The ROI of Systems Engineering: Some Quantitative Results

Barry Boehm
Center for Systems and Software Engineering University of Southern California
boehm@usc.edu

Ricardo Valerdi
Lean Aerospace Initiative, Massachusetts Institute of Technology
rvalerdi@mit.edu

Abstract

This paper presents quantitative results on the return on investment of systems engineering (SE-ROI) from an analysis of the 161 software projects in the COCOMO II database. The analysis shows that, after normalizing for the effects of other cost drivers, the cost difference between projects doing a minimal job of software systems engineering – as measured by the thoroughness of its architecture definition and risk resolution – and projects doing a very thorough job was 18% for small projects and 92% for very large software projects as measured in lines of code.

1. Motivation and Context

1.1. The Need for a Business Case for Systems Engineering Investments

How much systems engineering is enough? Some decision makers draw on analogies such as, "We pay an architect 10% of the cost of a building, so that's what we'll pay for systems engineering." But is 10% way too little, or way too much? Many cost-cutting decision makers see systems engineering as an activity that doesn't directly produce the product, and as a result try to minimize its cost. But this often leads to an increased amount of late rework and embarrassing overruns.

The International Council on Systems Engineering (INCOSE) has made the determination of the return on investment in systems engineering a high-priority research topic in its Vision 2020 document [1]. A partial answer to this question in the domain of software development is provided below.

1.2. Analysis of Contributing Factors to Software Development Productivity

Most of the quantitative analyses done to date on SE-ROI have shown statistical correlations between the percentage of system development cost and development time devoted to systems engineering and the percentage of additional cost and time that was needed to produce a satisfactory system. This is not a direct measure of business value or mission effectiveness, but it is a good proxy.

In general, though, the data available for these analyses has not included data that could help determine how much of the correlation is due to systems engineering effectiveness or to other factors such as requirements volatility, contractual budget and schedule stretchouts, domain experience, or personnel capability.

The 161 software projects in the COCOMO II database collected over a 25-year period contain data on these attributes as part of each project's report on 23 size, product, process, project, and personnel factors. Its attribute for systems engineering effectiveness is the degree of thoroughness of the project's architecture definition and risk resolution by its Preliminary Design Review or equivalent, based on seven factors discussed below.

Emerging models for estimating systems engineering cost and time such as COSYSMO [16] have databases including many of these attributes, but they are limited to addressing the cost aspect of ROI since they only estimate system engineering costs and not their effects on development. The cost and schedule data in the COCOMO II database includes both software systems engineering and software development effort; allowing for analysis of their corresponding effect on cost.

2. Foundations of the COCOMO II Architecture and Risk Resolution (RESL) Factor

2.1. Experiential Origins of the RESL Factor

The original Constructive Cost Model (COCOMO) for software cost and time estimation [2] did not include a factor for systems engineering thoroughness, or any factors reflecting management control over a project's diseconomies of scale. The closest factor to systems engineering thoroughness was called Modern Programming Practices, which included such practices as top-down development, structured programming, and design and code inspections. Diseconomies of scale were assumed to be built into a project's development mode: a low-criticality project had an exponent of 1.05 relating software project size to project development effort. This meant that doubling the product size increased effort by a factor of 2.07. A mission-critical project had an exponent of 1.20, which meant that doubling product size increased effort by a factor of 2.30.

Subsequent experience and analyses at TRW during the 1980's indicated that some sources of software development diseconomies of scale were management controllables, and that thoroughness of systems engineering was one of the most significant sources. For example, some large TRW software projects that did insufficient software architecture and risk resolution had very high rework costs [4], while similar smaller projects had smaller rework costs.

2.1.1. Reducing software rework via architecture and risk resolution: Analysis of project defect tracking cost-to-fix data (a major source of rework costs) showed that 20% of the defects accounted for 80% of the rework costs, and that these 20% were primarily due to inadequate architecture definition and risk resolution.

For example, in TRW Project A in Figure 1, most of the rework was the result of development of the network operating system to a nominal-case architecture, and finding that the systems engineering of the architecture neglected to address the risk that the operating system architecture would not support the project requirements of successful system fail-over if one or more of the processors in the network failed to function. Once this was discovered during system test, it turned out to be an "architecture-breaker" causing several sources of expensive rework to the already-developed software. A similar "architecture-breaker," the requirement to handle extra-long messages (over 1 million characters), was the cause of most of the rework in Project B, whose original nominal-case architecture assumed that almost all messages would be short and easy to handle with a fully packet-switched network architecture.

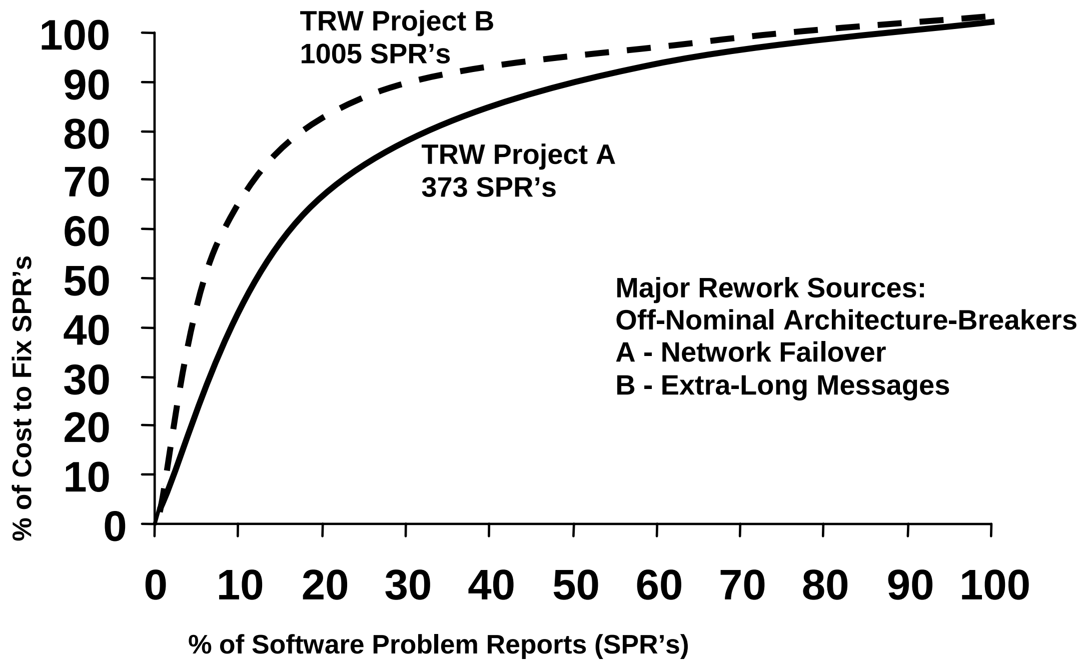

Figure 1. Most Rework Comes from Unaddressed Architecture Risks

The results of these and related analyses at TRW and elsewhere showing the frequently steep growth of the cost-to-fix curve vs. system phase in Figure 2, caused TRW to develop policies requiring thorough risk analyses of all requirements by the project's Preliminary Design Review (PDR). With TRW's adoption of the Ada programming language and associated ability to verify the consistency of Ada module specifications, the risk policy was extended into an Ada Process Model for software, also requiring that the software architecture pass an Ada compiler module consistency check prior to PDR [15].

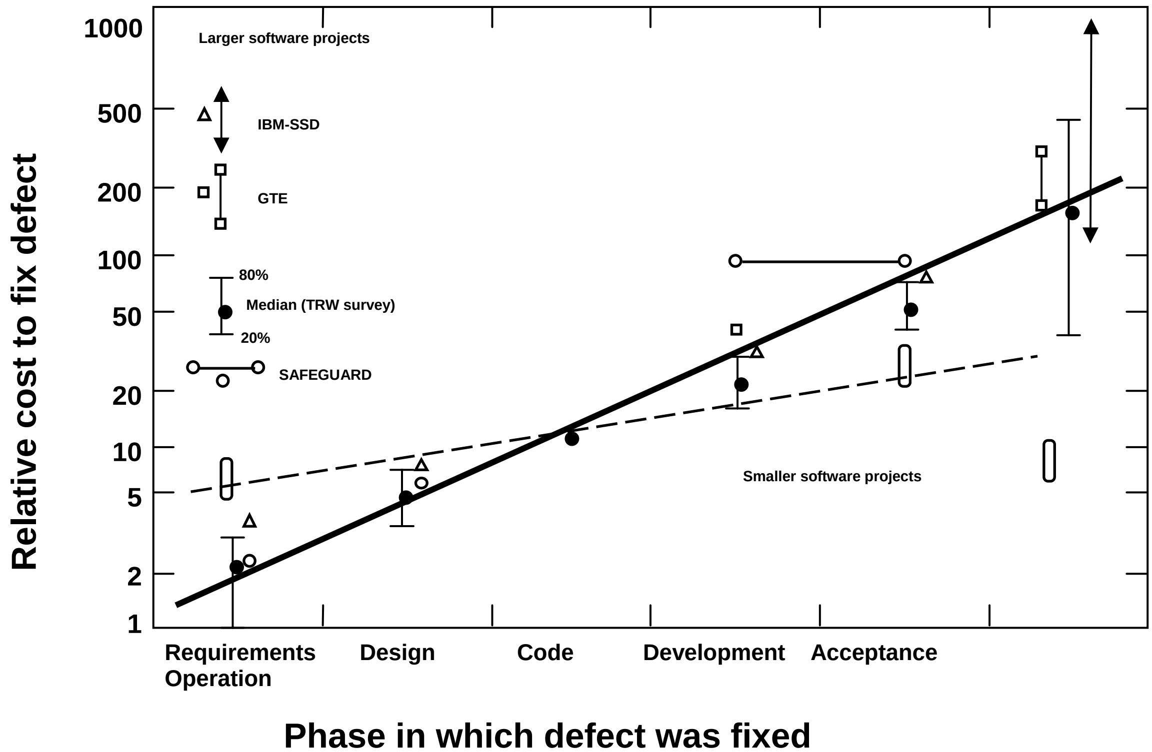

Figure 2. Risk of Delaying Risk Management

2.2. The RESL Factor in Ada COCOMO and COCOMO II

In 1987-1989, TRW developed a version of COCOMO for large mission-critical projects using the Ada Process model, called Ada COCOMO [9]. It reduced the 1.20 exponent relating product size to project effort as a function of the degree that the project could follow the Ada Process model. Performing early risk analysis and resolution was difficult to do on some projects required by government standards and contracts to use sequential waterfall-model processes. Ada COCOMO showed that reduction of software project diseconomies of scale via architecture and risk resolution operates as a management controllable factor, and helped the field evolve toward more risk-driven concurrently engineered processes vs. documentation-driven processes.

2.2.1. Resulting Risk-Driven Concurrent Engineering Software Process Models: The Ada Process Model and the CCPDS-R project [15] showed that it was possible to reinterpret sequential waterfall process model phases, milestones, and reviews to enable projects to perform risk-driven concurrent engineering of their requirements, architecture, and plans, and to apply review criteria focusing on the compatibility and feasibility of these artifacts.

Subsequently, these practices were elaborated into general software engineering – and systems engineering for software-intensive systems – process models emphasizing risk-driven concurrent engineering and associated milestone review pass-fail criteria. These included the Rational Unified Process [15, 12, 10, 11], and the USC Model-Based (System) Architecting and Software Engineering (MBASE) model [8, 7], which integrated the risk-driven concurrent engineering spiral model [6] with the Rechtin concurrent engineering Systems Architecting approach [13, 14]. Both RUP and

MBASE used a set of anchor point milestones, including the Life Cycle Objectives (LCO) and Life Cycle Architecture (LCA) as their model phase gates. Actually, these were determined in a series of workshops involving the USC Center for Software Engineering and its 30 government and industry affiliates, including Rational, Inc., as phase boundaries for COCOMO II cost and schedule estimates [3].

2.2.2. The RESL Factor[1] in COCOMO II: The definition of the COCOMO II software cost estimation model [5] was evolved during 1995-1997 by USC and its 30 industry and government affiliates. Its diseconomy-of-scale factor is a function of RESL and four other scale factors, two of which are also management controllables: Capability Maturity Model maturity level and developer-customer-user team cohesion. The definition of the RESL rating scale was elaborated into the seven contributing factors shown in Table 1.

Table 1. RESL Rating Scale

Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Risk Management Plan identifies all critical risk items, establishes milestones for resolving them by PDR or LCA.	None	Little	Some	Generally	Mostly	Fully
Schedule, budget, and internal milestones through PDR or LCA compatible with Risk Management Plan.	None	Little	Some	Generally	Mostly	Fully
Percent of development schedule devoted to establishing architecture, given general product objectives.	5	10	17	25	33	40
Percent of required top software architects available to project.	20	40	60	80	100	120
Tool support available for resolving risk items, developing and verifying architectural specs.	None	Little	Some	Good	Strong	Full
Level of uncertainty in key architecture drivers: mission, user interface, COTS, hardware, technology, performance.	Extreme	Significant	Considerable	Some	Little	Very Little
Number and criticality of risk items.	> 10 Critical	5-10 Critical	2-4 Critical	1 Critical	> 5Non-Critical	< 5 Non-Critical

[1] RESL is one of five factors that are used to represent the diseconomies of scale in COCOMO II. The other four are Precedentedness, Development Flexibility, Team Cohesion, and Process Maturity. See:
[5] Boehm, B., Abts, C., Brown, A. W., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, D., and Steece, B., *Software Cost Estimation With COCOMO II*: Prentice Hall, 2000.

The values of the rating scale for the third characteristic, percent of development schedule devoted to establishing architecture, were obtained through a behavioral assessment of the range of possible values that systems engineers might face. The minimum expected level of effort spent on architecting was assumed to be 5%, or 1/20, of the total project effort. To operationalize the remaining rating levels, a similar logic was applied. It was assumed that the subsequent rating levels were 10% (1/10), 17% (1/6), 25% (1/4), or 33% (1/3) of the project effort. In the best case, 40% or more effort would be invested in architecting.

Each project contributing data to the COCOMO II database used Table 1 as a guide for rating its RESL factor. The ratings for each row could have equal or unequal weights as discussed between data contributors and COCOMO II researchers in data collection sessions. The distribution of RESL factor ratings is roughly a normal distribution, as shown in Figure 3.

Figure 3. RESL Ratings for 161 Projects in the COCOMO Database

The contribution of a project's RESL rating to its diseconomy of scale factor was determined by a Bayesian combination of expert judgment and a multiple regression analysis of the 161 representative software development projects in the COCOMO II database. These include commercial information technology applications, electronic services, telecommunications, middleware, engineering and science, command and control, and real time process control software projects. Their sizes range from 2.6 thousand software lines of code (KSLOC) to 1,300 (KSLOC), with 13 projects below 10 KSLOC and 5 projects above 1,000 KSLOC.

2.2.3. RESL Calibration Results: Calibrating the RESL scale factor was a test of the hypothesis that proceeding into software development with inadequate architecture and risk resolution results (i.e., inadequate systems engineering results) would cause project effort to increase due to the software rework necessary to overcome the architecture deficiencies and to resolve the risks late in the development cycle – and that the rework cost increase percentage would be larger for larger projects.

The regression analysis to calibrate the RESL factor and the other 22 COCOMO II cost drivers confirmed this hypothesis with a statistically significant result. The calibration results determined that for this sample of 161 projects, the difference between a Very Low RESL rating and an Extra High rating was an extra contribution of 0.0707 added to the exponent relating project effort to product size. This translates to an extra 18% effort for a small 10 KSLOC project, and an extra 92% effort for an extra-large 10,000 KSLOC project.

Figure 4 summarizes the results of the analysis. It shows that at least for this sample of 161 software projects, the difference between a project doing a minimal job of systems engineering – as measured by its degree of architecture and risk resolution – is an increasingly large increase in overall project effort and cost, independent of the effects of the other 22 COCOMO II cost drivers. This independence is because the regression analysis also accounts for variations in effort due to the other 22 factors in its statistical results. None of these factors are highly correlated with the RESL factor; the highest correlation coefficient is 41%.

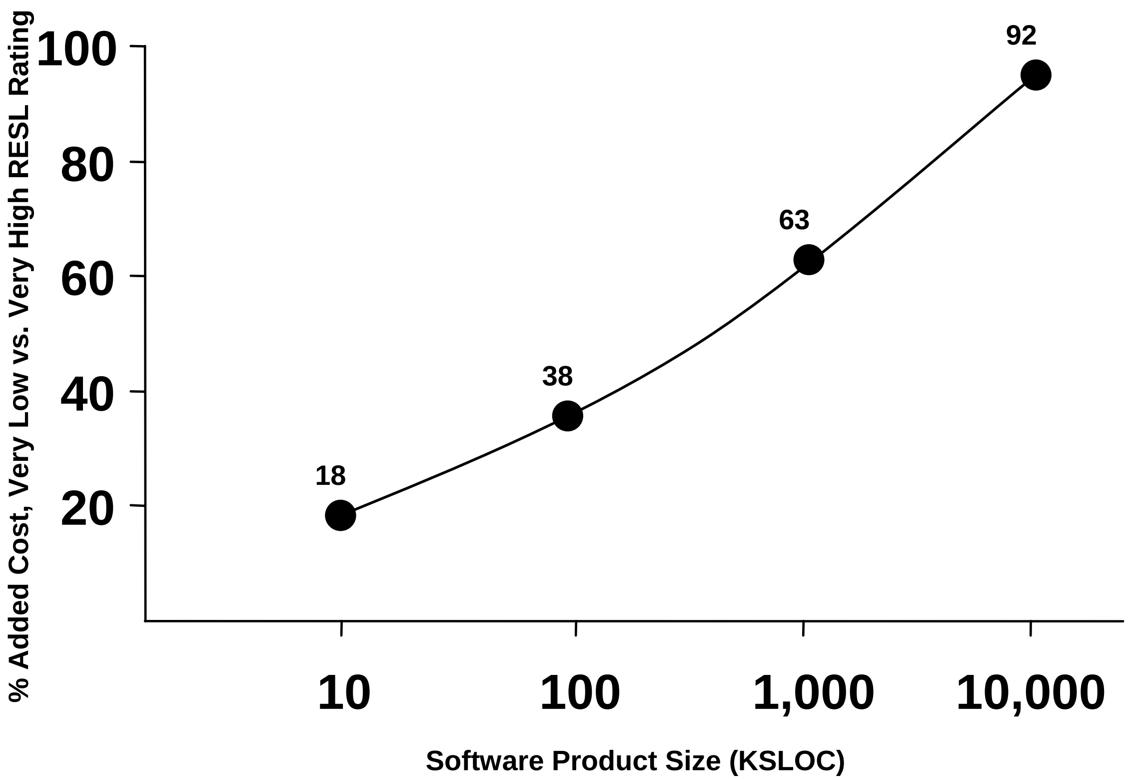

Figure 4. Added Cost of Minimal Software Systems Engineering

3. Resulting ROI for Software Systems Engineering Improvement Investments

Investing in improved software systems engineering involves a higher and stronger level and focus of effort on risk-driven concurrent engineering of software system requirements, architecture, plans, budgets, and schedules. It also requires assurance of their consistency and feasibility via prototyping, modeling, analysis, and success-critical stakeholder review and commitment to support the next phase of project activity, as discussed at the end of section 2.1.

The results of the COCOMO II calibration of the RESL factor shown in Figure 4 enable us to determine the ROI for such investments, in terms of the added effort required for architecture and risk resolution, and the resulting savings for various sizes of software systems measured in KSLOC. A summary of these results is provided in Table 2 for a range of software system sizes from 10 to 10,000 KSLOC.

Table 2. Software Systems Engineering/RESL Return on Investment

COCOMO II RESL Rating	Very Low	Low	Nominal	High	Very High	Extra High
RESL Time Invested %	5	10	17	25	33	>40 (50)
Level of Effort	0.3	0.4	0.5	0.6	0.7	0.75
RESL Investment Cost %	1.5	4	8.5	15	23	37.5
Incremental Investment		2.5	4.5	6.5	8	14.5
Scale Factor Exponent for Rework Effort	1.0707	1.0565	1.0424	1.0283	1.0141	1.0
<u>10 KSLOC Project</u>						
Added Effort %	17.7	13.9	10.3	6.7	3.3	0
Incremental Benefit, Cost		3.8, 2.5	3.6, 4.5	3.6, 6.5	3.4, 8	3.3, 14.5
Incremental $ROI = \frac{(B-C)}{C}$		0.52	-0.20	-0.45	-0.58	-0.77
<u>100 KSLOC Project</u>						
Added Effort %	38.4	29.7	21.6	13.9	6.7	0
Incremental Benefit, Cost		8.7, 2.5	8.1, 4.5	7.7, 6.5	7.2, 8	6.7, 14.5
Incremental $ROI = \frac{(B-C)}{C}$		2.48	0.80	0.18	-0.10	-0.54
<u>1,000 KSLOC Project</u>						
Added Effort %	63	47.7	34.0	21.6	10.2	0
Incremental Benefit, Cost		15.3, 2.5	13.7, 4.5	12.4, 6.5	11.4, 8	10.2, 14.5
Incremental $ROI = \frac{(B-C)}{C}$		5.12	2.04	0.91	0.42	-0.30
<u>10,000 KSLOC Project</u>						
Added Effort %	91.8	68.3	47.8	29.8	13.9	0
Incremental Benefit, Cost		23.5, 2.5	20.5, 4.5	18.0, 6.5	15.9, 8	13.9, 14.5
Incremental $ROI = \frac{(B-C)}{C}$		8.40	3.56	1.77	0.99	-0.04

The percentage of time invested on architecting is provided for each RESL rating level together with:

- Level of effort. The numbers reflect the fraction of the average project staff level on the job doing systems engineering if the project focuses on systems engineering before proceeding into development for 5%, 10%, 17%, etc. of its planned schedule; it looks roughly like a Rayleigh curve observed in software projects [2].
- RESL investment Cost %. The percent of proposed budget allocated to architecture and risk resolution. This is calculated by multiplying the RESL

percentage calendar time invested by the fraction of the average level of project staffing incurred for each rating level. For example, the RESL investment cost for the Very Low case is calculated as: 5*0.3 = 1.5.

- Incremental investment. The difference between the RESL investment cost % of the n^{th} rating level minus the n^{th}-1 level. The incremental investment to go from a Very Low to a Low RESL rating is calculated as: 4-1.5 = 2.5%
- Scale factor exponent for rework effort. The exponential effect of the RESL driver on software project effort as calibrated from 161 projects.

Return on Investment values are calculated for five different rating scale levels across four different size systems through the calculation of:

- Added effort. Calculated by applying the scale factor exponent for rework (i.e., 1.0707) to the size of the system (i.e., 10 KSLOC) and calculating the added effort introduced. For the 10 KSLOC project, the added effort for the Very Low case is calculated as follows:

$$\text{Added effort \%} \quad = \frac{10^{1.0707} - 10}{10} * 100$$
$$= 17.7$$

- Incremental benefit. The difference between the added effort for the n^{th} case and the n^{th}-1 case. The incremental benefit to go from a Very Low to a Low RESL rating is calculated as: 17.7-13.9 = 3.8.
- Incremental cost. Same as the value for incremental investment.
- Incremental ROI. Calculated as difference between the benefit and the cost divided by the cost. For the 10 KSLOC project, the incremental ROI for the Low case is calculated as follows:

$$\text{ROI} \quad = \frac{(3.8 - 2.5)}{2.5}$$
$$= 0.52$$

It is evident from each ROI row in Table 2 that architecting has a decreasing amount of ROI as later investments are made. However, Table 2 also indicates that larger projects enjoy higher levels of ROI, and a later “sweet spot” at which the ROI from added architecting investment goes from positive to negative.

4. Conclusions

Evidence has been provided for the return on investment for systems engineering in the context of software systems. While the numbers may be different for non-software systems, there is suggestive evidence that larger systems enjoy larger systems engineering ROI values compared to smaller systems and that the most cost-effective amount of systems engineering has an inherent sweet spot based on the size of the system.

These results strengthen the argument for the value of systems engineering by providing quantitative evidence that doing a minimal job of software systems engineering significantly reduces project productivity. Even higher ROIs would result from including the operational cost penalties in business or mission cost, schedule, and performance that could surface as a result of inadequate systems architecting and risk resolution.

5. References

[1] Blanchard, B., Honour, E., Lake, J., Jain, R., Martin, J., Schier, J., Sage, A., and Weiss, S., "Systems Engineering Vision 2020 - Ver 1.5," 2005.

[2] Boehm, B., *Software Engineering Economics*: Prentice-Hall, 1981.

[3] Boehm, B., "Anchoring the Software Process," *Software*, pp. 73–82, 1996.

[4] Boehm, B., "Unifying Software Engineering and Systems Engineering," *Computer*, pp. pp. 114-116, 2000.

[5] Boehm, B., Abts, C., Brown, A. W., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, D., and Steece, B., *Software Cost Estimation With COCOMO II*: Prentice Hall, 2000.

[6] Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., and Madachy, R., "Using the WinWin Spiral Model: A Case Study," *IEEE Computer*, pp. 33–44, 1998.

[7] Boehm, B. and Port, D., "Escaping the Software Tar Pit: Model Clashes and How to Avoid Them," *ACM Software Engineering Notes*, 1999.

[8] Boehm, B. and Port, D., "Balancing Discipline and Flexibility With the Spiral Model and MBASE," *CrossTalk*, 2001.

[9] Boehm, B. and Royce, W., "Ada COCOMO and the Ada Process Model," presented at Proceedings of 5th COCOMO User's Group Meeting, Software Engineering Institute, Pittsburgh, PA, 1989.

[10] Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language: User Guide*. Reading, MA: Addison-Wesley, 1999.

[11] Jacobson, I., Booch, G., and Rumbaugh, J., *The Unified Software Development Process*. Reading, MA: Addison-Wesley, 1999.

[12] Kruchten, P., *The Rational Unified Process: An Introduction*, 2nd ed. Reading, Massachusetts: Addison-Wesley, 2000.

[13] Rechtin, E., *Systems Architecting*. Englewood Cliffs, NJ: Prentice Hall, 1991.

[14] Rechtin, E. and Maier, M., *The Art of Systems Architecting*. Boca Raton, FL: CRC Press, 1997.

[15] Royce, W., *Software Project Management: A Unified Framework*: Addison Wesley, 1998.

[16] Valerdi, R., "The Constructive Systems Engineering Cost Model (COSYSMO)," PhD Dissertation, University of Southern California, 2005.