An Orthogonal Framework for Improving Life Cycle Affordability

Barry Boehm*, JoAnn Lanea, Supannika Koolmanojwonga

*Center for Systems and Software Engineering, University of Southern California, Los Angeles, USA

Abstract

Many approaches to improving system affordability focus on one or two strategies (e.g., automation, outsourcing, repurposing, reuse, process maturity), and miss the opportunity for further improvements. Often, they focus on one phase (e.g., acquisition) at the expense of other factors that increase total ownership cost (TOC).

Based on several related research projects, we have developed, applied, and evolved an orthogonal framework of strategies for improving Affordability. The framework includes:

- Get the best from people (Staffing, Teambuilding, Facilities, Kaizen)
- Make tasks more efficient (Tools, Work and Oversight Streamlining, Collaboration Technology)
- Eliminate tasks (Lean and Agile Methods, Automation, Model-Based Product Generation)
- Eliminate scrap and rework (Early Risk and Defect Elimination, Evidence-Based Decision Gates, Modularity around Sources of Change, Evolutionary Development, …)
- Simplify products: KISS (Risk-Based Prototyping, Value-Based Capability Prioritization, Satisficing vs. Optimizing)
- Reuse components (Domain Engineering, Composable Components and Services, Legacy Repurposing)
- Reduce O&S costs (Automate Tasks, Design for Maintainability, Streamline Supply Chain, Anticipate/Prepare for Change)
- Perform value-based tradespace analysis among the above.

The research presented will also summarize the use of calibrated quantitative cost models for reasoning about the strategy elements and their tradeoffs.

1. Introduction and Context

The INCOSE Systems Engineering Handbook [1] defines affordability as, “The balance of system performance, cost, and schedule constraints over the system life cycle, while satisfying mission needs in concert with strategic and organizational needs.” Similar definitions are cited by the National Defense Industrial Association (NDIA) and
Military Operations Research Society (MORS) Affordability Working Groups. The most significant aspect of this definition is that it emphasizes not only cost reduction but also addition to stakeholder value.

Figure 1 illustrates this perspective on affordability. It indicates that one’s objective is to achieve combinations of added benefits and reduced costs that enable their organization to advance its current state further and further away from the origin (the organization’s original costs and benefits) toward its Pareto boundary of achievable combinations of increased benefits and reduced costs. Its primary option space will include strategies that increase benefits at no loss in cost savings, reduce costs at no loss in increased benefits, or mixed strategies in between.

Fig. 1. Affordability Improvement Options

However, there are also common special cases, such as an overall-organization 10% budget cut. Too often, such cases are handled by imposing 10% across-the-board cuts in each area; singling out easy-to-cut costs such as travel, equipment, and education; or dismissing employees with the least or most seniority. These usually have serious long-term negative effects. More far-seeing organizations have affordability engineering functions that pursue more long-term strategies that enable them to execute budget cuts in more cost-effective ways.

2. The Orthogonal Framework for Improving Affordability

In this context, Figure 2 shows the orthogonal (in terms of classes of options) framework for improving affordability. It has evolved over several decades of related industrial and academic research and development [2]. Each class has several options that have been found to be cost-effective across many application domains. For each option, an organization can assess its current state with respect to the identified improvement candidates, and can determine which candidates are the current best fit for pursuing. For several of the options, quantitative data are available for assessing the effects of improvements, as will be discussed below.

2.1. Getting the Best from People

The first major option is to get the best from people through improvements in staffing, teambuilding, facilities, and involving its people in identifying and executing Kaizen (good ideas can come from anyone, particularly performers) approaches for increasing benefits or reducing costs. Staffing improvements involve initiatives for identifying and prioritizing current and future needs for personnel knowledge, skills, and abilities; and realizing
these needs through improved staffing, retention, and career path facilitation via education, training, and mentoring. For systems engineering (SE), competency models are available for assessing and improving SE competency, such as the INCOSE-UK SE Competencies Framework 2010-0205 Issue 3 [1] and the Mitre SE Competency model [3].

Fig. 2. The Affordability and Tradespace Options Framework

The People Capability Maturity Model [1] provides a good detailed framework and set of practices for continuous improvement of an organization’s human resources. Facilities and support services for knowledge workers will include techniques to reduce distractions and increase thought flow, as described in sources such as Peopleware [5].

Incentivizing can include reducing project turnover via project completion bonuses and flowdown of project award fees to individuals. At a group level, it involves balancing individual incentives with group incentives. At a cross-organizational level involving suppliers, distributors, or strategic partners, it can involve such practices as Vested Outsourcing or shared-decision contracting [6,7], in which organizations determine each other’s value propositions or win conditions, and develop win-win arrangements in which collaborative efforts provide positive outcomes with respect to each participant’s value propositions [8,9,10]. Such practices also contribute to teambuilding; a good source for further teambuilding practices is [11].

For software projects, a quantitative framework for identifying the project’s current status and estimating potential strategies with respect to improving software productivity and affordability via people factors is shown in Figure 3. The green arrows represent COCOMO II [12] productivity ranges (the ranges in effect on project cost of having very poor or very good ratings) for staffing factors such as analyst capability, programmer capability, applications experience, platform or infrastructure (hardware, operating system, data management system, etc.) experience, and programming language and tool experience. For each of these factors, an organization can identify their projects’ average COCOMO II rating level, which identifies the project’s current location on the productivity range, and can then identify strategies for increasing the rating level and determining the resulting productivity or affordability increase associated with the increase in rating level.
Thus, for example, if an organization had High (75%) levels of Programmer Capability but Low (35%) levels of Analyst Capability, investment in systems engineering staffing, education, career pathing, and mentoring that raised the Analyst Capability level to Nominal (55%) would reduce relative effort from 1.19 to 1.00, for an affordability gain of 19%. Further initiatives to raise the level to High would reduce the relative effort to 0.85, for an affordability gain of 40%. A framework and tool for performing such assessments is the COPROMO tool described in chapter 5 of [12].

Corresponding productivity ranges for improvements in incentivizing and teambuilding are shown by the orange arrows in Figure 3, representing benefits resulting from improved personnel continuity and team cohesion. Some of the COCOMO II productivity ranges such as team cohesion have exponential effects as a function of project size. The productivity difference between a Very Low and Extra High level of team cohesion is a factor of 1.13 for a 10 thousand source lines of code (KLOC) project, and 1.46 for a 1000 KLOC project. The productivity ranges were determined by a Bayesian combination of group expert judgment and a multiple regression analysis of the contributions of each factor to the overall productivity of 161 projects in the COCOMO II database. Productivity and affordability gains due to advances in continuous process improvement (designated by the purple arrow) were determined by using the project’s Capability Maturity Model level [13,14] as a rating scale. The resulting productivity ranges are 1.20 for a 10-KLOC project and 1.71 for a 1000-KLOC project.

A similar quantitative framework for identifying the project’s current status and estimating potential strategies with respect to improving systems engineering (SE) productivity and affordability is shown in Figure 4, based on the calibrated factors in the COSYSMO cost model [15]. The people factors are shown in corresponding green, orange, and purple arrows in Figure 4. The productivity ranges are different, as COSYSMO is calibrated to the SE of hardware as well as software projects, but the can be used similarly for organizations to assess their current SE status and estimate likely SE productivity and affordability gains via improvements in staffing, teambuilding and performer-involved continuous process improvement. Of course, one must avoid SE cost reductions that reduce SE effectiveness, since those increase life-cycle scrap and rework. This topic will be addressed in Section 2.4 on Eliminating Scrap and Rework.

Fig. 3. Quantitative Software Cost Improvement Insights from COCOMO II
Having provided a thorough discussion of the first (and arguably the most important) people option, space limitations do not permit comparably detailed discussions of the remaining seven options. However, the figures provided will provide context and quantitative relationships that will strengthen the later discussions.

### 2.2. Making Tasks More Efficient

A major source of affordability improvement involves investments in improved tools to improve the cost-effectiveness of the various life-cycle sources of effort such as system scoping, definition, human interface prototyping, system architecting, development, qualification, deployment, manufacturing, operations, and life cycle support, plus general tools for program management, supply chain management, and stakeholder collaboration support. For software engineering, Figure 3 shows a productivity range of 1.50 for going from a minimal set of tools to a mature and well-integrated toolset with full life cycle coverage. A subsequent analysis confirmed that tool coverage, maturity, and integration were essential to very high tool support [16]. For SE, the corresponding productivity range for tool support in Figure 4 is 1.93.

Another source of affordability improvement is automation, in which computing and software or devices replace more expensive personnel. This has happened significantly in information processing, robots in manufacturing, and financial systems needing microsecond response times. However, there are several hazards in trusting computers to properly recognize and handle off-nominal situations, such as with several stock market crashes that caused major financial losses. And there are risks that overautomation penalizes human interaction, as with several banks which discontinued strongly automated teller support systems because they were turning the bank tellers into computer peripherals, rather than their previous roles as mini-bankers satisfying their customers.

Work and oversight streamlining generally involves undoing the effects of creeping bureaucracy, such as requiring 14 signatures to order a COTS product, or having a progress report undergo six levels of management review before it is presented to the customer. Often, performer-initiated suggestion boxes and continuous process
improvement systems will help in streamlining processes.

Collaboration technology has made significant improvements in team performance across different locations, organizations, cultures and timezones. Many project support systems are still organized around individual vs. group use, causing frequent slowdowns or incompatible decisions to be made because of insufficient information sharing or lack of wideband collaboration support across interdependent groups. The COCOMO II Multi-Site Development factor in Figure 3 has a productivity range of 1.53 for variations in collaboration support across widely distributed teams. The corresponding SE Multisite Coordination factor in Figure 4 has a productivity range of 1.93.

2.3. Eliminating Tasks

Lean and agile methods have several ways to improve affordability via eliminating tasks. Lean methods emphasize elimination of non-value-adding processes such as the creeping bureaucracy examples above, or of non-value-adding products such as unnecessary documentation. Figures 3 and 4 show the relative productivity ranges for software (1.52) and SE (1.93) effects of documentation level on cost. Often, a risk-based guideline such as “If it’s risky to leave it out, put it in; if it’s risky to put it in, leave it out,” will help make such decisions. A further approach growing in use is the workflow-oriented Kanban method of limiting work in progress and prioritizing new features described in [17].

Agile methods such as Extreme Programming have guidelines such as Simple Design, that restrict design documentation to what has already been built [18]. However, many agile projects have found that the risk of leaving out an overall architectural definition of the system to be developed is very high as projects grow larger, such as the ThoughtWorks Lease Management project described in [19].

Agile methods also include timeboxing as a way of improving affordability by eliminating tasks. This involves having the customers prioritize their desired features, with the option of dropping low-priority features if the project is running out of time or budget, or if more valuable features need to be added during development. This has the added advantage of carrying a risk reserve in terms of lower-priority features vs. unspent funds, which are easier to take away. Eliminating tasks via automation has been discussed under tools and automation in Section 2.2.

Model-based product generation has been successfully applied in the software field, particularly for domain-specific models. It has traditionally been applied to hardware elements via numerical control systems, and its recent extension to three-dimensional printing has been identified as the beginning of a third industrial revolution by such journals as The Economist [20].

2.4. Eliminating Scrap and Rework

Most analyses of scrap and rework costs find a Pareto distribution that shows 80% of the rework costs coming from 20% of the problems. Figure 5 shows such distributions resulting from analyzing the problem reports of two TRW projects. In both cases, the main cause was the lack of attention to off-nominal use cases that turned out to be architecture-breakers: network failover for Project A and extra-long messages for Project B. Actually, the root cause for both projects was an inadequate budget, schedule, and focus on thorough architecture definition and risk resolution during the preliminary design phase.

The quantitative impact of shortfalls in architecture definition and risk resolution are also shown in Figure 3 as a function of project size. A 10 KSLOC software system will have a productivity range of 1.18, as compared to a productivity range of 1.63 for a 1000 KSLOC project. This result has been used to determine an increasingly positive return on investment in systems engineering as a function of project size in [21]. Such investments include not only the definition of a system’s requirements, architecture, and plans, but also evidence that a system built to the architecture would satisfy the requirements, and be buildable within the budgets and schedules in the plans. Such evidence was not produced in Projects A and B of Figure 5.
Another valuable result of analyzing the Project A and B data was the identification of the most frequent sources of change, as these can be used to modularize the architecture around the sources of change, thus reducing the costs of change effects that would otherwise ripple through the entire product [22]. These insights led to the development of an evidence and risk-driven process model and architecture framework used in the highly successful CCPDS-R project described in Appendix D of Walker Royce’s book, *Software Project Management* [23]. It used risk-based prototyping of the user interfaces and actual development of its network operating system to provide evidence of feasibility at its PDR, which was held in month 14 rather than the usual 4 months after contract start. Its rework costs per change stayed relatively constant, compared to the usual high escalation of cost of change vs. time.

A further source of high rework costs is the premature specification of requirements before fully understanding the difficulty of achieving them. Rather than prespecify all the system’s requirements, incremental and evolutionary processes enable projects to defer commitments until their implications are better understood. As with agile methods, this approach enables projects to prioritize the content of future increments based on better-understood stakeholder value.

2.5. Simplify Products: KISS (Keep It Simple, Stakeholders)

The risk-based prototyping and value-based capability prioritization enabled CCPDS-R to simplify its products, in one case reducing its size from 12 KSLOC to 5 KSLOC without reducing essential capability. Another key approach is to use risk and evidence-based decision criteria to converge on a mutually satisfactory vs. an optimized set of performance requirements. A good example is the TRW project described in [24], whose customer specified a
A 1-second response time requirement to handle an extremely large workload. After finding that COTS products could not scale up to the workload and deliver a 1-second response time, the TRW engineers devised a complex custom architecture that would meet the 1-second response time, but that would cost $100 million. The customer’s budget was only $30 million (maybe it’s not a requirement if you can’t afford it), and it was decided to prototype some usage scenarios to determine whether a 1-second response time was needed. The result was that a 4-second response time was workable 90% of the time, enabling TRW to provide a simpler $30 million COTS-based solution.

2.6. Reuse Products

A good example of investment in product line architectures and reusable components was provided by Hewlett-Packard, which in the late 1980s found that its average market lifetime for products was 2.7 years, while developing the software for the products was taking 4 years. An example of HP’s experience in investing in a product line architecture and set of reusable components is provided in Figure 6. It shows that the first two 1987 projects required roughly 5 years to create the architecture and components, but that by 1992 their projects were finishing in roughly 1 year each [25].

Fig. 6. Hewlett-Packard Experience in Product Line Reuse

Use of COTS products vs. custom development was covered in the TRW example in Section 2.5. Another increasingly attractive option for reuse is the reuse of full legacy systems by repurposing them for new missions. Good examples are the DoD B-52 and C-130 aircraft, which have been repurposed to support wide ranges of new missions over time periods of over 50 years.

2.7. Reducing Operations and Support Costs

Many projects engage in short-term thinking to minimize acquisition costs, or (as often with agile methods) to rapidly produce early capabilities, but thereby end up with brittle or unscalable architectures that significantly increase life cycle costs. Table 1 summarizes data from [26] and [27] on the percentage of life cycle costs expended on operations and support. It shows the general dominance of post-deployment costs, and the need for better preparation for and execution of life-cycle support.
As discussed in Section 2.2, automating operational tasks can both decrease operational costs and increase operational effectiveness. However, there may be risks in taking humans out of the decision loop. Similarly, design for maintainability can improve both life cycle cost and effectiveness, as discussed in Section 2.4 on the Parnas principle, and for hardware in sources such as [28]. Streamlining supply chains for such approaches as just-in-time manufacturing can again improve both life cycle cost and effectiveness, as discussed in [29]. And given the increasing pace of change in competition, technology, marketplaces, and organizations, pro-active investments in anticipating and preparing for change are increasingly valuable for both reducing costs and increasing benefits.

### 2.8. Value and Architecture-Based Tradeoffs and Balancing

As shown in Figure 1, affordability can involve numerous combinations of options for decreasing life cycle costs and increasing life cycle effectiveness. Evaluating these combined options often involves complex tradeoffs among affordability and other –ilities such as reliability, availability, maintainability, usability, adaptability, interoperability, scalability, and others such as safety, security, reliance, and timeliness. For software systems, a considerable literature on ity tradeoffs has evolved, from [30] through [31,32], to the [33] Architecture Tradeoff Analysis Method. Current systems engineering approaches include real options analysis [34, 35], total cost of ownership analysis [36], incremental-commitment decision-space narrowing, [37], and physical tradespace analysis [38]. All of the approaches face significant challenges in multi-criteria decision analysis, but the need for improved capabilities continues to increase.

### 3. Conclusions

Affordability is not only about cost reduction, but also about preserving or enhancing both near-term and long-term benefits. Many cost reduction efforts focus only on a few easy ways to eliminate costs, and not only unnecessarily eliminate benefits, but also increase long-term costs. More far-seeing organizations have affordability engineering functions that pursue more long-term strategies that enable them to execute budget cuts in more cost-effective ways.

The orthogonal Life Cycle Affordability Framework, evolved over several decades of related industrial and academic research and development, provides a wider set of options for addressing both near-term and long-term costs and benefits. Cost models such as COCOMO II, COSYSMO, and hardware cost models can enable organizations to determine their current status with respect to the cost driver ratings, and to determine the potential cost savings achievable by improving their ratings. Challenges for the future primarily include the development of similar methods, models, processes and tools for estimating benefits, and for evaluating tradeoffs among affordability and other desired system –ilities.

### Acknowledgements

This material is based upon work supported, in whole or in part, by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract H98230-08-D-0171. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.
References

4. B. Curtis et al., The People Capability Maturity Model, Addison Wesley, 2002.