

MODELING RESOURCES

There are a variety of reasons for modeling resources. We may wish to do an initial prediction of resources, i.e., based upon a set of factors that can be estimated about a project, we can try to predict total effort, cost, staffing, computer use, etc.

We can develop descriptive models of the development pattern, that is show how and when resources get allocated. This provides insights into what is going on, shows how different parameters change the resource allocation pattern, helps us evaluate the effects of various techniques and methods so we can better engineer software, and provides baselines from which to plan future developments.

We can use resource modeling to help predict the resources to be used in the next phase based upon the current phase. This is a more detailed application of the initial prediction activity mentioned above. That is, given where we are in the project, the model should tell us what should happen next. If it doesn't, why not; is it a sign of trouble, etc.?

Lastly we can use the model for validation of our understanding of the environment, i.e., does the model explain our behavior and environment, do the factors (parameters) agree with our environmental factors, and are they calibrated correctly?

A good model explains our behavior and the development environment. Its parameters are calculable from known or easy to estimate data e.g., maximum staffing, time to delivery, complexity of the software, number of lines of code, number of modules, number of I/O formats, type of software, amount of old/new software (design, code, specification). The parameters describe and can be calibrated for our environment, redundancy checks and risk analysis factors are available and when the model doesn't work, we can gain insight into why and what is different about the environment than was expected or how to improve the model or better calibrate it to the environment.

There are a variety of resource models that provide descriptions of such resources as effort, staffing, cost, computer use, and calendar time. These models can be categorized with respect to type of formula used for total effort. Models can be distinguished as single variable vs. multi-variable, empirically based or theoretically based. Some describe the dynamic allocation of resources while others give a single overall figure. Some are defined at the macro level, based on high level parameters while others are defined at the micro level.

Early on Barry Boehm collected data to provide some insights into where the effort resources were being spent. He surveyed several projects and collected data within his own organization, TRW. The results of this study are given in Figure 7.1.

Based upon this data, the 40-20-40 rule-of-thumb came into practice, i.e. 40% of the resources should be spent on analysis and design, 20% on code, and 40% on checkout and test. However, this is a misinterpretation of the data. The data represents phase data data, rather than actual resource

expenditures, i.e., a better interpretation of the data might be that 40% into the resources the design completion milestone should be reached, and 60% into the resources, the code completion data should be reached. This provides data for management as to when various milestones are appropriate.

Figure 7.2 provides similar data from IBM and the SEL at NASA Goddard Space Flight Center as a basis for comparison. In this data, another category, other, is present. Other represents resource effort that is not associated with a particular phase of the life cycle, e.g. training, meetings, etc.

Where Does the Software Effort Go?			
	Analysis and Design	Coding and Auditing	Checkout and Test
Sage	39%	14%	47%
NTDS	30	20	50
Gemini	36	17	47
Saturn V	32	24	44
S/360	33	17	50
TRW Survey	46	20	34

Figure 7.1 PROFILE DATA EFFORT BY PHASE

Examining the first three columns, it should be noticed that the data from the three environments are different, i.e. the design effort varies from 40% to 35% to 20%, code effort varies from 20% to 30% to 45%, and test effort varies from 40% to 25% to 28%. Only two organizations report effort in the other category. There are several explanations for this. First, each of the organizations may allocate different activities to the phases, e.g., in the SEL, the life cycle starts with functional specification analysis and design, which are what analysis and design mean. Second, each organization may have a different point at which they define their milestones, e.g., the SEL defines the analysis and design phase to end at PDR (preliminary design review). Clearly, the interpretation of the data is highly dependent upon the environmental characteristics in which it was collected. That makes it very difficult to compare across environments.

It should also be noted that there are two sets of data for the SEL environment. That is because two types of data are collected, **phase date** data and **activity data**. The phase date data represents the total amount of effort expended each week by each staff member within the dates established for the phase. The activity data is the total amount of effort expended on each activity by each staff member each week. These figures are not the same because staff member perform activities in phases other than

the named phase they are in. For example, coding may begin on some components before the design end milestone occurs. Also, design may continue to occur after the design end milestone because the design may not be complete or some redesign may be necessary.

Activity patterns do not necessarily follow date patterns. Both sets of data are important. The activity data provides insight into the actual effort expended in each activity. The phase data provides insight into how management should allocate milestones.

The other category in the SEL activity column is 27%. This says that project staff spend 27% of their time performing activities chargeable to the project but not directly associated with a project activity, i.e., travel, education, meetings, etc. Actually the figure for the SEL is estimated as low compared to what that figure might be in other organizations. It is worth noting that if one estimates solely on the basis of activities, the estimation would be 27% low in the SEL.

	TRW	IBM	Phase	SEL Activity
Design	40	35	20	21
Code	20	30	45	28
Checkout/Test	40	25	28	23
Other		10	5	27

Figure 7.2 Comparison of Resource Effort Data

Static Single Variable Resource Models

The most basic model is the static single variable resource model. The effort equation is based on a single variable, usually a measure of size. There are several possible variations on the basic equations:

$$\text{Effort} = A * \text{size} + C$$

$$\text{Effort} = A * \text{size}^B$$

$$\text{Effort} = A * \text{size}^B + C$$

where A, B and C are constants determined by regression analysis on historical data. In this relationship effort may be measured in staff hours, weeks, months, or years and size may be measured in a variety of forms, e.g., lines of code, modules, I/O formats.

Since size is the typical base measurement, we need mechanisms for estimating the size of a system. The most traditional approaches, suggested by Ray Wolverton, are top-down estimating, similarities and differences estimating, ratio estimating, standards estimating, bottom-up estimating, or a combination of two or more basic methods.

Top-down estimating is based upon comparing the new system with prior systems of a similar type and modifying the estimate based upon intuition. It is used as to get a ballpark figure for the system cost

and requires historical data on prior projects of a similar type. **Similarities and differences estimating** subdivides the project into units where similarities to and differences from known units are available activities. It can be viewed as a more detailed version of top down estimating. **Ratio estimating** uses more detailed data in the form of anticipated number of lines of object code and type and complexity of component, which are assumed to be invariant, to develop ratios of cost per instruction. It is usually done to provide more detailed estimates on smaller system units. **Standards estimating** relies on a table of standard costs for specific functions. As with all the other approaches it assumes historical data that can be packaged in a form specifically usable for size estimation. The last approach is **bottom-up estimating**, which can be done when there is little information available from prior systems of a similar kind. It relies on breaking the task down into relatively small work units until each of the units is easy to estimate. In most situations a combination of these approaches is used.

Typical to any estimation approach is to gather data from more than one source, i.e., get the estimates from three people, each of which may have used a different approach. Ideally, the estimator should ask for three numbers, the maximum possible size, the minimal possible size and the expected size. This approach has the benefits of eliminating potential optimism or pessimism and providing a set of bounds on the estimate.

An excellent example of a static single variable resource model was proposed by Claude Walston and Chuck Felix at IBM's old Federal Systems Division. They developed a relationship between size and effort using the equation

$$E = A * SIZE^B$$

Their goal was to measure the rate of production of lines of code by projects, influenced by a number of product conditions and requirements. Based upon a data base of 60 projects, ranging in size from 4K to 467K source lines of code, from 12 to 11,758 staff months, over a variety of task types, applications and programming language, they used a regression equation to fit the data, yielding a relationship

$$E = 5.2L^{.91}$$

where E = effort in staff months, and L = lines of code in thousands.

Besides the basic effort/ size relationship, they examined the relationship between a variety of other factors available in the data base, e.g., pages of documentation, project duration, and average staff size.

The full set of derived equations are:

$$E = 5.2L^{.91}$$

$$DOC = 49L^{1.01}$$

$$D = 4.1L^{.36}$$

$$D = 2.47E^{.35}$$

$$S = .54E^{.6}$$

where E = effort in staff months, L = lines of code in thousands, DOC = documentation in pages, D = project duration in calendar months, and S = average staff size = E/D

To provide better insight into influence of a variety of variable and aid in the estimation process they examined the effect of 68 variables with regard to their effect on productivity. Of the 68 variables

examined, 29 showed significantly high correlation with productivity. These 29 variables, along with their effect are given in table 7.3

They used these 29 variables for estimating effort by combining them into a productivity index,

$$I = \sum_{i=1}^{29} W_i X_i$$

where W_i = question weight ($1/2 \log_{10}$ (ratio of productivity change for question i)) and X_i = question response (+1, 0, or -1), depending on whether the responses indicate increased, nominal or decreased productivity.

The productivity index is used to adjust the initial estimator from the baseline equation by explaining deviations from the norm.

Question or Variable	Response Group			Change (DSL/MM)
	<normal	normal	>normal	
Customer interface complexity	500	295	124	376
User participation in the definition of requirements	491	267	205	286
Customer originated program design changes	297		196	101
Customer experience with the application area of the project	313	340	206	112
Overall personnel experience and qualifications	132	257	410	278
Percentage of programmers doing development who participated in design of functional specifications	153	242	391	238

Previous experience with operational computer	minimal 146	average 270	extensive 312	166
Previous experience with programming languages	minimal 122	average 225	extensive 385	263
Previous experience with application of similar or greater size and complexity	minimal 146	average 221	extensive 410	264
Ratio of average staff size to duration (people/month)	<0.5 305	0.5-0.9 310	>0.9 173	132
Hardware under concurrent development	no 297		yes 177	120
Development computer access, open under special request	0% 226	1-25% 274	>25% 357	131
Development computer access, closed	0-10% 303	11-85% 251	>85% 170	133
Classified security environment for computer and 25% of programs and data	no 289		yes 156	133
Structured programming	0-33% 169	34-66% -	66% 301	132
Design and code inspections	0-33% 220	34-66% 300	>66% 339	119
Top down development	0-33% 196	34-66% 237	>66% 321	125
Chief programmer team usage	0-33% 219	34-66% -	>66% 408	189

Overall complexity of code developed	<average 314		>average 185	129
Complexity of application processing	<average 349	average 345	>average 168	181
Complexity of program flow	<average 289	average 299	>average 209	80
Overall constraints on program design	minimal 293	average 286	severe 166	107
Program design constraints on main storage	minimal 391	average 277	severe 193	193
Program design constraints on timing	minimal 303	average 317	severe 171	132
Code for real-time or interactive operation, or executing under severe timing constraint	<10% 279	10-40% 337	>40% 203	76
Percentage of code for delivery	0-90% 159	91-99% 327	100% 265	106
Code classified as non-mathematical application and I/O formatting programs	0-33% 188	34-66% 311	67-100% 267	79
Number of classes of items in the data base per 1000 lines of code	0-15 334	16-80 243	>80 193	141
Number of pages of delivered documentation per 1000 lines of delivered code	0-32 320	33-88 252	>88 195	125

Figure 7.3 VARIABLES THAT CORRELATE SIGNIFICANTLY WITH PROGRAMMING PRODUCTIVITY AT IBM/FSD

In an attempt to understand the SEL environment and to validate the effort equation proposed by Walston and Felix, the SEL studied the equation. One question was, do the W/F equations hold in other environments?

As part of the study, several other variables were included in the study because the represented data that was relevant or available in the environment. For example since a fair amount of code is reused from prior systems, and new size variable, DL = number of developed source lines of code was considered. It was estimated that the cost of reusing old code was about 20% of the cost of new code. (There is a cost since the old code has to be understood and integrated and tested into the new system. Thus DL = new code + 20% reused code. Since modules (FORTRAN subroutines were a possible mechanism for estimation, a variable M = total number of modules was added, as well as the variable DM = total number of developed modules, i.e., all new or more than 20% new. In order to understand the effect of reuse on productivity, the variables P = productivity = L/E , $RDTODL$ = ratio of developed to total lines of code (which takes on the value .2 if all the code is old and 1 if all the code is new), and $RDTODM$ = ratio of developed to total modules.

The SEL data considered in the study consisted of 15 projects dealing with ground support software, ranging in size from 2K to 101K developed source lines (DL), with a duration ranging from 4.6 to 17.4 months, effort ranging from 5 to 138 staff months, average staff size from 1 to 8 people, productivity, in terms of lines of source code per staff month, from 413 to 1068 developed source lines/staff month with an average of 668 developed source lines/staff month. The data covers design through acceptance test and includes the effort of managers, programmers and support staff.

The figures show that the SEL data is about one standard error of estimate below the estimate by the Walston/Felix equations. This is not surprising since the IBM/FSD data base was filled with one of a kind projects, which the organization was developing to keep up with the state of the art. On the other hand, the SEL was developing projects of a common application which were essentially of a similar design. There were no new state of the art developments there. The actual fit to the SEL data is $E = 1.38L^{.93}$, with a standard error of estimate of $SE = XX$ and a coefficient of determination $R^2 = .XX$. This means the SEL generated equation was much more appropriate for the SEL environment. However, since the SEL type projects were at the lower end of the complexity scale for the IBM data base and the SEL data was on standard estimate of error away from the norm, it probably also means the Walston/Felix equation is reasonable, but clearly not a sufficiently good estimator for SEL projects.

When the variable DL is used, it generates an equation of the form $E = 1.48DL^{.98}$ which is almost a straight line. The standard error of estimate improves to $SE = 1.29$ and the coefficient of determination improves to $R^2 = .96$. Clearly, DL gives a better estimate than L , assuming DL can be approximated reasonably well in the environment.

The variable DM does not give as good an estimate, yielding an equation $E = .185DM^{1.05}$ with a standard error of estimate of $SE = 11.9$ and a coefficient of determination $R^2 = .88$.

The duration versus effort equation for the SEL data is $D = 4.39E^{.26}$ with a standard error of estimate of $SE = 1.37$ and a coefficient of determination $R^2 = .52$, as opposed to $D = 2.47E^{.35}$ for the Walston Felix data, with a standard error of estimate of $SE = 1.35$ and a coefficient of determination $R^2 = .59$. The duration vs. lines of code equation for the SEL was $D = 4.55L^{.26}$ with a standard error of estimate of $SE = 1.36$ and a coefficient of determination $R^2 = .55$ as opposed to $D = 4.1L^{.35}$ for the Walston Felix data, with a standard error of estimate of $SE = 1.72$ and a coefficient of determination $R^2 = .41$. This implies that projects in the SEL take longer for the amount of effort expended and lines of code developed. This is another reason why the SEL projects should be cheaper per line of code.

The documentation vs. lines of code equation for the SEL was $DOC = 30.4L^{.9}$ with a standard error of estimate of $SE = 1.41$ and a coefficient of determination $R^2 = .92$. The Walston/Felix equation is $DOC = 49L^{1.01}$ with a standard error of estimate of $SE = 2.63$ and a coefficient of determination $R^2 = .62$. But these two equations are really representing different things. The SEL equation does not count source lines of code as part of the documentation but the Walston/Felix equation does.

With regard to staff size vs. effort, the SEL equation is $S = .24E^{.73}$ with a standard error of estimate of $SE = 1.38$ and a coefficient of determination $R^2 = .89$. The Walston/Felix equation is $S = 54E^{.6}$ with a standard error of estimate of $SE = 1.56$ and a coefficient of determination $R^2 = .79$. Again the SEL data is approximately one standard error of estimate below the Walston/Felix data, implying that the SEL uses a smaller staff for a similar size project. This coupled with the fact that the SEL uses more calendar time helps explain the lower costs.

Finally, when comparing productivity with the ratio of developed to total lines of code, the SEL data yields $P = 704 RDTODL^{-.96}$ with a standard error of estimate of $SE = 1.38$ and a coefficient of determination $R^2 = -.59$. This implies that it is cost effective to reuse code.

The conclusion drawn by the authors in the validation study was that the Walston/Felix equation was a very broad but probably reasonable equation for defining software projects in general but the error in estimation was much too large to be considered as a reasonable approximator for the SEL. However, since there was sufficient data in the SEL to create an SEL specific set of equations, and the data fit these equations reasonably well, these equations represented a much better set of estimation equations than the Walston/Felix equations.

Static Adjusted Baseline Equations

In the static adjusted baseline type resource model, the static single variable effort equation acts as a baseline equation, e.g., $effort = A * size^B$. This provides a basic estimate of effort. This initial estimate is adjusted by a set of multipliers that attempt to incorporate the effect of important product and process attributes. For example, if the initial estimate is $E = 100$ staff months and the complexity of the job is rated higher than normal, a multiplier 1.1 is associated with it, yielding an adjusted estimate of 110 staff months.

An example of a static adjusted baseline model is COCOMO. The COCOMO cost model offers three different baseline equations, depending upon the complexity of the project. Each of the equations is of the form $\text{Effort} = A * \text{Size}^B$, where A and B are determined by regression analysis.

The COCOMO model has three levels of detail: **basic**, which is used for quick, early, approximate estimates of software cost and schedule, but its accuracy is limited due to not using detailed data, **intermediate**, which is used for better estimates of cost and schedule, because it considers software project environment factors in terms of their aggregate impact on the project parameters, and **detailed**, which is used for even better estimates, because it accounts for the influence of the software project environment factors on individual project phases. We will concentrate our discussions on the basic and intermediate levels.

Boehm, TRW

The modes of the COCOMO model are **organic mode**, which is appropriate for small, stable projects, similar to the ones in the SEL, **embedded mode**, which is appropriate for large projects with tight constraints, that require some degree of innovation and have a complex software interface, and **semi-detached mode**, which is appropriate for projects that fall in between the above two categories.

The formulas for the basic model for each of these modes are given in Figure 7.4. The required effort (**E**) to develop the software system as a function of source size (**S**), where E is expressed in Person-Months and S is expressed in KLOC. The project duration (**TDEV**) as a function of effort (**E**) where TDEV is expressed in calendar months, and E in Person-Months).

BASIC COCOMO MODEL

Mode	Effort	Schedule
Organic	$E = 2.4 * (S^{1.05})$	$TDEV = 2.5 * (E^{0.38})$
Semi-detached	$E = 3.0 * (S^{1.12})$	$TDEV = 2.5 * (E^{0.35})$
Embedded	$E = 3.6 * S^{1.20}$	$TDEV = 2.5 * (E^{0.32})$

Figure 7.4 The Effort and Schedule Formulas of the Basic Cocomo Model

The source size (**S**) is expressed in KLOC, i.e. thousands of delivered lines of code, i.e. , the source size of the delivered software (which does *not* include the size of test drivers or other temporary code). If code is reused, then the following formula should be used for determining the equivalent software source size S_e , for use in the COCOMO model:

$$S_e = S_n + \frac{a}{u} * S_u$$

where S_n is the source size of the new code, S_u is the source size of the reused code, and a is determined by the formula:

$$a = 0.4 * D + 0.3 * C + 0.3 * I$$

based on the percentage of effort required to adapt the reused design (D) and code (C), as well as the percentage of effort required to integrate the modified code (I).

The duration, or time to development, **TDEV**, starts when the project enters the product design phase (successful completion of a software requirements review) and ends at the end of software testing (successful completion of a software acceptance review). The effort, **E**, covers management and documentation efforts, but not activities such as training, installation planning, etc. COCOMO assumes that the requirements specification is not substantially changed after the end of the requirements phase. Person-months can be transformed to person-days by multiplying by 19, and to person-hours by multiplying by 152.

The intermediate COCOMO is an extension of the basic COCOMO model, which uses only one predictor variable, the KLOC variable. However, the intermediate COCOMO uses 15 more predictor variables called "cost drivers." The manager assigns a value to each cost driver from the range: Very low, Low, Nominal, High, Very high, Extra high. For each of the above values, a numerical value corresponds, which varies with the cost drivers. The ratings for the COCOMO cost drivers are given in Figure 7.5.

The manager assigns a value to each cost driver according to the characteristics of the specific software project. The numerical values that correspond to the manager assigned values for the 15 cost drivers are multiplied. The resulting value **I** is the multiplier that we use in the intermediate COCOMO formulas for obtaining the effort estimates. This is similar to the productivity index suggested by Walston and Feliz but simpler to apply.

Thus $I = \text{RELY} * \text{DATA} * \text{CPLX} * \text{TIME} * \text{STOR} * \text{VIRT} * \text{TURN} * \text{ACAP} * \text{AEXP} * \text{PCAP} * \text{VEXP} * \text{LEXP} * \text{MODP} * \text{TOOL} * \text{SCED}$

The formulas for the intermediate model for each of the modes are given in Figure 7.6. Note that although the effort estimation formulas for the intermediate model are different from those used for the basic model, the schedule estimation formulas are the same. The required effort to develop the software system (E) as a function of the nominal effort (E_{nom}), where E and E_{nom} are expressed in Person-Months, and S in KLOC is

$$MM = MM_{nom} * I$$

INTERMEDIATE COCOMO MODEL

MODE	EFFORT	SCHEDULE
------	--------	----------

Organic	$Enom = 3.2 * (S^{1.05})$	$TDEV = 2.5 * (E^{0.38})$
Semi-detached	$Enom = 3.8 * (S^{1.12})$	$TDEV = 2.5 * (E^{0.35})$
Embedded	$Enom = 2.8 * (S^{1.20})$	$TDEV = 2.5 * (E^{0.32})$

Figure 7.6 The Effort and Schedule formulas of the Intermediate COCOMO Model

The number of months estimated for software development (TDEV), where TDEV is expressed in calendar months, and E in Person-Months.

The major limitation to a general model like COCOMO with regard to the accuracy of resource estimation is the organizations ability to understand and estimate the appropriate parameters required by the model and the relevance of the model equations, to the organization's particular environment. Remember this is an empirical model, based upon data that has been accumulated from a variety of environments and thus the baseline equations may not fit the environment being modeled. Secondly, the parameters or cost drivers may be difficult to estimate because it may be hard to determine what nominal means relative to the set of environments the data represents. In fact, these particular cost drivers may not even be relevant to the environment. On the other hand, the benefits to a global model such as COCOMO are that (1) it provides a world view that allows us to understand what other organizations consider reasonable with regard to resources for this project, (2) it requires no local data base.

The Meta-Model

It would seem logical that we can achieve greater accuracy when we use data gathered from the local environment in building the resource model, however this requires a data base of past project performance within the organization. Bailey and Basili proposed a mechanism for building an empirical model relative to the data within the local environment.

The model is based upon the assumptions that (1) each software development environment is different, (2) there are factors that reflect the organizational environment rather than the project, and (3) there are other factors that reflect the differences in the projects. The approach is to (1) compute the background equation (which should reflect those factors that are constant across the environment), (2) analyze the factors available to explain the difference between actual effort and effort as predicted by the background (which represent the actual cost drivers within the current environment) and the (3) use the model to predict the effort for new project

To perform step 1, we must first compute the background equation. This requires that we pick some size measure (S). This can be any measure of size for which we have data and which is easiest for the organization to estimate. Possible size measures include lines of code (total delivered source lines, executable statements, developed sources lines, etc.) modules (new, total, with data blocks, etc.),

function points, number of input output formats, etc., We must pick a measure of effort, i.e. hours, days, months, years, etc. Then we must choose a background relationship for effort (E) and size (S). Possible background equations include

$$E = aS + b$$

$$E = aS^b$$

$$E = aS^b + c$$

Then we have to calculate the best fit based on some criteria such as minimizing the absolute error (least squares) or minimizing the percent error using any standard numerical analysis package.

Based upon 18 data points in the SEL data base the following models were generated:

Linear Fit

$$E = 1.37 DL + 3.48 \quad \text{see} = 29.3\%$$

(minimizing absolute error) see = 9.54mm

Log Transformation Fit

$$E = 1.991 DL^{.914} \quad \text{see} = 27.8\%$$

(minimizing absolute error) see = 9.11mm

Direct Fit

$$E = 1.545 DL^{.967} \quad \text{see} = 24.7\%$$

(minimizing percent error) see = 9.44mm

Direct Fit (with constant)

$$E = .5766 DL^{1.2} + 3.75 \quad \text{see} = 20.8\%$$

(minimizing percent error) see = 11.68mm

In step 2 of the meta-model we analyze the factors available to explain the difference between actual effort and effort as predicted by the background equation for the available data. This requires that we choose a set of factors, group the factors based upon relevance and experience, choose a few factor groups based upon intuition and correlations with productivity and the difference between actual and predicted effort (this is necessary since we may have too many factors), and run a forward multiple regression that adds a factor at a time. There is no rule to judge how many factors to use but a rule of thumb is to limit the number of factors to either 10% of the number of data points or to stop adding factors when the new factor account for minimal improvement in the explanation of the difference (R^2). Finally on should iterate through the various sub-steps of step 2 to try to improve the explanation of the difference.

In applying this approach to the SEL data base, the authors analyzed about 100 factors, based upon the Walston/Felix factors, the COCOMO factors, a set of factors that appeared to be relevant from the Basili/Reiter controlled experiment, and a set of factors available in the SEL data base. These factors were grouped into three major groups: total methodology, cumulative complexity, and cumulative experience.

Total methodology included a subjective evaluation of the process conformance in the use of tree charts, top down design, design formalisms, formal documentation, code reading, chief programmer teams, formal test plans, unit development folders, and formal training. Cumulative complexity consisted of a subjective evaluation of customer interface complexity, customer-initiated program design changes, application process complexity, program flow complexity, internal communication complexity, external communication complexity, and data base complexity. Cumulative experience consisted of a subjective evaluation of programmer qualifications, programmer experience with the machine, programmer experience with language, programmer experience with application, and the team previously working together on same type problem.

In step 3 of the meta-model, we use the model to predict the effort for the new project by estimating size, using the background equation to predict standard effort, estimating the values of the factors used in the regression analysis, computing the difference this project should exhibit based upon the values from the factors used in the multiple regression equation, and applying that difference to the standard effort to compute the improved effort estimate.

The authors applied the meta-model by building a 17 project model and using it to estimate the eighteenth project. They chose

S = DL (developed lines of code),
 E = staff months (sm),
 the background relationship $E = aS^b + C$
 minimizing the percent error

This yielded a background equation (based upon the first 17 data points) of

$$E_s = .72DL^{1.17} + 3.4$$

with a standard error of estimate = 1.254

For project 18 they estimated 101,000 developed lines yielding

$$E_s = 163sm \text{ with an error interval of } (130,204)$$

Using the factor groups discussed earlier, the forward regression program brought in two factors: total methodology (meth) and cumulative complexity (cmplx). They applied the multiple regression routine to compute the effort ratio (ER): using the factor meth,

$$ER = -.036meth + 1.0 = -.224$$

yielding an improved estimate $E_i = E_s / 1.224 = 133sm$ with error interval of (115,154), and using meth and cmplx to compute the effort ratio:

$$E_R = -.036meth + .006cmplx + .86 = -.166$$

yielding an improved estimate $E_i = E_s / 1.166 = 140sm$ with interval (121,162). The actual effort was 138sm.

In summary, the background equation tries to express the relationship between size and effort for the average project. It should reflect all those properties that are constant across the environment. The factors reflect local differences among projects within the environment. Using the SEL data, the authors

could only explain half the variation with factors they used. This is partly because the baseline equation provides such a good estimate to begin with.

The authors drew the conclusions that although it is hard to get good factors and good data for the factors, this is a viable approach to estimating software development resource expenditures. It provides a local model of resources based upon the organizations own experiences.

The drawbacks to a local model, such as the meta-model approach, are: (1) it requires local data on past projects, (2) we are unable to calibrate our results, even if they are more accurate, against what other organizations might consider a reasonable estimate. The benefits of a locally developed model are: (1) greater accuracy of estimation based upon the organization's own past experiences, (2) the ability to use locally developed parameters, relevant to the environment, and (3) a deeper understanding of the local environmentally developed cost drivers and their values.

This third benefit can be used to provide insight to the organization on how to modify their environment to lower cost and improve productivity. For those factors that bring projects below the expected effort, such as methodology in the above example, more projects can use that methodology to achieve greater productivity gains. For those factors that bring projects above the expected effort, new approaches may be tried to minimize their negative effects.

However, this last benefit, becomes a drawback from an estimation point of view, i.e., by changing the factor values for future projects, we are changing the background equations and the factors that differentiate projects. For example, using the effective methods in all projects changes the background equation and removes methodology, as currently defined, as a differentiating factor. Thus the model has to be redeveloped when the environment has been changed, i.e., a new baseline equation and new factors need to be developed and old data points should be removed from the model.

Although local models have several advantages over the accuracy of global models, assuming local data exists, both types of model share the problem that it is difficult to estimate the major parameter, e.g., function points or source lines of code, based upon the product characteristics and the amount of information we have at the time we estimate.

Analyzing Cost Factors

In order to study the effects of environmental factors further, Bailey and Basili used the same data to evaluate the effect of various factors on productivity and quality. Based upon a similar study by Doug Brooks at IBM/FSD, in analyzing the Walston/Felix data, they tried to see if methodology had a significant effect on productivity in the SEL environment.

The assumption was that projects vary with respect to the set of software development techniques used and the extent to which they were used. There was formal training for some projects which was included as a factor. Each project was rated with respect to a large set of factors covering environment, methodology, experience, performance, etc. For each of the factors, values were assigned on a six-point

scale, e.g., 0 to 5. The ratings were subjective, based upon an intuitive model of process conformance. The models were developed relative to the local environment and the values were assigned near end of project without knowledge of the productivity or quality results. The ratings were performed by someone from each of the three organizations in the SEL, NASA (Frank McGarry), CSC (Jerry Page), and the University of Maryland (Vic Basili).

A non-parametric statistical test was used to see if the projects with high methodology use came from a different environment, with respect to productivity, than the projects with a low methodology use. The approach was to divide the ratings for each technique into 3 categories: low (-1), medium (0), high (1). This was done to offset differences in scales. The ratings were then added to get a cumulative methodology rating. The projects were then divided into groups based upon their rating and analyzed using the Mann-Whitney-U test (non-parametric statistics).

The results of the grouping of projects into three groups with a low, medium and high methodology rating, and into two groups with a low and high methodology rating are given in Figure 7.7. For the three groupings, the Low group was different from the combination of the Medium and High groups at a significance at the 0.5 level. The High group was different from the combination of the Medium and Low groups at a significance at the .03 level. For the two groupings, the Low group was different from the High group with a significance of the .05 level.

Group:	Low	Medium	High
Ratings:	(-11,-9,-9,-9)	(2,2,2,1,0,-1,-3-3)	(12,11,8,5,5,3)
Productivity:	535 DL/SM	660 DL/SM	768 DL/SM

Group:	Low	High
Ratings:	(-11,-9,-9,-9,-3,-3,-1)	(0,1,2,2,2,3,5,5,8,11,12)
Productivity:	602 DL/SM	710 DL/SM

Figure 7.7 Effect of Methodology Factors on Productivity.

To gain some deeper insight into the effect of the individual factors, a set of correlations were derived to analyze the relationship between productivity and each of the various factors. There was no significant relationship between productivity and size (no point in categorizing by size). This was not a surprise since a straight line fit was as good as a non-linear fit to the original data in the cost equation. For all

factors that showed a difference among projects, the correlations between methodology and productivity are given below:

PDL	.26
Formal design review*	.62
Design formalism	.38
Design decision notes*	.62
Design walk-through	.28
Code walk-through	.19
Code reading*	.58
Top down design	- .19
Structured code	.02
Librarian use*	.52
Chief programmer team*	.62
Formal test plans**	.51
Heavy management involvement	- .09
Formal training*	.58
Top down code	.29

Those marked with a single * were significant at $< .01$, Those with a ** were significant at $< .05$.

It is important to note the inaccuracy of such an approach. The approach does not provide any indication of cause and effect. It is possible for a factor to correlate highly with productivity only because it happened to occur in a set of projects that had other factors that improved productivity. On the other hand it is possible for a factor that does improve productivity to have no correlation, or even a negative correlation, because it occurred with a variety of positive and negative other factors in the set of projects under study.

For example, top down design appears to have had a negative effect on productivity (although the results are not significant). This was because top-down design was used in some projects where no other effective methods were used and **most methods cannot be effective in isolation**. On the other hand, formal training in methodology had a significantly high correlations with productivity. This is most likely because the formal training caused a set of methods to be used that had an effect on productivity (and possibly because training in methodology is important).

They also examined the correlations with the other factors they used in developing the meta-model. These factors included customer interface complexity, customer originated program design changes, the complexity of such issues as: application processing, program flow, internal communication, external communication, data base complexity, a general complexity rating, constraints such as I/O capability, timing, main store, programming group experience with machine familiarity, language familiarity, application experience, same type before, hardware changes during development, percent real time or interactive, percent programmer involved in specifications. All but one, percent real time or interactive showed no significant correlation with productivity, which showed a significant difference at .05 level in the intuitively wrong direction, i.e., a higher % real time implied a higher productivity. This might be

explained by the fact that the amount of real time in these applications is really small and so when there was some real time programming, the best people worked on the problem and they were in general more productive.

In examining the relationship between productivity and various factors, the authors concluded that in the SEL environment, size was not a factor in differentiating productivity, since they found no significant relationship between productivity and size, that a large set of methodology factors showed varying degrees of positive correlation with productivity and that a combined methodology factor showed a significant positive correlation with productivity. Thus they were able to demonstrate that projects with high methodology rating came from a different population than those with a low methodology rating and so methodology is correlated with productivity. Essentially no other factors showed a significant positive correlation with productivity.

In a follow-up study Bailey and Basili tried to study the relationship between various factors and quality. They compressed three sets of metrics into three factors: quality, methodology, and complexity and used factor analysis to determine the effects. Methodology and complexity were not significantly correlated. Quality was significantly correlated with methodology ($R = .67$) and complexity ($R = .64$) at less than .001 significance level. Using methodology alone to predict quality, they obtained an $R^2 = .65$, using methodology and complexity, they obtained an $R^2 = .65$. Thus there is evidence that in the SEL, one can predict quality from methodology and complexity and that methodology is correlated with quality.

Theoretical Dynamic Resource Modelling

Once an effort estimate is made, the next question is how to assign people to the project so that he deadlines for the various development activities will be met. Each of the methods discussed so far uses an empirical approach to identify the activities that are parts of the development process of a typical project within their environment. Using effort data from past projects, the percentage of effort expended on each activity is estimated is determined. These percentages serve as a baseline and are intuitively adjusted to meet the expected demands of a new project. For example, total cost may be allocated into five major subareas: analysis cost, design cost, coding cost, testing cost, and documentation cost. Each of these subareas is subdivided again, depending upon the activities in the subareas. In this way, each activity can be staffed according to its individual budget. Allocation of time is determined by history and good management intuition.

An alternative approach is to justify resource expenditures based upon an underlying theory of how people solve problems. We will refer to these types of approaches as theoretical dynamic resource models. The original model of this type is due to Larry Putnam.

The model is based upon a hardware development model (due to Peter Norden) which noted that there are regular patterns of staff buildup and phase-out independent of the type of work being done. It is related to the way people solve problems. Each activity could be plotted as a curve which grows and then shrinks with regard to staff effort across time. Norden isolated several activities associated with

hardware development: planning and specification, design, prototyping, and release. Similar curves were derived by Putnam for software cycles: planning, design and implementation, testing and validation, extension, modification and maintenance.

The basic theory behind this model is based upon the ideas that software development is a problem-solving effort, design decision making is the exhaustion process, and activities partition problem space into subspaces corresponding to the various stages (cycles) in the life cycle. Assumptions about the problem subset are that the number of problems to be solved is finite, the problem-solving effort makes an impact on and defines an environment for the unsolved problem set, a decision removes one unsolved problem from the set (the model assumes events are random and independent) and the number of people is proportional to the number of problems $\hat{\rho}$ for solution.

Based upon these assumptions, Putnam derives an effort curve, the integral form of the life cycle equation

$$y = K * (1 - e^{-at})$$

where

y = the cumulative manpower used through time t

K = the total manpower required by the cycle stated in quantities related to the time period used as a base, e.g., staff-months/month

a = A parameter determined by the time period in which y reaches its maximum value (shape parameter)

t = time in equal units counted from the start of the cycle

The life cycle equation (derivative form) is

$$y' = 2 K a t e^{-a t}$$

where

y' = the manpower required in time period t stated in quantities related to the time period used as a base

K = total manpower required by the cycle stated in the same units as y

The curve represents the staffing buildup (a Rayleigh curve). Putnam argued that the sum of the individual cycle curves results in a pure Rayleigh shape because software development is implemented as a functionally homogeneous effort (single purpose).

The shape parameter a depends upon the point in time at which y reaches its maximum, i.e.,

$$a = 1/2 t_d^2$$

where t_d is the time to reach peak effort. Putnam empirically showed that t_d corresponds closely to the design time (time to reach initial operational capability). Substituting for a we can rewrite the life cycle equation as

$$y = \frac{K}{t_d^2} * t e^{-t/2 t_d^2}$$

Based upon his analysis, Putnam concludes that large software development projects follow a life cycle pattern describable by the Rayleigh (manpower) equation:

$$y = 2 K a t e^{-at^2}$$

Software systems have 3 fundamental parameters: the life cycle effort (K), development time (t_d), and difficulty ($D = K/t^2$). Productivity is related to the difficulty and state of technology constant C_k .

Management cannot arbitrarily increase productivity. Management cannot reduce development time without increasing difficulty. The tradeoff law shows the cost of trading time for people.

Resource Planning

We have discussed only a sampling of the models that exist in the literature. There are a variety of other models, but most of them are of a similar to those discussed above. In order to estimate resources, the models should be an aid to software development management and engineering. Common sense should never be abandoned.

None of these models are not accurate enough to be taken as the sole source. One approach to doing resource estimation might be to apply more than one model and examine the range of predictions offered, compare the results. If they agree, one can be more secure about the estimate. If they don't agree, examine why not, ask what model assumptions were not satisfied and what makes this project different. Make sure you are comfortable with the explanation of the difference.

What is needed is a system that combines global and local of models, allowing for different inputs at different times, and providing an integration mechanism, that allows us to better understand the parameters of both sets of models. This way we can learn from our application of the system so that we can improve our estimates over time. That is, if we apply both global and local models when possible, and based upon our analysis of the both models, modify our understanding of their parameters and evolve them with respect to the choice of baseline equation and cost drivers to provide better accuracy for future projects, we can do a better job of cost estimation and comparison with other environments in the future..

In order to provide a basis for resource planning and allocation and take advantage of prior history, Jeffrey and Basili proded a resource consumption model that characterized types of resources and their use

Resources are consumed during the software process. Software process characteristics are super-ordinate to the resources consumed on a project. A process characterization includes such characteristics as project type, organizational development conventions, project manager preferences, target computer system, development computer system, project schedules or milestones, and project deliverables.

They classify a variety of different types of resources: hardware, software, human, calendar time, support, e.g., supplies, materials, communications, facility costs, etc. They categorize the viewing of

resources by three different dimensions: incurrence, availability, and use descriptors. **Incurrence** is subdivided into estimated and actual. **Availability** is subdivided into desirable (resources of value), accessible (resources able to be used), and utilized (resources used). Desirable resources are those considered ideal for any project, i.e., unconstrained by availability, implying the ideal hardware, software and calendar time, ideal people characteristics. Accessible resources are theoretically those available to the project available within the organization, chosen from data base of corporate resources. Utilized resources are those available and anticipated to be used or actually used by the project, i.e., those driven by project constraints (e.g., cost) and other corporate needs. **Use Descriptors** are subdivided into the nature of the work or activity (e.g., testing, design), the point in time (e.g., calendar dates needed), and the resources utilized (e.g., hours, dollars, units).

Information about resources can be obtained from individual and group knowledge, a knowledge base, a data base of prior projects, or algorithmic models. Inputs to the model can occur at project milestones, by manager initiated points in time based upon divergence between estimated and actual, or at system initiated points in time based upon divergences recognized by the measurement system.

If we combine incurrence and availability dimensions, we get the following categories: Estimated Desirable: those resources considered ideal for the project at planning time, Estimated Accessible: those resources available with the organization that can be used by the project, Estimated Utilized: those resources anticipated to be used by the project, Actual Utilized: those resources actually used by the project, Actual Accessible: with hindsight the resources which were available and should have been utilized, Actual Desirable: with hindsight the resources which should have been made available and used.

The differences between each of the different resource models provides a unique form of input to the organization. The difference between estimated desirable and estimated accessible provides input to the risk management plan. The difference between estimated accessible and estimated utilized provides input to the contingency plan. The difference between the estimated and actual utilized provides input to the manager for real time adjustments to resource allocation and can be used to provide measures of progress and problems. The differences between actual utilized and accessible is feedback needed for future project planning, i.e., what should we really have used to make this project work. It represents an analysis and smoothing of the actual utilized resources. The difference between actual accessible and actual desirable is the feedback to corporate planning, i.e., what resources does the corporation have to acquire in order to successfully complete projects of the type developed. The relationships of these models and their differences are given in Figure 7.10.

Estimated Desirable: Resources considered ideal
 |-----> ***Input to Risk Management Plan***

Estimated Accessible: Resources available
 |-----> ***Input to Contingency Planning***

Estimated Utilized: Resources anticipated to be used
 |-----> ***Real time management adjustment***

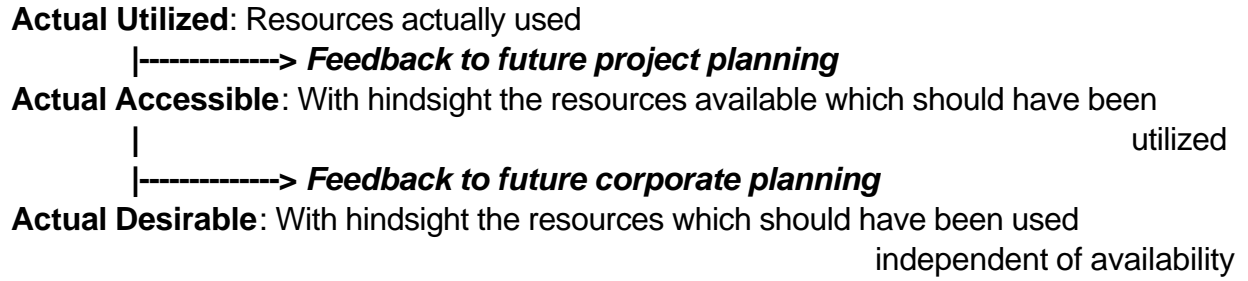


Figure 7.8 Combining Views

When viewed from the point of view of the Improvement Paradigm, resource planning involves the following activities: During planning, decisions are made about such things as obtaining a further resource (updating the corporate resource data base), committing to development without the expert, negotiating for full or partial decommitment of the expert. During execution, review and re-estimation are done to modify the plan by allocating contingency resources, revise estimates of accessible resources, and revise desirables. During post mortem analysis and the packaging of experience, revisions are made to various experience base models, including project and environment models, and the desirable and accessible resources models. Lessons learned are developed.