Experiences in Software Cost Modeling  
Barry Boehm, USC  
January 1997


My first exposure to software economics came on my first day in the software business, in June 1955 at General Dynamics in San Diego. My supervisor took me on a walking tour through the computer, an ERA 1103 which occupied most of a large room. His most memorable comment was,

"Now listen. We're paying this computer six hundred dollars an hour, and we're paying you two hundred dollars an hour, and I want you to act accordingly."

This created some good habits for me, such as careful desk checking, test planning, and analyzing before coding. But it also created some bad habits—a preoccupation with saving microseconds, patching object code, etc.—which were hard to unlearn when the balance of hardware and software costs began to tip the other way.

I joined the Rand Corporation in 1959, and confirmed this changing balance of hardware and software costs in 1971, when Rand lent me to the Air Force to run a study called CCIP-85. This study involved a forecast of Air Force command and control information processing needs into the mid-1980's, an identification of the most critical information processing deficiencies, and a recommended Air Force R&D program to address the deficiencies.

The study's sponsors and I expected that the main deficiencies would be computer speed and memory, large-screen displays, etc. But what we found in visiting Air Force command and control sites was that their biggest operational deficiencies came from shortfalls in the software. And we found that the balance of Air Force hardware-software costs was becoming dominated by software costs.

This led me to investigate what it was about software production that caused it to cost so much. The main results were presented in 1973 Datamation article, "Software and Its impact: A Quantitative Assessment." Most of the costs did not involve coding, but analysis and design, testing, and documentation. And much of the work involved repetitive manual tasks which could be largely eliminated by automation. But there was a lot about understanding and reducing software costs that could not be done from a think-tank such as Rand, so in 1973 I decided to join TRW as their Director of Software Research and Technology.
The first TRW data analysis I was involved in was the "Characteristics of Software Quality" study for NBS in 1973. Its most significant conclusion for TRW was that the bulk of the serious software defects were inserted in the requirements and design phases, but that they weren't being eliminated until testing, when they were much more expensive to fix. This caused us to focus TRW's software technology investments on the front end of the software life cycle, leading to such tools as the Design Assertion Consistency Checker and the Software Requirements Engineering Methodology, now commercialized as RDD-100.

At the time, TRW had a software cost estimation model developed by Ray Wolverton in 1972. It was a good match to TRW's needs at the time, but as TRW's software business expanded (eventually becoming #2 in the world to IBM in the Datamation 100 list), top management felt the need for a more general model, and this became my top priority goal in 1976.

In surveying existing software cost estimation models at the time, I found some very good work being done by Capers Jones, Al Pietrasanta, Joel Aron, and others at IBM; by Larry Putnam in U.S. Army applications; by Frank Freiman and Bob Park with RCA PRICE S; by Vic La Bolle and others at SDC; by Rachel Black and others at Boeing; and by Doty Associates. In comparing these with TRW project experience, and a few early TRW project data points, it appeared that the functional form of the Doty model (cost as an exponential function of size, modified by a set of multiplicative cost drivers) appeared to be the best match, but that it had several problems with stability, definition, and choice of cost drivers.

This led to a group effort at TRW involving ten experienced software managers and analysts determining the most appropriate initial set of cost drivers, rating scales, and multiplier values for a provisional cost model. This was then calibrated to 20 TRW project data points to produce a model called SCEP (Software Cost Estimating Program). SCEP worked well enough for TRW that it was established as the company standard for use on all proposals and as the basis for software data collection. Its continued use led to a number of upgrades to add new cost drivers, phase and activity distributions, reuse and maintenance effects, etc. It also served as the quantitative basis for TRW's investment in productivity-enhancing tools, processes, personnel, and management practices in the 1980's such as the TRW Software Productivity System, rapid prototyping tools, and the Spiral Model.

Concurrently, I was gathering non-TRW data through student term projects in my USC and UCLA courses, and through open sources such as the NASA Software Engineering Lab via Vic Basili. I tried fitting this data to the TRW SCEP model, but many of the data points did not fit well. In analyzing the patterns among the poorly-fitting projects, I found that the differences could be explained fairly well by the concept of a "development mode." TRW's mode tended to be what became called the Embedded mode, characterized by tight, ambitious requirements and unprecedented applications. Batch, precededented business and scientific applications with easy-to-relax requirements and schedules fell into a different mode that became called the Organic mode. By fitting different size-to-cost coefficients and scaling exponents to these modes (and an intermediate mode called Semidetached), a three-mode model was able to explain the data fairly well (within 20% of the actuals, 70% of the time).
The resulting model was calibrated to 56 project data points in 1978, and produced comparably accurate estimates for another 7 project data points collected in 1979. It became the Constructive Cost Model (COCOMO) published in the book Software Engineering Economics in 1981.

Since then over a dozen commercial versions of COCOMO and hundreds of in-house versions of COCOMO have been developed. A number of people have developed significant extensions to cover staffing constraints (Paul Rook and GECOMO), calibration capabilities (Dan Ligett and COSTAR, Bernie Roush and COSTMODL), labor grade distributions (Walker Royce and PCOC), function point sizing (Bob Gordon and BYL), customer costs (Bernie Price, Wolf Goethert and SECOMO), expert system diagnostics and risk advisors (Virginia Day and ESCOMO, Ray Madachy and Expert COCOMO), and numerous tailored versions for defense acquisition (Ray Kile and REVIC) and foreign countries (Ali Mili and TUCOMO for Tunisia, Yukio Miyazaki and Kuniaki Mori for Fujitsu, and others). Aspects of COCOMO percolated into new models such as SEER (Randy Jensen and Dan Galorath) and Softcost (Don Reifer). 

In 1987-89, Walker Royce and I developed Ada COCOMO, with some revised features and multipliers to cover the effects of using Ada and associated process improvements such as the Ada Process Model, an extension of the risk-driven Spiral Model. This model capitalized on the ability to create compiler-checkable Ada package specifications to perform much of software integration prior to developing the code. This and other early architecting and risk-resolution activities shifted more of the cost to the front end, and significantly reduced overall costs. Ada COCOMO also incorporated models for costing incremental development and developing software for subsequent reuse.

In 1989, I early-retired from TRW and worked in Washington DC managing the DARPA Information Science and Technology Office and the DDR&E Software and Computer Technology Office. This enabled me to sponsor such efforts as the SEI Core Metrics program under Anita Carleton, Bob Park, Wolf Goethert, and others, and the Amadeus metrics tool under Rick Selby, but it kept me away from the software cost estimation field until my return to California and the TRW Professorship of Software Engineering at USC in 1992.

By this time, it was clear that many of the assumptions underlying the original COCOMO model were not well matched to the new ways that software was being developed. Graphic user interface (GUI) builders made sizing parameters such as delivered source instructions irrelevant for the GUI portions of the software. Non-sequential process models obscured the definition of a projects' beginning and end points, confounding both estimation and data collection. Simple linear models for software adaptation, reuse, and COTS integration did not fit experiential data. Widespread interactive development made such cost drivers as turnaround time irrelevant. Considerable controversy existed over the cost implications of investments in SEI-CMM process maturity.

To address the situation, one of our main projects at the USC Center for Software Engineering is the COCOMO 2.0 project. Its goals are to provide a new version of COCOMO better turned to current and likely future software practices, while preserving the open nature of COCOMO's internal algorithms and
external interfaces, and preserving those parts of COCOMO which continue to be relevant. The COCOMO 2.0 research group includes Prof. Ellis Horowitz, Dr. Ray Madachy, Chris Abts, Brad Clark, Sunita Devnani-Chulani, and Prof. Rick Selby at UC Irvine. It also includes the 27 industry and government COCOMO 2.0 Affiliate organizations listed in the Acknowledgments, who provide sponsorship, expertise, and data for calibrating the model.

Our first step in scoping COCOMO 2.0 was to perform a comparative analysis of the approaches taken by the leading commercial software cost models. The model proprietors were very helpful in providing information, particularly Dan Galorath for SEER, Capers Jones for CHECKPOINT, Don Reifer and Tony Collins for Softcost, and Howard Rubin for Estimacs. We also performed a forecast of future software development processes, which indicated that cost models would need to address projects using a mix of process models (Rapid Application Development and prototyping; waterfall; incremental; evolutionary; spiral; COTS/reuse-driven; design-to-cost/schedule). They would have to address more sophisticated forms of reuse, evolution, breakage, and COTS integration. They would have to reflect new technologies such as GUI builders, object-oriented methods, application generators, and Internet-based distributed development.

Based on the cost model review and technology forecast, we began a series of iterations with the COCOMO 2.0 Affiliates to converge on a set of functional forms and baseline cost drivers for the model - or, more accurately, family of models, as COCOMO 2.0 is organized into three stages. Stage 1 is an Applications Composition model, focused on rapid application development or prototyping with GUI builders, COTS products, and distributed middleware support packages. Its basic sizing parameter is Object Points, a complexity-weighted count of the number of screens developed, reports generated, and third-generation language modules integrated. It is based on recent work done in this area by Banker, Kauffman, and Kumar. Stages 2 and 3 of COCOMO 2.0 have functional forms similar to the original COCOMO (COCOMO 81). Stage 2 of COCOMO 2.0 is an Early Design model, tailored for use in early stages of the life cycle when only partial information is available on the ultimate nature of the software project, product, platform, and personnel. Stage 3 is a Post-Architecture model, tailored for use once the project has resolved its major risk items, and has converged on a thorough and compatible set of plans, requirements, and architecture for the product.

Stage 2 uses function points as its primary size metric, but accommodates the use of source lines of code when appropriate. Stage 3 supports a mixed use of function points for some components and source lines of code for others. Sizing for both stages is adjusted for software breakage during development, and reuse or adaptation of existing software. The model for reuse and adaptation has been changed from a linear to a nonlinear model, based on the results of Selby's and others' data and analyses.

The Stage 3 model has 17 multiplicative cost drivers, most of which are similar to counterparts in COCOMO 81. Turnaround Time and Modern Programming Practices have been dropped, and four cost drivers have been added based on workshops with Affiliates. These cover multisite development, amount of documentation, personnel continuity, and development for subsequent reuse. The Stage 2 model compresses the 17 stage 3 cost drivers into 7 aggregate cost drivers; for example, a single Personnel Experience cost driver is used in place of Applications Experience, Platform Experience, and
Language & Tool Experience.

Both Stages 2 and 3 replace the COCOMO 81 development modes by a set of 5 scale factors which adjust the additional costs of scaling up to large projects (diseconomies of scale). Two of these, Precededness and Flexibility, cover the effects of the COCOMO 81 development modes. Another two, Architecture/Risk Resolution and Team Cohesion, are adapted from counterparts in Ada COCOMO. The fifth, Process Maturity, is a test of the hypothesis that increased SEI-CMM process maturity levels reduce costs by reducing the rework involved in immature processes.

Currently, COCOMO 2.0 has been calibrated to 65 Affiliate project data points. The overall fit is reasonably good, but the increased variability of current data due to variability of process and product definitions leaves the analysis of detailed effects such as process maturity currently short of statistical significance. Overall, we are finding that collecting well-defined data points is a much more arduous process than it was for COCOMO 81.

As with Capers Jones, I am finding that the software cost modeling field needs to keep running to stay current with the rapid pace at which the software field reinvents itself, but that this continuing challenge makes it an increasingly significant and fascinating field in which to work.

**Acknowledgments**

This research is sponsored by the Advanced Research Projects Agency (ARPA) through Rome Laboratory under contract F30602-94-C-0195 and by the Affiliates of the USC Center for Software Engineering.

The current Affiliates are:

- Aerospace Corporation
- Air Force Cost Analysis Agency
- AT&T
- Bellcore
- DISA
- Electronic Data Systems Corporation
- E-Systems
- Hughes Aircraft Company
- Interactive Development Environments
- Institute for Defense Analysis
- Jet Propulsion Laboratory
- Litton Data Systems
Lockheed Martin Corporation
Loral Federal Systems
Motorola Inc.
Northrop Grumman Corporation
Rational Software Corporation
Rockwell International
Science Applications International Corporation
Software Engineering Institute (CMU)
Software Productivity Consortium
Sun Microsystems, Inc.
Texas Instruments
TRW
U.S. Air Force Rome Laboratory
U.S. Army Research Laboratory
Xerox Corporation