"> - Growth vectors - Priorities • Stakeholders' concurrence on essentials 	<ul style="list-style-type: none"> • Elaboration of functions, interfaces, quality attributes by increment <ul style="list-style-type: none"> - Identification of TBDs (to-be-determined items) • Stakeholders' concurrence on their priority concerns
Definition of System and Software Architecture	<ul style="list-style-type: none"> • Top-level definition of at least one feasible architecture <ul style="list-style-type: none"> - Physical and logical elements and relationships - Choices of COTS and reusable software elements • Identification of infeasible architecture options 	<ul style="list-style-type: none"> • Choice of architecture and elaboration by increment <ul style="list-style-type: none"> - Physical and logical components, connectors, configurations, constraints - COTS, reuse choices - Domain-architecture and architectural style choices • Architecture evolution parameters
Definition of Life-Cycle Plan	<ul style="list-style-type: none"> • Identification of life-cycle stakeholders <ul style="list-style-type: none"> - Users, customers, developers, maintainers, interpreters, general public, others • Identification of life-cycle process model <ul style="list-style-type: none"> - Top-level stages, increments • Top-level WWWWWHH* by stage 	<ul style="list-style-type: none"> • Elaboration of WWWWWHH* for Initial Operational Capability (IOC) <ul style="list-style-type: none"> - Partial elaboration, identification of key TBDs for later increments
Feasibility Rationale	<ul style="list-style-type: none"> • Assurance of consistency among elements above <ul style="list-style-type: none"> - Via analysis, measurement, prototyping, simulation, etc. - Business case analysis for requirements, feasible architectures 	<ul style="list-style-type: none"> • Assurance of consistency among elements above • All major risks resolved or covered by risk management plan

* WWWWWHH: Why, What, When, Who, Where, How, How Much

Table 1:

Table 2:

Table 2. LCO, LCA, and IOC Pass/Fail Criteria

LCO	LCA	IOC
<p>For at least one architecture, a system built to that architecture will:</p> <ul style="list-style-type: none"> • Support the core Operational Concept • Satisfy the core Requirements • Be faithful to the Prototype(s) • Be buildable within the budgets and schedules in the plan • Show a viable business case • Have its key stakeholders committed to support the Elaboration Phase (to LCA) 	<p>For a specific detailed architecture, a system built to that architecture will:</p> <ul style="list-style-type: none"> • Support the elaborated Operational Concept • Satisfy the elaborated Requirements • Be faithful to the Prototype(s) • Be buildable within the budgets and schedules in the plan • Have all major risks resolved or covered by a risk management plan • Have its key stakeholders committed to support the full life cycle 	<p>An implemented architecture, an operational system that has:</p> <ul style="list-style-type: none"> • Realized the Operational Concept • Implemented the initial operational requirements • Prepared a system operation and support plan • Prepared the initial site(s) in which the system will be deployed for transition • Prepared the users, operators, and maintainers to assume their operational roles

Table 3:

<u>Table 3. CeBASE Method Coverage of CMMI</u>	
•	Process Management <ul style="list-style-type: none">○ Organizational Process Focus: 100+○ Organizational Process Definition: 100+○ Organizational Training: 100-○ Organizational Process Performance: 100-○ Organizational Innovation and Deployment: 100+
•	Project Management <ul style="list-style-type: none">○ Project Planning: 100○ Project Monitoring and Control: 100+○ Supplier Agreement Management: 50-○ Integrated Project Management: 100-○ Risk Management: 100○ Integrated Teaming: 100○ Quantitative Project Management: 70-
•	Engineering <ul style="list-style-type: none">○ Requirements Management: 100○ Requirements Development: 100+○ Technical Solution: 60+○ Product Integration: 70+○ Verification: 70-○ Validation: 80+
•	Support <ul style="list-style-type: none">○ Configuration Management: 70-○ Process and Product Quality Assurance: 70-○ Measurement and Analysis: 100-○ Decision Analysis and Resolution: 100-○ Organizational Environment for Integration: 80-○ Causal Analysis and Resolution: 100