Web Development:
Estimating Quick-to-Market Software

Developers can use this new sizing metric called Web Objects and an adaptation of the Cocomo II model called WebMo to more accurately estimate Web-based software development effort and duration. Based on work with over 40 projects, these estimation tools are especially useful for quick-to-market development efforts.

Donald J. Reifer, *Reifer Consultants, Inc.*

Over the years, I have become confident in my ability to estimate software costs and schedules. Such confidence comes naturally to someone who has developed many hundreds of estimates. To accomplish this feat in a disciplined and repeatable manner, I have, of course, inserted mature processes, metrics, and estimating models. For predictability and control, I have then invested in data collection and tuned the processes, refined the metrics, and calibrated the models.

During the past few months, I found a simple way to rid myself of my self-assurance: I tried to estimate the cost and schedule for a Web development project. Seemingly, developers use hypertext markup language, Java applets and script, and visual programming languages to generate software for the Web in the blink of an eye. But such projects defeat my processes, defy my models, and make my size metrics obsolete.

By generating thousands of Web objects and making them operational in just a few months, this quickened pace raises pressing questions. For example, as we move to the Web and embrace electronic commerce business models, how do we estimate the software project costs and schedules? How do we examine the breakeven points and justify the investments needed to bring in the e-commerce bounty? More important, how do we adapt our existing processes, size metrics, and models and make them work? In this article, I will try to answer these and other questions about getting a handle on Web software development costs.

Characterizing Web Development Projects

A banner on my Web site recently announced that electronic commerce reached US$3 billion in sales last year. That's a marvelous achievement, but in good news there is also bad. The bad news is that the headline heralds a change to the way we go about developing and deploying software. In summarizing these changes, Table 1 high-

Table 1

Characteristics of Traditional versus Web Development Projects

Characteristics	Traditional development	Evolving Web development
Primary objective	Build quality software products at minimum cost	Bring quality products to market as quickly as possible
Typical project size	Medium to large (hundreds of team members)	Small (3–5 team members)
Typical timeline	10–18 months	3–6 months
Development approach employed*	Classical, requirements-based, phased and/or incremental delivery, use cases, documentation-driven	Rapid application development, gluing building blocks together, prototyping, Rational Unified Process,[1] MBASE[2]
Primary engineering technologies used	Object oriented methods, generators, modern programming languages (C++), CASE tools, and so forth	Component-based methods, fourth- and fifth-generation languages (HTML, Java, and so forth), visualization (motion, animation), among others
Processes employed	Capability Maturity Model-based	Ad hoc
Products developed	Code-based systems, mostly new, some reuse, many external interfaces, often complex applications	Object-based systems, many reusable components (shopping carts, etc.), few external interfaces, relatively simple
People involved	Professional software engineers typically with 5+ years of experience in at least two application domains	Graphic designers, less experienced software engineers (2+ years), new hires right out of school
Estimating technologies used	Analogy using historical data as its basis, SLOC or function point-based models, Work Breakdown Structure (WBS) approach for small projects	Analogy based upon current experience, "design-to-fit" based on available resources, WBS approach for small projects

**Often a function of best processes used in the past by the firm or industry*

lights the move to component-based software development, systematic reuse, and visual technologies. It identifies the move to quick-paced developments and quick-to-market software: Getting their software to market first is the top priority for firms doing business on the Web.

The way we develop software is changing. Rather than develop software from requirements through the waterfall, Web development firms glue together building blocks and reusable components using rapid application development methods and continuous prototyping. While exciting, management of such projects can cause nightmares. Things happen so quickly that it is hard to get a handle on their status and whether they are making suitable progress. Web developments are also hard to estimate. Especially in firms with limited resources, software developers need to better predict the time and effort required to pull off such projects successfully.

In estimation, source-lines-of-code (SLOC) and function-point (FP) estimating models such as Cocomo, Price-S, Slim, and SEER are the mainstay tools used for traditional development.[3–6] The reasons for this are simple; the software development community has

- extensively studed the phenomenology associated with development and the parameters that drive cost;
- developed, validated, and commercialized estimating models that take this phenomenology into account over a period of 20+ years;
- calibrated the models using normalized historical data to accurately predict cost and schedule; and
- developed, refined, and optimized processes that incorporate the models into the planning and control processes firms use to manage their businesses.

Addressing the Estimating Challenges

The estimating community currently has not agreed on how to develop estimates for Web-based projects. The trouble is that the characteristics of the Web-based projects that are listed in Table 1 make it difficult for estimators to adapt and put existing processes, metrics, and models to work operationally. Table 2 highlights the challenges estimators face in Web estimation. For comparison, this table also identifies the more traditional approaches projects use to develop their estimates.

Devising New Size Metrics

Many professionals would like to use the more traditional processes, metrics, and models for estimating Web projects. However, as Table 2 notes, these traditional approaches do not seem to address the challenges facing the field. The major concern estimators face is in estimating size, because size drives most of their models. In response, they need new size metrics to accurately scope the work involved in projects that currently cannot be accurately estimated using SLOC and FPs.

Table 2
Web-Based Estimating Challenges

	Traditional approach	Web-based challenges
Estimating process	Most use analogy supplemented by lessons gleaned from past experience	Job costing done ad hoc based on inputs from the developers (often too optimistic)
Size estimation	Because systems are built to requirements, SLOC or function points are used. Separate models are used for COTS and reused software (generate equivalent new lines that are merged into the estimates).	Applications are built using templates and a variety of Web-based objects (html, applets, components, building blocks). No agreement on a size measure for Web applications has yet been reached within the community.
Effort estimation	Effort is estimated via regression formulas modified by cost drivers (plot project data to develop relationships between variables)	Effort is estimated by breaking the job down into tasks and identifying what is needed to do the work. Little history is available.
Schedule estimation	Schedule is estimated using a cube root relationship with effort	Schedule is estimated based upon analogy. Models typically estmate schedules high because cube root relationship doesn't hold.
Quality estimation	Quality is measurable from internal metrics like defect rates and system properties	Quality is hard to measure. New metrics are needed to assess "quality" of multimedia.
Model calibration	Measurements from past projects are used to calibrate models to improve accuracy[7]	Measurements from past projects are used to identify folklore (too few to be used yet)
"What if" analysis	Estimating models are used to perform *quantitative* "what if" and risk analysis. They are also used to compute return-on-investment (ROI) and cost/benefits.	Most "what if" and risk analysis is *qualitative* because models do not exist. ROI and cost–benefit analysis for electronic commerce applications remain an open challenge.

The first major question is "How do I measure size?" Developers involved in most working Web projects agree that SLOC might not be suitable for early estimation because they are design-based, while FPs might be inappropriate because applications do more than just transform inputs to outputs. In response, dozens of size metrics have emerged for Web development (object points, application points, and multimedia points, for example).[8] Researchers seem to agree only that they cannot agree which of these size metrics is best.

Let's muddy the waters a bit more. Based upon my research, I believe I have developed yet another size metric that resolves the current debate over size metrics. My proposed metric, Web Objects, computes size by considering each of the many elements that comprise the Web application. The metric computes size using Halstead's equation[9] for volume (that is, a proposed measure of size that is language independent and related to the vocabulary used to describe it in terms of *operands* and *operators*) as follows:

$$V^* = N \log_2(n) = (N_1^* + N_2^*) \log_2 (n_1^* + n_2^*)$$

where

N = number of total occurrences of *operands* and *operators*

n = number of distinct *operands* and *operators*

N_1^* = total occurrences of *operand* estimator

N_2^* = total occurrences of *operator* estimators

n_1^* = number of unique *operands* estimator

n_2^* = number of unique *operators* estimators

V^* = volume of work involved represented as Web Objects

Using the predictors listed in Table 3 to compute the number of Web Objects, I have been able to predict a Web application's size repeatably and robustly. These predictors let me consider the elements that contribute to the Web application's size. I can represent each predictor by the unique number of *operands* and *operators* that they contribute to the application. Like function points, the key to developing repeatable predictor counts is a well-defined set of counting conventions. This approach lets me achieve consistency across organizations and resolve conflicts, because size estimates are formulated using such standards.

By their very nature, such counts must clearly separate *operands* from *operators* because the latter represent what we do to an object, not what the object does. Table 3 also provides examples of *operands* and *operators* to clarify what is counted as I develop our Web Object estimates. In addition, Table 4 shows a worksheet I developed to weight the predictors to reflect the actual data I collected on 46 completed projects, which

Table 3

Web Based Predictors of Size

Web Object predictors	Example operands	Example operators
Number of building blocks	Fine grained components (ActiveX, DCOM, OLE, etc.), widgets, …	Create, apply, call, dispatch, interface, terminate, …
Number of COTS components (includes any wrapper code)	Commercial packages, library routines, objects like shopping carts, …	Initiate, terminate, apply, bind, customize, export, wrap, …
Number of multimedia files	Text, video, sound, 3D objects, plug-ins, metatags (no graphics files), …	Create, cut, paste, clear, edit, animate, broadcast, …
Number of object or application points[3] (or others proposed)	# server data tables, # states, # client data tables, percent reuse, …	Transform (inputs to outputs), access, generate, modify,…
Number of xml, sgml, html and query language lines	# lines including links to data attributes	Create, call, browse, link, find, search, retrieve, optimize…
Number of Web components	Applets, agents, guards, …	Create, schedule, dispatch, …
Number of graphics files	Templates, pictures, images, …	Apply, align, import, export, insert, …
Number of scripts (visual language, audio, motion, and so forth)	Macros, containers, …	Create, store, edit, distribute, serialize, generalize, …
Other		

ranged in size from 20 to 100 Web Objects. I developed the worksheet weightings much as Alan Albrecht and John Gaffney used a software engineering approach to validate how well function points fit their data.[10]

To use this worksheet, you must first identify the Web elements that contribute to the job you are estimating. You would start by selecting the applicable items listed in the predictor column. For example, you would enter application or object points, but not both, depending on which of the estimates you had available. If the item you need to size your project does not appear in the column, you would use "other" to account for it. Next, you would determine how each of these elements contributes to the total size by counting the unique number of *operands* (the objects) and *operators* (actions that can done to the object) involved in the application. Then, you would classify each set of *operands* and *operators* in terms of its complexity and enter the number into the appropriate worksheet column. Next, you would apply the complexity ratings and compute the total number in each column. Finally, you would sum the columns to compute the number of Web Objects. (I initially had the number of function points in this list, but I took it out because I could not get the data I needed to estimate this predictor on any of the 46 projects I analyzed.)

Computing size this way offers important advantages:

- The metric used has a solid mathematical foundation.
- It can be easily extended to include new predictors as new elements are introduced for Web applications (video markup languages, motion, sound, and so forth).
- The approach lets us address the unique characteristics of Web-based developments.

The approach also has some disadvantages:

- Some in the metrics community would argue against the use of software science because of its statistical mathematical basis. My research shows that the literature is full of arguments for and against use of the technique. Independent of who is right, use of software science as a basis is controversial.
- The planning and data collection costs rise as you add predictors to handle new elements. Temperance is needed or it might take you longer to estimate than develop your Web applications.
- Web Object counts tend to be very sensitive to counting conventions. For example, the size and complexity of multimedia components can vary from single objects such as buttons to long video sequences.

To address these challenges, I am currently documenting counting conventions as I collect and analyze project data. Based on my current rate of progress, I am at least a year away from publishing definitive guidelines for this endeavor.

Table 4

Web Object Calculation Worksheet

Web Object predictors	Complexity-weight Low	Average	High
Number of building blocks	1	2	4
Number of COTS components	2	4	6
Number of multimedia files	1	2	4
Number of object or application points	*	*	*
Number of Web components	2	4	6
Number of xml, sgml, html, and query lines	3	5	8
Number of graphics files	2	4	6
Number of scripts	1	2	3
Other	2	4	6

** We assume weights have already been applied (otherwise would result in double counting).*

Formulating New Models

Having a metric for size is just the first step in developing a model that accurately estimates Web development costs and schedule. The mathematical issues associated with predicting effort and duration must be reconciled before such models launch. The major issues revolve around the form of the mathematical equations and the schedule law. Analysis of data reveals that the equations can be expressed as regressions. However, the traditional cube-root relationship between effort and duration in most estimation models does not seem to accurately predict Web development schedules. Barry Boehm at the University of Southern California is looking at using a square-root relationship to more precisely represent the relationship. Larry Putnam has published several papers arguing that such relationships can be represented by a fourth power trade-off law.[11]

My initial data analysis reveals that the square-root relationship seems to exist for projects smaller than 100 Web Objects. For larger projects, the cube-root relationship seems to produce a better fit. Such a variable schedule law relationship is expected because software science scales effort mathematically as a function of length and volume to predict duration. As I continue gathering additional project data, I will look at how the equations scale for different-sized projects.

Now that these mathematical issues are out of the way, let's take a good look at the model that I propose for estimating Web development costs. I call the new model WebMo, the Web Model, because it is an extension of the Cocomo II early design model. I developed the model using a mix of expert judgment and data from 46 projects using regression analysis. Its mathematical formulation rests upon parameters from both the Cocomo II and SoftCost-OO software cost-estimating models.[12] I have computed exponents for both equations by segmenting the estimating trends into the following three domains: Web-based electronic commerce, financial and trading applications, and information utilities. Figure 1 shows the WebMo model for estimating equations for effort (in person-months) and duration (in calendar months).

As noted, the resulting effort estimation model has nine cost drivers and fixed power laws. I deviated from the Cocomo II formulation because I observed colinearity between cost drivers as I performed a regression analysis. In the duration estimation model, I switched from a cube- to square-root relationship with effort based upon built-in scaling rules to improve the accuracy of my estimates.

Table 5 summarizes the values for all of the model's parameters except the cost drivers. Tables 6 through 9 provide a quick rating scheme for each of my cost drivers, while Table 10 provides the values for each of the settings. The cost drivers include

Figure 1. WebMo model.

$$\text{Effort} = A\prod_{i=1}^{9} cd_i(\text{Size})^{P1} \qquad \text{Duration} = B(\text{Effort})^{P2}$$

Where: A and B are constants

P1 and P2 are power laws

cd_i are cost drivers

Size is the number of Web Objects

Table 5

Web Development Model Parameter Values

	A	B	P1	P2
Web-based electronic commerce	2.3	2.0	1.05	*
Financial/trading applications	2.7	2.2	1.05	*
Business-to-business applications	2.0	1.5	1.00	*
Web-based information utilities	2.1	2.0	1.00	*

** Either 0.5 or 0.33 depending on the scaling (>40 Web Objects)*

- RCPX: product reliability and complexity (product attributes);
- PDIF: platform difficulty (volatility of platform and network servers);
- PERS: personnel capability (skills, knowledge and abilities of the workforce);
- PREX: personnel experience (the breadth and depth of the team's experience);
- FCIL: facilities (tools, equipment and colocated facilities);
- SCED: schedule (degree of risk taken to shorten duration);

Table 6

Rating Scale for RCPX and PDIF

Driver	VL	L	N	H	VH
CPLX	Client side No distribution Invocation Simple math Simple I/O Limited data Reliability not a factor	Client–server Limited distribution Adaptation Standard math File management Some files Easy to recover from losses	Client–server Fully distributed Integration Statistics DBMS Databases Moderate recovery goal	Client–server Wide distribution Synchronization Math intensive Distributed database Virtual database High financial loss due to error	Client–server Full distribution Collaborative Soft real-time Persistent database Virtual database Errors cause risk to life
PDIF	Rare changes to platform Speedy net Best possible connectivity No computer resource limitations	Few changes to platform Fast net service Rare loss of connectivity Few computer resource limitations	Platform stable Acceptable net performance Acceptable connectivity Must watch use of computer resources	Frequent changes to platform Slow network performance Poor connectivity Lack of computer resources causing problems	Platform not stable Unacceptable performance Unacceptable connectivity Timing and storage impacts

Table 7

Rating Scale for PERS and PREX

Driver	VL	L	N	H	VH
PERS	15th percentile Major delays due to turnover	35th percentile Minor delays due turnover	55th percentile Few delays due to turnover	75th percentile Infrequent delays due to turnover	90th percentile No delays due to turnover
PREX	$\leq$ 2 months average tool, language, platform, and applications experience	$\leq$ 6 months average tool, language, platform, and applications experience	$\leq$ 1 year average tool, language, platform, and applications experience	$\leq$ 3 years average tool, language, platform, and applications experience	$\leq$ 6 years average tool, language, platform, and applications experience

Table 8

Rating Scale for FCIL, SCED, and RUSE

Driver	VL	L	N	H	VH
FCIL	International Phone/fax Ad hoc methods Language tools Basically no collaboration	Multisite Phone/email Phase-dependent methods Basic CASE Limited collaboration	One complex LAN Life-cycle methods Tools support methods Integrated product teams	Same building WAN Integrated methods Integrated toolset Collaborative teams	Colocated Broadband State-of-the-art method Integrated toolset that supports collaboration
SCED	Must shorten 75% nominal	Must shorten 85% nominal	Keep as is Nominal	Can relax 130% nominal	Can extend 160% nominal
RUSE	Not used	Not used	Not used	Not used	Not used

Table 9

Rating Scale for TEAM and PEFF

Driver	VL	L	N	H	VH
TEAM	No shared vision Stakeholders do not work to meet each others' goals Teamwork limited	Little shared vision Stakeholders talk and build respect for each other Some teamwork	Some shared vision Stakeholders pull together and work joint goals Basic teamwork	Considerable shared vision Stakeholders respect each others' goals and collaborate Integrated teams	Extensive shared vision Stakeholders pull together and focus on goals Seamless teams
PEFF	Totally ad hoc, confused process Reliance on heroes to get job done	Project-based processes Reliance on management leadership to meet goals	Streamlined process tailored for the job Reliance on process for guidance	Efficient process matched to job Process is how engineers do their work	Effective process that meets goals Everyone uses and believes in the proces

- RUSE: reuse (degree of reuse planned and executed);
- TEAM: teamwork (the ability to work synergistically as a team); and
- PEFF: process efficiency (streamlined for the business)

I do not convert my size estimates from Web Objects to SLOC in the equation for effort. Rather, I used my initial data set to calibrate directly the relationships that existed between size in Web Objects and effort in person months. Another open issue that I will research is whether it makes sense to develop back-firing ratios from Web Objects to and from SLOC as the function point community has done.

Several of these ratings differ greatly from the original models. For example, SCED is flat when extended past its estimated duration in Cocomo II. But, my data shows that schedule adheres to a bell-shaped curve, consistent with the similar factor in the SoftCost-OO model. In other words, it costs more to both compress and elongate the nominal estimated duration.

Table 10

Values for Cost Drivers

Driver	VL	L	N	H	VH
RCPX	0.63	0.85	1.00	1.30	1.67
PDIF*	0.75	0.87	1.00	1.21	1.41
PERS	1.55	1.35	1.00	0.75	0.58
PREX	1.35	1.19	1.00	0.87	0.71
FCIL	1.35	1.13	1.00	0.85	0.68
SCED*	1.35	1.15	1.00	1.05	1.10
TEAM	1.45	1.31	1.00	0.75	0.62
PEFF	1.35	1.20	1.00	0.85	0.65
RUSE	Not rated	Not rated	1.00	Not rated	Not rated

** Significant differences from original model observed*

About the Author

Donald J. Reifer is a teacher, change agent, consultant, contributor to the fields of software engineering and management, and author of *Tutorial on Software Management, Fifth Edition.* He is president of Reifer Consultants, Inc., and serves as a visiting associate at the Center for Software Engineering at the University of Southern California. He is also a member of the *IEEE Software* editorial board and editor of its Manager column. Contact him at d.reifer@ieee.org.

I plan to address the numerous open issues by gathering and analyzing more data from completed Web development projects. As I've discussed, the most important of these is developing consistent counting conventions for Web Objects. I will work with clients to collect the data I need to both resolve counting issues and improve the model's estimating accuracy. Currently, I can estimate Web development projects in my database within 30%, 60% of the time by segmenting the databases by the application domains in Table 5. But the accuracy of the model must be improved for commercialization. I want to improve this accuracy incrementally using project data to refine ratings developed through expert opinion. I aim to gather data on at least another 30 projects so that I can improve the model's accuracy for both effort and schedule to within 20% of actuals at least 80% of the time. This will take me about a year to accomplish.

References

1. P. Kruchten, *The Rational Unified Process*, Addison-Wesley, Reading, Mass., 1999.
2. B.W. Boehm, "Transitioning to the CMMI via MBASE," *SoCal SPIN Meeting Presentation*, Dept. Computer Science, Univ. Southern California, Los Angeles, 2000.
3. B.W. Boehm et al., *Software Cost Estimation with Cocomo II*, Prentice-Hall, Upper Saddle River, N.J., 2000.
4. R.E. Park, "The Central Equations of the PRICE Software Cost Model," *Proc. Fourth Cocomo User's Group*, 1988.
5. L.H. Putnam and W. Myers, *Measures of Excellence*, Prentice-Hall, Upper Saddle River, N.J., 1992.
6. *Parametric Cost Estimating Handbook*, US Dept. of Defense, Washington D.C., 1995.
7. D.V. Ferens and D.S. Christensen, "Does Calibration Improve Prediction Accuracy?," *CrossTalk*, Vol. 13, No. 4, Apr. 2000, pp. 14–17.
8. A.J.C. Cowderoy, "Size and Quality Measures for Multimedia and Web-Site Production," *Proc. 14th Int'l Cocomo Forum*, 1999.
9. M.H. Halstead, *Elements of Software Science*, Elsevier North Holland, Dordrecht, The Netherlands, 1977.
10. A.J. Albrecht and J.E. Gaffney, Jr., "Software Function Points, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Trans. Software Eng.*, Vol. SE-9, No. 6, 1983, pp. 639–47.
11. L.H. Putnam and D.T. Putnam, "A Data Verification of the Software Fourth Power Trade-off Law," *Proc. Int'l Soc. Parametric Analysts Conf.*, 1984.
12. D.J. Reifer, *SoftCost-OO Reference Manual*, Reifer Consultants, Inc., Torrance, Calif., 1993.