Achievements and Challenges in Software Resource Estimation

Barry W. Boehm
University of Southern California
941 W. 37th Place
Los Angeles, CA 90089
+1 (213) 740-5703
boehm@usc.edu

Ricardo Valerdi
Massachusetts Institute of Technology
77 Vassar Street
Cambridge, MA 02139
+1 (617) 253-8583
rvalerdi@mit.edu

ABSTRACT
This paper summarizes major achievements and challenges of software resource estimation over the last forty years. We address critical issues that enabled major achievements such as the development of good model forms, criteria for evaluating models, methods for integrating expert judgment and statistical data analysis, and processes for developing new models that cover new software development approaches. Future trends in software development and evolution processes are projected, along with their implications and challenges for future software resource estimation capabilities.

Keywords
Software cost estimation, effort estimation, schedule estimation, software engineering economics, software metrics.

1. INTRODUCTION
Software resource estimation methods and models have had a major impact on successful software engineering practice. They provide milestone budgets and schedules that help projects determine when they are making satisfactory progress and when they are in need of corrective action. They help decision makers make software cost-schedule-value tradeoff analyses and investment, outsourcing, COTS product, and legacy software phaseout decisions. They help organizations prioritize investments in improving software productivity, quality, and time to market. They are included as essential capabilities in virtually all major software capability maturity models, software engineering textbooks, and software engineering bodies of knowledge.

A counterpart appreciation of their contribution involves what happens to projects that go forward without good estimates of their milestone budgets and schedules. All too frequently, they commit to develop too much software within the agreed-on budget and schedule, have no framework to determine whether they are on track or not, and end up with serious overruns or terminated projects. The Standish Group's 1995 CHAOS survey of over 350 organizations and 8000 projects produced the following results [43]:

- 16% delivered within budget and schedule
- 31% cancelled before completion
- 53% overrun in budget or schedule
  - 89% average budget overrun
  - 122% average schedule overrun
  - 61% average of originally specified content delivered

Subsequent Standish survey results have somewhat improved (29% within budget and schedule in 2004), but still leave much opportunity for further improvement.

1.1 Relations to Other Software Engineering Areas
Progress in software resource estimation is deeply intertwined with progress in other software engineering areas, particularly software architectures, processes, programming, and requirements engineering. Estimation results showing the asymptotic escalation of software costs as utilization of hardware resources approached 100% led to information architectures with significant memory and CPU cycle margins [7]. Software cost-schedule tradeoff relations showing that excessive schedule compression led to asymptotic cost increases [36] led to more realistic project schedules and research on processes enabling more rapid development [4, 30].

A well-calibrated COCOMO II software cost driver showing that inadequate requirements engineering and architecting led to disproportionate extra rework focused more emphasis on requirements engineering and architecting [10]. The need to determine appropriate milestone endpoints for estimating spiral process model costs and schedules led to the development of spiral anchor point milestones [13] subsequently used in many projects and in the Rational Unified Process (RUP) [38, 24, 22]. On the other hand, the Rational Unified Process work breakdown structure was used to define the project activities included in COCOMO II software cost estimates. Another well-calibrated COCOMO II scale factor, Software Process Maturity, provided the best evidence to date that increased process maturity correlated with increased software productivity. The regression analysis of 161 projects indicated a 4-11% per maturity level increase, excluding the effect of other factors, depending on product size [15].

Resource estimation also has deep interactions with software product engineering. Most of the leading cost and schedule estimation models discussed in later sections have parameters relating software project costs and schedules to such product attributes as complexity, required reliability, hardware constraints, database size, and software reuse. These provide major contributions to the key practice of software architecture tradeoff analysis. This is a necessary discipline for meeting simultaneous stakeholder needs for high performance, reliability, usability, evolvability, et al., within project budget and schedule constraints [16]. Additional estimation models in this regard include performance models [42], reliability models [33], and

real-options models for analyzing investments in software modularity to provide future evolvability options [44].

Software product engineering and resource estimation are particularly highly coupled in the area of software product line reuse. Most of the books in this area integrate cost and schedule estimation considerations in software domain architecting for product line reuses [35, 37, 22, 17, 23]. An example success story is Hewlett-Packard's investment in product line reuse to reduce time to market from 48 months to 12 months [26].

These interactions all come together in the emerging area of value based software engineering. This involves not just business case analysis in the early stages, but also the integration of value considerations into all parts of software engineering, including life cycle processes, requirements engineering, architecting, development, test, tool support, release planning, and project management. Chapters on these topics are in a recent book on value-based software engineering, along with fundamentals of using cost, schedule, and value models to determine the relative returns on investment of alternative software engineering decisions [6].

2. EARLY ACHIEVEMENTS: 1965-1985

The earliest achievements in software resource estimation are tied to specific models developed, calibrated, and published as early as 1965, the most prominent of which are shown in Table 1.

Table 1. Early software resource estimation models

Model	Year
SDC	1965
TRW Wolverton	1974
Putnam	1976
Doty	1977
RCA Price S	1977
IBM-FSD	1977
Boeing	1977
Function points (IBM)	1979
Bailey-Basili Meta-Model	1981
SoftCost, -R	1981
COCOMO	1981
Jensen/SEER	1983
Estimacs	1983
SPQR/Checkpoint	1985

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICSE '06, May 20-28, 2006, Shanghai, China.
Copyright 2006 ACM 1-58113-000-0/00/0004…$5.00.

2.1 The Search for Good Model Forms

The most critical early resource estimation modeling issue was to find the right parametric forms for estimating software project effort and schedule. The experiences in analyzing the SDC database convinced people that a purely linear additive model did not work well. It was clear that the behavioral phenomenology of software development was not consistent with effort estimators combining such factors as size and complexity in linear additive forms.

For a while, the best relationships people could find involved estimating effort as a linear function of size, modified by a complexity multiplier. The initial complexity multiplier came from the nonlinear distribution of programming rates in the 169-project SDC sample shown in Figure 1. For example, if one's project involved developing 10,000 object instructions of software that was considered to be more complex than 80% of the projects in the SDC sample, one would determine that its programming rate would be roughly 7 person-months per thousand object instructions. Then the estimated project effort would be 7*10=70 person-months.

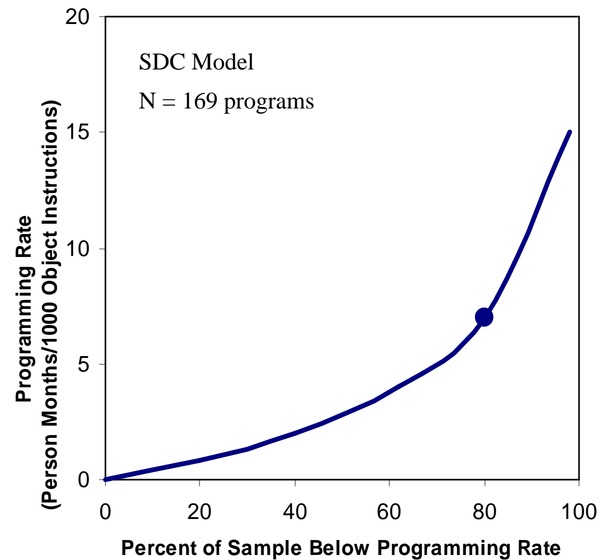

Figure 1. SDC Model Example [8].

Most of the successful early effort estimation models employed variants of this approach. The TRW Wolverton model, the Boeing Black model, and early versions of the RCA PRICE S model employed different programming rate curves for different classes of software (scientific vs. business vs. embedded real-time; familiar vs. unfamiliar; and/or for different software life-cycle phases).

By the late 1970's, the software community was finding that simple complexity ratings were not adequate for many software situations that produced different programming rates. Some organizations found that their programming rates were more productive rather than less productive for higher-complexity software, as they assigned their best people to the most complex projects. Most importantly, though, the complexity rating was purely subjective. There was no objective way of determining if a project was at the 60% or 80% level, but going from 80% to 60%

in Figure 1 will reduce the estimated effort by roughly a factor of 2. Some organizations were also finding that their software projects exhibited economies or diseconomies of scale, involving estimation relations with exponential functions of size.

Quite a few estimation models were developed in the late 1970's. Multiple combinations of multiplicative cost drivers were employed in the Doty, IBM, Walston-Felix, and intermediate versions of the PRICE S model. Bailey and Basili experimented with an additive combination of productivity multipliers and an exponential scale factor. The Putnam SLIM model developed exponential relationships linking size, productivity, and schedule. Alternative sizing methods such as function points [3] were being developed to support better early size estimation.

Positive and negative experiences with these and other models led to a set of criteria for developing additive, exponential, multiplicative, and asymptotic model factors. This underlying logic provided the general form for the Constructive Cost Model (COCOMO) in 1981.

$$PM = A * (Size)^{B} * (EM)$$

ADDITIVE EXPONENTIAL MULTIPLICATIVE

Where:

PM = Person Months

A = calibration factor

Size = measure(s) of functional size of a software module that has an additive effect on software development effort

B = scale factor(s) that have an exponential effect on software development effort

EM = effort multipliers that influence software development effort multiplicatively.

The general rationale for whether a factor is additive, exponential, or multiplicative comes from the following criteria:

1. A factor is additive if it has a local effect on the included entity. For example, adding another source instruction, function point entity, module, interface, operational scenario, or algorithm to a system has mostly local additive effects.

2. A factor is multiplicative or exponential if it has a global effect across the overall system. For example, adding another level of service requirement, development site, or incompatible customer has mostly global multiplicative or exponential effects. Consider the effect of the factor on the effort associated with the product being developed. If the size of the product is doubled and the proportional effect of that factor is also doubled, then it is a multiplicative factor.

3. If the effect of the factor is more influential for larger-size projects than for smaller-size projects, often because of the amount of rework due to architecture and risk resolution, team compatibility, or readiness for system-of-systems integration [25], then it is treated as an exponential factor.

4. The effects of asymptotic cost driver forms generally interact multiplicatively with other cost factors. Their effects on cost tend to increase unboundedly as they reach constraint boundaries. Examples are reaching limits on available computer execution cycles or main memory capacity, or on achievable schedule compression. Such factors can be calibrated as a multiplicative coefficient times the shape of the asymptotic curve.

These rules have been applied to the development of the COCOMO model and associated models. The assumptions made about the Cost Estimating Relationships (CER) in these models require that they be validated by historical projects. A crucial part of developing these models is finding representative data that can be used to calibrate the size, multiplier, and exponential factors contained in the models. Sixty-three data points were provided with the initial release of COCOMO. This allowed the software engineering research community to validate the model and experiment with the data set to elaborate the concepts presented as well as investigate new ones.

2.2 Development of Model Evaluation Criteria

Models are frequently evaluated for the goodness of their ability to estimate software development. The following criteria are most helpful in evaluating the utility of a cost model for practical estimation purposes [8]:

5. *Definition*. Has the model clearly defined the costs it is estimating, and the costs it is excluding? One model's answer to this question was "What would you like it to include?" -- not a strong confidence-booster.

6. *Fidelity*. Are the estimates close to the actual costs expended on the projects? An important follow-up question is the next question on Scope.

7. *Scope*. Does the model cover the class of software projects whose costs you need to estimate? Parametric models are generally not as strong for very small and very large applications. Function points are a better match to business applications and early estimation of GUI-based applications than to algorithm-intensive scientific applications.

8. *Objectivity*. Does the model avoid allocating most of the software cost variance to poorly calibrated subjective factors (such as complexity)? That is, is it harder to jigger the model to obtain any results you want?

9. *Constructiveness*. Can a user tell why the model gives the estimates it does? Does it help the user understand the software job to be done? Neural net models whose internals bear no relation to software phenomenology are a counterexample. Proprietary models were initially reluctant to discuss their internals, but have become increasingly communicative.

10. *Detail*. Does the model easily accommodate the estimation of a software system consisting of a number of subsystems and units? Does it give (accurate) phase and activity breakdowns? A limiting factor here is that

the greater the model detail, the less data there is to support its calibration. Newer process models such as overlapping incremental development make the data less precise.

11. *Stability*. Do small differences in inputs produce small differences in output cost estimates? Some models, such as the Doty model, exhibited factor-of-2 discontinuities at model boundaries due to the binary nature of their inputs.

12. *Ease of Use*. Are the model inputs and options easy to understand and specify? Some related criteria are tailorability and composability for ease of calibration; addition of new parameters; composition with sizing models, risk analyzers, or generators of proposals and project plans; or model simplification for ease of early estimation, as discussed next.

13. *Prospectiveness.* Does the model avoid the use of information which will not be known until the project is complete? The greatest difficulty here is with source lines of code (SLOC) as a model's size parameter. A number of higher level quantities have been tried, such as number of requirements, use cases, web objects, function points, feature points, object points, and application points. A major difficulty with such parameters is that their number increases as the product is better defined. Coming up with easy-to-understand definitions of the right level of detail is difficult. The best framework to date is the "clouds-sea level-bottom feeder" goal hierarchy in [18].

14. *Parsimony.* Does the model avoid the use of highly redundant factors, or factors which make no appreciable contribution to the results? The Walston-Felix model had four separate factors which were highly correlated for IBM projects: use of top-down development, structured programming, structured walkthroughs, and chief programmer teams. This can cause both multiple-counting of correlated effects or difficulties in performing regression analyses.

For the most part, the significance of each of these criteria is reasonably self-evident. From a generic standpoint, the criteria have also proven to be very helpful in the development and evaluation of other cost estimation models in the COCOMO suite and elsewhere.

2.3 Emergence of a Model Marketplace and Community of Interest

The early 1980's marked the development of a software resource estimation community of interest, including conferences, journals, and books in the area. These helped in socializing the addressal of the issues above, and the emergence of several estimation models that passed both usage tests and tests of market viability. These included the refinement of earlier models such as PRICE S and SLIM, and the development of early-1980's models such as SPQR/Checkpoint, Estimacs, Jensen/SEER, Softcost-R, and COCOMO and its commercial implementations such as PCOC, GECOMO, COSTAR, and Before You Leap. These models were highly effective for the largely waterfall-model, build-from-scratch software projects of the 1980's, but began to encounter new classes of challenges, as discussed next.

3. INTERMEDIATE ACHIEVEMENTS: 1985-2005

3.1 1985-1995: Mainstream Refinements

The 1985-1995 time period primarily involved proprietors of the leading cost models addressing problem situations brought up by users in the context of their existing mainstream capabilities. A good example has been risk analyzers, either based on Monte Carlo generation of estimate probability curves, or based on agent-based analysis of risky combinations of cost driver ratings. Another good example has been the breakdown of overall cost and schedule estimates by phase, activity, or increment.

The most significant extensions during this period were in the area of software sizing. As discussed under the Prospectiveness criterion, accurate early estimation of SLOC is a major challenge. Some comparison-oriented methods involving paired size comparisons, ranking methods, and degree-of-difference comparisons were developed. They have been helpful, but their performance is spotty and expert-dependent. During 1985-1995, the Function Point community made a major step forward in defining uniform counting rules for its key size elements of inputs, outputs, queries, internal files, and external interfaces, along with associated training and certification capabilities. This made Function Points a good match for business applications, which tend to have simple internal business logic, but less good for scientific and real-time control applications with more complex internals.

Function point extensions such as feature points, COSMIC function points, and 3D function points have been developed for these, but their definitions and counting rules have not converged as well as those of the initial function points quantities. As discussed under Prospectiveness, other higher-level early sizing metrics have been developed, but their counting rules and granularity standards have been more difficult to standardize than those of function points. Within individual organizations, some of the higher-level early sizing metrics have been made to work, but the challenge of developing a general early software sizing metric remains high.

3.2 1995-2005: Proliferation of Software Development Styles

The development of COCOMO II starting in 1995 was based primarily on the realization that the 1981 COCOMO model's assumptions of sequential waterfall-model development, three stratified development modes, and occasional software reuse with linear savings were getting too obsolete. We projected software applications development classes out to 2005, and developed COCOMO II to address them [10]. This involved developing new anchor point milestones to serve as the endpoints for concurrent spiral-model cost and schedule estimates; developing a more realistic nonlinear reuse model; adding exponential scale factors for such scalability controllables as process maturity and architecture/risk resolution; adding new cost drivers for such phenomena as development for reuse, distributed software development, and personnel continuity; dropping obsolete cost

drivers such as turnaround time, and enabling the use of alternative early sizing methods such as function points.

Each of these changes was supported by valuable research results, and by our experiences in trying to tailor the original COCOMO to new situations. Nonlinear effects of software reuse were based on research at the NASA Software Engineering Laboratory [40], maintenance effort distributions [34], and nonlinear software integration effects [19]. Anchor point milestones and their phase distributions were supported by work at Rational, AT&T, and spiral model usage at TRW [38, 29]. Process maturity characterization was supported by collaboration with CMU-SEI and its associated definitions and data on productivity effects [20]. Estimation of the relative cost of writing for reuse was supported by the software reuse experiences discussed in Section 1.1. Exponential diseconomies of scale effects in software development were addressed in [5].

The resulting COCOMO II model was successfully calibrated to 161 carefully collected and verified project data points. Its predictions within this sample were within 30% of the actuals 75% of the time for effort (80% with local calibration) and 64% for schedule (75% with local calibration). It has been broadly adopted and incorporated into several commercial cost estimation models. However, although it does a good job for the 2005 development styles projected in 1995, it does not cover several newer development styles well. This led us to develop additional COCOMO II-related models to address these. Table 2 provides a summary of the current COCOMO II suite of models.

Counterpart commercial software cost model companies, such as CostXpert, Galorath (SEER), Price Systems (PRICE S), and Softstar Systems (COSTAR), have become Affiliates of the USC Center for Software Engineering (CSE), have participated in the research on these model extensions, and are developing counterpart extensions to their commercial offerings. This enables the software resource estimation field to share expertise while offering users a range of solution approaches.

Table 2. Current COCOMO suite of models.

Model	Description
COCOMO II	Constructive Cost Model
COINCOMO	Constructive Incremental COCOMO
COQUALMO	Constructive Quality Model
COPLIMO	Constructive Product Line Investment Model
CORADMO	Constructive Rapid Application Development Model
COPROMO	Constructive Productivity-Improvement Model
COCOTS	Constructive Commercial-Off-The-Shelf Cost Model
COSYSMO	Constructive Systems Engineering Cost Model
COSOSIMO	Constructive System of Systems Integration Cost Model
COSECOMO	Constructive Security Cost Model

3.3 Coping with Future Trends

One response to these trends has been the development of a modeling methodology that serves as a framework for developing specialized models in software engineering. Shown in Figure 3, this 8-step methodology has been successfully used to develop the COCOMO II family of models.

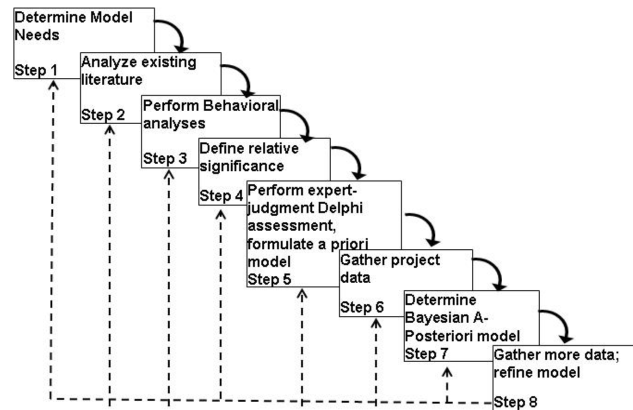

Figure 3. 8-step modeling methodology.

Step 1: Determine Model Needs
Identifying a need and a group of supporters for a model is the first and most important step in the process. Unless there are stakeholders willing to provide funding, expertise, and data there is no market for a model. Similar to software requirements determination, this step involves identifying success-critical stakeholders and their modeling needs.

Step 2: Analyze Existing Literature
The analysis of existing literature aids in the understanding of underlying phenomenology such as sources of cost, defects, etc. This step also helps identify promising or unsuccessful model forms and a list of the most promising model parameters.

Step 3: Perform Behavioral Analysis
Insight garnered from step 2 can suggest a set of factors that correlate to the aspect of software engineering effort being targeted. This preliminary list can serve to focus the initial scope of the model.

Step 4: Define Relative Significance
Prioritizing the items on the list by defining the relative significance is an important step in creating a concise model. Typically, this condensing can be done with the use of subject matter experts with techniques such as the Analytical Hierarchy Process.

Step 5: Perform Expert-judgment Delphi
The Wideband Delphi technique is a means of guiding a group of experts to a consensus of opinion on each parameter's outcome-influence value. Data definitions and rating scales are also established for significant parameters. This exercise is repeated multiple times to give individuals the opportunity to re-estimate their values after group discussion. This process often uncovers overlaps and changes in outcome drivers.

Step 6: Gather & Analyze Empirical Data
It is best to collect initial data via interviews to avoid misinterpretation of parameters and rating scales as well as uncover hidden assumptions. Next, data should be piloted with early adopters to identify data definition ambiguities and availability problems. Once stabilized, the data collection process

can shed some insight on historical information in organizations that can help calibrate the model. Initial data analysis may require model revision.

Step 7: Perform Bayesian Calibration
Multiple regression analysis of project data points produces outcome-influenced values. For COCOMO II, 161 data points produced mostly statistically significant parameters values. A recent enhancement was the introduction of the Bayesian approach which allows the combination of expert opinion from step 5 and empirical data from step 6 [14]. The Bayesian approach favors experts when they agree, or historical data where results are significant. The Bayesian version of COCOMO II performed considerably better than the pure data-regression version in estimating the 161 projects in the COCOMO II database, showing that the inclusion of expert judgment was able to produce a more robust model by avoiding overly chasing noisy data or outliers.

Step 8: Gather more data & refine model
Once a calibrated model has been released and usage experience has been recorded, the model enters a feedback loop involving improvements via adjustments. In the case of COCOMO, it was determined that a Programmer Unfamiliarity factor was needed after users provided feedback on the model.

3.4 Usage Example: COCOTS

Development of COCOTS began with a USC-CSE industry-government Affiliates' workshop on processes and architectures for the development and evolution of COTS-based systems. A breakout group at the workshop identified model needs (Step 1) and sources of further information for Step 2, including a summary of a proprietary model developed by one of the Affiliates. This and other literature reviewed in Step 2 led to a hypothesis that a COCOMO II-like model using source lines of COTS-integration glue code as a sizing parameter; some COCOMO II effort multipliers such as personnel capability, continuity, and experience; and some COTS-specific effort multipliers such as COTS product maturity, vendor support, interface complexity, and performance limitations would make a good estimation model.

At this point, the lead PhD student, Chris Abts, performed a behavioral analysis of the candidate cost drivers (Step 3). This was iterated with the Affiliates at two follow-up workshops, which also determined the relative significance of the cost drivers and ended up dropping and combining some. The remainder of Step 4 involved development, review, and iteration of the cost driver rating scales and model data definitions, including definition of the relationships between COCOTS estimates and COCOMO II estimates. These enabled the execution of the Delphi process with Affiliate and USC-CSE experts (Step 5), and the beginning of Step 6 with initial collection of two Affiliate pilot project data points (which identified further data definition clarifications needed).

The two Affiliate data points appeared compatible with the model. However, at this point, we completed six well-instrumented COTS-based applications as part of our series of campus e-services team project applications. Four of these did not fit the model well at all. In analyzing the data and interviewing the developers, we found that these projects were spending a great deal of effort in COTS assessment and tailoring, none of which resulted in the generation of glue code. This caused Abts to revise his hypothesis, and go back to Step 3 to perform behavioral analyses and develop forms for estimating COTS assessment and tailoring effort. The resulting revised model, with some significant support from the FAA, ONR, USAF/ESC, and the USC-CSE Affiliates, was able to achieve a reasonably accurate set of estimates across a 20-project set of project data points [2].

Beyond this, the data collection and analysis effort provided us with further insights on COTS-based application development critical success factors and processes. Using the principle of "process happens where the effort happens," we were able to develop, apply, and validate a set of composable process elements for COTS-based applications development [47].

The 8-step process has also been used on such COCOMO II extensions as COQUALMO, COSYSMO [45], and an elaboration of the COCOMO II Tools effort multiplier. Further uses of the 8-step process are underway for COCOMO II extensions addressing computer security cost increases, and software-intensive systems of systems integration. The process has enabled successful responses to the challenges of developing new resource estimation models for new software development styles and objectives.

3.5 Alternative Model Forms

Concurrently, explorations have been made into separate alternative software resource estimation model forms such as analogy or case-based estimation, learning-based estimation such as neural nets or other learning engines, and systems dynamics modeling.

Analogy or case-based estimation [32, 41] uses metadata about a project to be estimated (size, Type, process, domain, etc.)to base an estimate of its required resources on the resources required for the most similar projects in a large database such as the ISBSG database of 3000 software projects [21]. In one study [39], analogy-based estimation performed better than alternative estimation methods in 60% of the cases, but worst in 30% of the cases, indicating some promise but need of further refinement.

Neural net models use layouts of simulated neurons and training algorithms to adjust neuron connection parameters to learn the best fit between input parameters and values to be estimated. In some situations, accuracies of 10% have been reported [46], but in many cases, estimation of projects outside the training set has been much less accurate. And as discussed in Section 2.2 under Constructiveness, they do not provide constructive insights on the software job to be done. Other machine learning techniques have recently been used to successfully determine reduced-parameter versions of parametric cost models [31].

Systems dynamics models integrate systems of differential equations to determine the flow of effort, defects, or other quantities through a process as a function of time. They are very good for understanding the effects of dynamic relations among software development subprocesses, such as the conditions under which Brooks' Law holds (adding more people to a late software project will make it later) [28]. Pioneering work in this area has been done in [1] for general software project relationships, and in [27] for interactions among effort, schedule, and effect density, in performing software inspections.

Each of these model forms provides complementary perspectives to those of parametric models. Challenges for the future include finding better ways to integrate their contributions.

4. FUTURE CHALLENGES AND RESPONSES

4.1 The Receding Horizon

In the 1980's, our vision of the future was that those in the software estimation field were like the Tycho Brahes in the sixteenth century, compiling observational data that later Keplers and Newtons would use to develop a quantitative science of software engineering [9]. As we went from unprecedented to precedented software applications, our productivity would increase and our error in estimating software costs would continue to decrease, as on the left side of Figure 2.

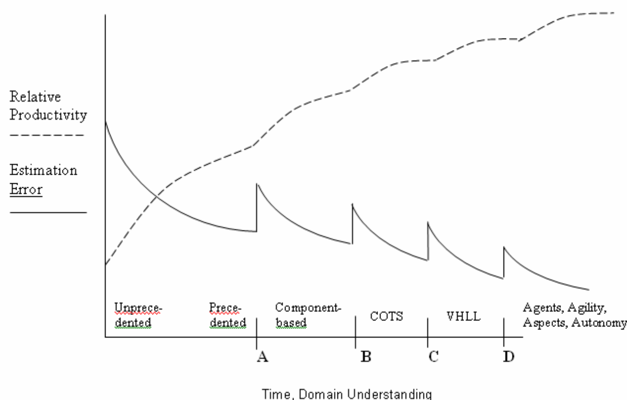

Figure 2. Software Estimation – The Receding Horizon.

However, this view rested on the assumption that, like the stars, planets, and satellites, software projects would continue to behave in the same way as time went on. But, as we have repeatedly seen, this assumption turned out to be invalid. The software field is continually being reinvented via structured methods, abstract data types, information hiding, objects, patterns, reusable components, commercial packages, very high level languages, rapid application development (RAD) processes, model-driven development, agent technology, agile methods, aspect technology, automatic software, and so on.

Thus, at point A in Figure 2, increased domain understanding led to the ability to develop and reuse software components. This introduced a new boost in productivity, but also increased the estimation error of existing resource estimation models, until model refinements for software reuse were developed and calibrated.

With each reinvention, software cost estimation and other software engineering fields need to reinvent themselves just to keep up, resulting in the type of progress shown in the right side of Figure 2 [11]. The most encouraging thing in Figure 2, though, is that leading existing and new companies are able to build on this experience and continue to increase our relative productivity in delivering the huge masses of software our society increasingly depends on. Our biggest challenge for the future is to figure out how to selectively prune the parts of the software engineering experience base that become less relevant, and to conserve and build on the parts with lasting value for the future.

4.2 Future Trends in Software Engineering

In order to anticipate and prepare for future trends in software resource estimation and related technology, we periodically assess future trends affecting software engineering practices. A recent paper [47] identifies seven relatively surprise-free trends and two likely but relatively unpredictable wild-card trends. The seven surprise-free trends are:

1. An increased emphasis on users and end value;
2. Increasing software criticality and need for dependability;
3. Increasingly rapid change;
4. Increasing IT globalization and need for interoperability;
5. Increasingly complex systems of systems;
6. Increasing needs for COTS, reuse, and legacy software integration;
7. Computational plenty.

The two wild-card trends are:

8. Increasing software autonomy;
9. Combinations of biology and computing;

The surprise-free trends point to the need for better resource estimation models for integrating cost and value estimation; analyzing tradeoffs among cost, schedule, dependability, and agility; estimating systems of systems and enterprise software integration costs and schedules; estimating the effects of integrating new custom software, COTS, open source, reusable components and legacy software; and estimating the effects of new software generation and verification capabilities enabled by computational plenty. The wild-card trends point to the need to monitor trends in autonomy and biocomputing to anticipate future discontinuities in productivity and estimation accuracy.

5. CONCLUSIONS

5.1 Major Technical Achievements

Despite the challenges in software resource estimation, major technical achievements have enabled the development of sophisticated estimation models. Among the most significant achievements are:

- Finding appropriate functional forms for estimation models. Determining which parameters contribute in additive, multiplicative, exponential, and asymptotic ways.
- Statistically significant model calibration. Obtaining critical masses of carefully defined, multi-parameter project data that produce robust and statistically significant parameter values.
- Bayesian combination of expert judgment and statistical data analysis. Providing the ability to bootstrap model usage and accumulation of critical masses of project data.
- Model reinvention to accommodate new development paradigms. Development of anchor point milestones enabling not only principled estimation but controllable concurrent engineering for spiral and evolutionary

software processes [12]. Models for rapid development, reuse, product lines, and COTS integration.

- New sizing parameters. These include function points, object-oriented metrics, and specification-based sizing parameters.
- Methodology for development, calibration, and evolution of new models. These include the multi-step process for exploration, analysis, definition, calibration, and refinement of parametric estimation models and techniques for determining viable reduced-parameter models in special domains.
- Contributions to value-based software engineering. Integration of resource investment levels and benefits estimation models into return on investment models.

5.2 Impact on Software Engineering Practice

These achievements have also impacted the management of software engineering. Practitioners have benefited from these advancements in the following areas:

- Basis of project stakeholder negotiation and expectations management. Ability to avoid overcommitment to infeasible budgets and schedules.
- Basis of project planning and control and impact on processes. Anchor point milestones enable control of complex concurrent engineering processes. Schedule/cost/quality as independent variable processes enable meeting targets by prioritizing and adding or dropping marginal-priority features.
- Improved project performance. Phase and activity estimates provide a framework for better progress monitoring and control.
- Framework for process improvement. Enablement of improved planning realism, monitoring and control, model improvement, and productivity/cycle time/quality/business value improvement.
- Contributions to communities of interest. Besides the core estimation community, these include the communities concerned with empirical methods, metrics, economics-driven or value-based software engineering, systems architecting, software processes, and project management.

5.3 Challenges for the Future

As we expect software engineering to continue changing, future challenges will introduce new opportunities for improved methods and tools. The most significant are:

- Integration of software and systems engineering estimation. Challenges include compatible sizing parameters, schedule estimation, and compatible output estimates.
- Sizing for new product forms. These include requirements or architectural specifications, stories, and component-based development sizing.
- Exploration of new model forms. Candidates include case-based/analogy-based estimation, neural nets, system dynamics, and use of new sizing, complexity, reuse, or volatility parameters.
- Maintaining compatibility across multiple classes of models. Including compatibility of inputs, outputs, and assumptions.
- Total cost of ownership estimation. In addition to software development, this can include estimation of costs of installation, training, services, equipment, COTS licenses, facilities, operations, maintenance, and disposal.
- Benefits and return on investment estimation. This can include valuation of products, services, and less-quantifiable returns such as customer satisfaction, controllability, and staff morale.
- Accommodating future software engineering trends. These can include ultra large software-intensive systems, ultrahigh dependability, increasingly rapid change, massively distributed and concurrent development, and effects of computational plenty, autonomy, and biocomputing.

These trends contribute to the ever-receding horizon of perfectable resource estimation models, but keep the model development and evolution community in a highly stimulating and challenge-driven state.

6. REFERENCES

[1] Abdel-Hamid, T., Madnick, S., *Software Project Dynamics*: Prentice Hall, 1991.

[2] Abts, C., "A Cost Estimation Model for COTS Integration," vol. PhD. Los Angeles: USC, 2004.

[3] Albrecht, A. J., Gaffney, J., "Software function, source lines of code, and development effort prediction: A software Science validation," *IEEE Transactions on Software Engineering*, vol. Vol. SE-9, pp. pp. 639-648, 1983.

[4] Arthur, L., *Rapid Evolution Development*: Wiley, 1992.

[5] Banker, R., Chang, H., and Kemerer, C., "Evidence on economies of scale in software development," *Information and Software Technology*, pp. pp. 275-282, 1994.

[6] Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Gruenbacher, P., "Value Based Software Engineering," Springer Verlag, 2005.

[7] Boehm, B. W., "Software And Its Impact: A Quantitative Assessment," *Datamation*, pp. pp. 48-59, 1973.

[8] Boehm, B. W., *Software Engineering Economics*: Prentice Hall, 1981.

[9] Boehm, B. W., "Software Engineering Economics," *IEEE Trans on Software Engineering*, 1984.

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

[11] Boehm, B. W., Abts, C., Chulani, S., "Software Development Cost Estimation Approaches," USC-CSE Tech Report CSE-TR-2000-505 2000.

[12] Boehm, B. W., Hansen, W., "The Spiral Model as a Tool for Evolutionary Acquisition," *CrossTalk*, 2001.

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

[14] Chulani, S., Boehm, B. W., Steece, B., "Bayesian Analysis of Empirical Software Engineering Cost Models," *IEEE Transactions on Software Engineering*, pp. pp. 513-583, 1999.

[15] Clark, B., "Quantifying the Effects on Effort of Process Improvement," *Software*, pp. pp. 65-70, 2000.

[16] Clements, P., Kazman, R., Klein, M., *Evaluating Software Architectures*: Addison Wesley, 2002.

[17] Clements, P., Northrop, L., *Software Product Lines*: Addison Welsey, 2002.

[18] Cockburn, A., *Writing Effective Use Cases*: Addison Wesley, 2001.

[19] Gerlich, R., and Denskat, U., "A cost estimation model for maintenanace and high reuse," presented at Proceedings, ESCOM, Ivrea, Italy, 1994.

[20] Hayes, W., and Zubrow, D., "Moving on Up: Data and experience doing CMM-based process improvement," CMU/SEI-95-TR-008 1995.

[21] International Software Benchmark & Standards Group, "Data R8," 2005.

[22] Jacobson, I., Booch, G., Rumbaugh, J., *The Unified Software Development Process*: Addison Wesley, 1999.

[23] Jacobson, I., Griss, M., Jonsson, P., *Software Reuse*: Addison Wesley, 1997.

[24] Kruchten, P., *The Rational Unified Process*, 2nd ed: Addison Wesley, 2001.

[25] Lane, J., Valerdi, R., "Synthesizing System-of-Systems Concepts for Use in Cost Estimation," presented at IEEE Conference on Systems, Man, and Cybernetics, Waikoloa, HI, 2005.

[26] Lim, W., *Managing Software Reuse*: Prentice Hall, 1999.

[27] Madachy, R., "System Dynamics Modeling of an Inspection-Based Process," presented at Proceedings, ICSE 18, ACM/IEEE, 1996.

[28] Madachy, R., *Software Process Dynamics*: IEEE-CS Press, 2006 (to appear).

[29] Marenzano, J., "System Architecture Review Findings," in *ICSE 17 Architecture Workshop Proceedings*, D. Garlan, Ed.: CMU, 1995.

[30] McConnell, S., *Rapid Development*: Microsoft Press, 1996.

[31] Menzies, T., Chen, Z., Port, D., and Boehm, B., "Reduced-Parameter Estimation Models," presented at Proceedings, PROMISE Workshop, ICSE, 2005.

[32] Mukhopadhyay, T., Vincinanza, S., Prietula, M., "Examining the Feasibility of a Case-Based Reasoning Model for Software Effort Estimation," *MIS Quarterly*, vol. 16, pp. pp. 155-171, 1992.

[33] Musa, J., *Software Reliability Engineering*: McGraw Hill, 1999.

[34] Parikh, G., Zvegintsov, N., "The World of Software Maintenance," in *Tutorial on Software Maintenance*: IEEE CS Press, 1993, pp. pp. 1-3.

[35] Poulin, J., *Measuring Software Reuse*: Addison Wesley, 1997.

[36] Putnam, L., "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering*, pp. pp. 345-361, 1978.

[37] Reifer, D., *Practical Software Reuse*: Wiley, 1997.

[38] Royce, W., *Software Project Management*: Addison Wesley, 1998.

[39] Ruhe, M., Jeffery, R, and Wieczorek, I., "Cost Estimation for Web Application," presented at Proceedings, ICSE 2003, ACM/IEEE, 2003.

[40] Selby, R., "Empirically analyzing software reuse in a production environment," in *Software Reuse: Emerging Technology*, W. Tracz, Ed.: IEEE-CS Press, 1988, pp. pp. 176-189.

[41] Shepperd, M., Schofield, C., "Estimating Software Project Effort Using Analogies," *IEEE Transactions on Software Enginering*, vol. 23, pp. pp. 736-743, 1997.

[42] Smith, C., *Performance Engineering of Software Systems*: Addison Wesley, 1990.

[43] Standish, G., "CHAOS," 1995.

[44] Sullivan, K., Cai, Y., Hullen, B., Griswold, W., "The Structure and Value of Modularity in Software Design," presented at Proceedings, ESEEC/FSE, 2005.

[45] Valerdi, R., Boehm, B., Reifer, D., "COSYSMO: A Constructive Systems Engineering Cost Model Coming Age," presented at 13th INCOSE Symposium, Crystal City, VA, 2003.

[46] Wittig, G., "Estimating Software Development Effort with Connectionist Models," Monash University Paper 33/95 1995.

[47] Yang, Y., Bhuta, J., Port, D., Boehm, B., "Value-Based Processes for COTS-Based Applications," *Software*, pp. pp. 54-62, 2005.