Estimating and Tracking Software Projects

By Vince Kellen
5/1/1993
mailto:vince@kellen.net
http://www.kellen.net/vk

Have all your projects been delivered on time and within budget? If you are a corporate or professional developer, have all your bids been 100% accurate? If so, don't bother reading any further, you probably don't need any help. But if you are like other developers who continually struggle with trying to predict how long software development should take, read on.

The answer to the question "Why estimate and track software projects?" is simple. To avoid pain. To prevent the discomfort you feel when customers call and complain that the project is too expensive and is taking too long. To prevent the loss of sleep you and your staff suffer pulling all-nighters to GET-THE-SYSTEM-OUT-THE-DOOR. To take vacations without toting a computer around the beach. The list goes on.

Another reason you might want to improve estimating and tracking projects is equally compelling: your competition may be coming up with lower bids than you and is stealing away work. And they might be delivering on their lower bids. You might be tempted to say that your competitor is underbidding and that your bid is correct.

But how accurate are your bids?

In systems development, it is not uncommon to experience differences of 100% to 300% between bids for one project, *even when the bidders are using proven project estimation techniques.* In fact, people get cynical when estimating projects, often taking a best guess and multiplying it by two or three.

Some people believe that software development will, by its nature, defy our efforts to measure it. Software is an art, not a science, they say. You cannot quantify exactly how long it takes to create software. The process is much too complex and *creative* to be measured effectively. If that is what you believe, then it is my job to convince you that you're wrong. In any case, let's take a very quick tour down the Software-As-Art path anyway and see where it leads.

Software-As-Art

The problems with tracking and estimating projects start with the Software-As-Art mentality. This sort of thinking often leads to conclusions such as "Only we can do GOOD software quickly because we have better people (artists) than our competitors" or "They (our competitor)

will never be able to do GOOD software quickly because they can't think CREATIVELY" or "Why estimate? I've been doing this long enough, I know how long something will take." These conclusions lead to bad practices which can slowly erode any advantages your firm or programming shop might have had.

In order to measure software projects, you have to think like an engineer, not like Michaelangelo. You have to stop thinking of software as a piece of marble, waiting for the artist's inspiration to remove the pieces, revealing a beautiful statue. You have to start thinking of it as a complex highway that needs to be built with proven materials, consistent results, measurable milestones, and lots of drivers with car phones and your number written on their dashboard.

This does not mean that programmers can't be creative. After all, engineers can be creative. Mathematicians can be creative. Particle physicists can be creative. Just because something is creative doesn't make it art. Rejecting the Software-As-Art notion means that software development can be predicted, managed, communicated and measured.

Measuring Projects: Advantages

In order to estimate and monitor projects effectively, you must first measure. You will have to measure existing applications and new ones. After all, you can't easily estimate and monitor a new project if it isn't measured in some way. Fortunately, measuring some software projects can be easier than measuring other types of software projects. To understand why, look at what is involved in doing accurate size estimation for projects using multiple development platforms.

Estimation must take into account the different speeds at which different programmers code and make adjustments for different development platforms. Programmers just learning a language are less productive than old pros. Coding in C is faster than coding in assembler. Using a code generator for user interface programs is faster than coding them by hand. Estimation must also accurately measure a system's size. The measurement method chosen often needs to be applied across all platforms, which introduces further variance in the estimation process.

Accurately estimating projects is frustrated by the following factors:

1. Variations in programmers' skill

2. Variations in development platforms

3. Variations in project scope and size

4. Variations in project team size and organization

Given the magnitude of these variations, it is a not just a wonder that estimation can be accurate, it is amazing that it can be done at all. If you already estimate projects as part of your company's mixed-platform estimation process, your estimation process might not be as accurate as one tailored specifically for a specific platform. Predicting development effort in a specific platform

is easier because the variation of the above four factors tends to be quite less than in mixed platform environments.

Why? Several reasons pop up. Estimating single-platform projects is easier than estimating mixed platform projects because all programmers will be using one language and one development environment. If your same-platform programmers are together in one group, chances are they have similar coding styles and have less variation in skill level than you would find in a large programming shop. Some single-platform projects fall into a narrower range of sizes and scopes and most can be easily handled by single-person or small programming teams. In fact, if your shop specializes in a single-platform, you have some unique estimation advantages over shops that handle a variety of platforms.

Estimating Projects: One Approach

The approach I had used at a prior company, Kallista, Inc., to estimate projects is based on Function Point Analysis. Function Points were invented by A.J. Albrecht at IBM in the 1970s and became public when he published a paper on them in 1979. Function Points attempt to measure the software from the standpoint of the *consumer* of the software product, not the producer. Function points are a type of ruler that measures the size or deliverable functionality of a software product without regard to lines of code, number of modules, number of instructions, size of the executable or other technical factors.

Since 1979, much research and industry practice has shaped Function Point Analysis. Tom DeMarco published his "Bang" functional metric in 1982 and Albrecht published a revision to his methodology in 1984. In 1986, an International Function Points Users' Group (IFPUG) was formed and Software Productivity Research developed Feature Points, a superset of Function Points. In 1988, Charles Symons of Nolan Norton and Company published the Mark II Function Point method. In 1990 IFPUG published a Standard Counting Practices manual. Function Points have found support in the IEEE as well.

As you can see, function points have evoked interest from industry and academia.

At Kallista, we had taken the function point methodology and applied it directly to projects. In the process, we deviate from the actual Function Point methodology to make measuring easier, quicker and hopefully more accurate. Since our development environment is quite homogenous (compared with shops with programmers in multiple platforms), the shortcuts have not yet seriously compromised accuracy. In addition, we use the function methodology as a starting point for project tracking and monitoring.

Implementing a function point estimation process is a two-step process. First, data on existing projects must be recorded in a database. Second, the estimated project is then compared with the projects in the database using regression analysis or ranking to determine the man-hours estimate.

Unlike other estimation methods, function point estimation uses statistical correlation to predict the amount of effort required to build a system. With statistical correlation, you evaluate your projects' key measures. For example, we determine how big a completed application is by counting up function points (each form, report and table is assigned a number based on how many fields are in that object). Then we add up the hours spent building the application, which leaves us two numbers, function points and effort. These two numbers are then analyzed and we use regression analysis to determine the extent these two numbers are correlated. If they are sufficiently correlated, we then have an equation which converts function points to man hours.

While this may seem like a wild leap of faith to some of you who are familiar with bottom-up estimation in which each component in the application is independently estimated and then all components are added together, it does work and has value. For the statisticians out there, this is, of course, old news. Statistical correlation is used in a variety of industries and applications.

Recording Statistics on A Completed Project

In order to be able to estimate projects effectively, you need a good record of previous projects. For this reason, the first step in estimating projects is maintaining an up-to-date and accurate record of prior projects. The data that you might want in this project database includes:

1. Project ID. To identify the project

2. Man-hours expended. How much time spent developing this application?

3. Function point summaries

4. Total number of table elements (fields in tables)

5. Total number of report elements (fields in reports)

6. Total number of form elements (fields in forms)

7. Total number of procedures

8. Total number of source lines of code

Man hours expended

This includes *all* the time it took to complete this project, not just billable time. It should accurately reflect the total hours required to create the application, including testing, rework, removing defects, administration and documentation.

Function point summaries

Function point analysis enables system designers to estimate a project directly from attributes of a project that are easily counted, usually early in the project life cycle. It requires that someone count up the following items:

1. external inputs

2. external outputs

3. logical internal files

4. external interface files

5. external inquiries

In Database development, certain objects can be equated with the function point counterparts. Reports can be roughly equated with external outputs, forms and dialog boxes with external inputs and tables with logical internal files. External interface files are files that communicate to other applications. External inquiries are screens which let the user ask for help in the system. For the purposes of maintaining consistent counting practices in the Database environment, we ignore the external inquiries category. In normal practice, external inquires carry the same weight as external inputs.

Function point types Database equivalents

1. external inputs = Forms and dialog boxes

2. external outputs = Reports

3. logical internal files = Tables

4. external interface files = Tables, files

5. external inquiries

To count function points, each report, form and table is evaluated for complexity which can be simple, average or complex. The rating of simple, average or complex carries a weight according to the following table:

Table 1. Function Point Weights

Object	Simple	Average	Complex
Forms	3	4	6
Reports	4	5	7
Tables	7	10	15

Interface files	5	7	10
External inquiry	3	4	6

Each object is assigned a value based on this table and all objects are summed by type and recorded. In addition, all function point type summaries are added together, giving what is called an unadjusted function point total (UFP).

A technical complexity factor can them be assigned. This has the effect of increasing or decreasing the function point total to ±35% of the UFP. The technical complexity factor is determined by answering questions to about 14 or so questions concerning the overall complexity and other factors which may affect the estimation of effort. Multiplying the UFP by the TCF results in a function point total (FP).

The 1984 Function Point revision outlines the following 14 TCFs:

1. Data communication

2. Distributed functions

3. Performance objectives

4. Heavily used configuration

5. Transaction rate

6. On-line data entry

7. End-user efficiency

8. On-line update

9. Complex processing

10. Reusability

11. Installation ease

12. Operational ease

13. Multiple sites

14. Facilitate change

Each factor is assigned a score from 0 to 5, with 0 indicating that the factor is not an issue in the application and 5 indicating that is an important factor which will require special effort. These

scores are summed with the total multiplied by .01 to convert it to a percent. Then .65 is added to the total giving a number that ranges from .65 to 1.35. The unadjusted function point total is then multiplied by this factor giving the total function points.

For the purposes of analysis at Kallista, UFPs were easier to count and seem to be reliable enough to use. This saves the effort of determining a TCF for each project. In addition, determining TCFs can't be automated easily and are subject to variation in interpretation.

Total table, report, form elements

Total table elements includes a total count of all fields in all tables, including reporting tables and any other system tables. Total report elements is the total number of fields in all reports. Total form elements is the total number of fields in forms.

These three categories are also useful for comparing projects and can be used to predict project effort instead of function points. In fact, the regression analysis that we have performed suggests that the total number of table elements is one of the best indicators of total effort.

The total table elements is simply a count of all the fields in all tables. The report and form elements are also a count of all fields in all forms but with the following addition: if the report has a lookup relationship with X number of tables, the count of fields is increased by X. This was done to make sure that linked reports and multi-table forms have a higher score than other forms and reports. If the form has three embedded forms, the total count of fields for the form is increased by only three. In addition, all calculated and summary fields on reports and forms are treated just like regular fields.

Total number of procedures/methods

This measure is useful for comparing different projects when considering an estimate. Getting the total count of procedures in most environments is easy, if the application uses procedure libraries. All you have to do is use the appropriate command or documentation tool in that envrionment.

Total number of source lines of code

This figure also is useful for comparing different projects. The count of lines of code includes comments and white space. We made this assumption to make counting lines of code easy. However, we have found that most of our programs have a consistent mix of documentation and code. Moreover, if code is more heavily commented, it is probably more complex and more expensive. By including comments in the source code count, we reflect the application's complexity in our measurement.

Automatic Counting

Counting each of these parameters for existing applications was automated at Kallista, even the determining of whether an object is simple, average or complex. Because this part of the recording process remains automated, the counting of all objects remains consistent and uniform between projects.

Automatic counting enabled us to "backfill" our database with existing projects by turning the backfill program loose on several completed projects. In fact, the more completed projects recorded, the more accurate the estimation will be. Once the number of recorded projects gets in the forties and fifties, project estimation accuracy can be considered much more reliable than when the number of projects is in the teens or twenties. The larger the population sample, the less error the estimate will contain. In order for your estimates to be worthwhile, you need a database with at least 15 or so recorded projects.

Estimating Project Effort

The most important part of estimating project effort involves determining the number of unadjusted function points (UFP) and table elements (TE) (and optionally function points (FP), screen elements (SE) and report elements (RE)) the proposed application needs. The more complete the specification, the more accurate the estimate can be. In order to estimate projects accurately, you need as specific of a design document as possible. Ideally, you would like to have sample screens, forms, reports and a completed data model.

Very often, you can't get all of these, which is why we record more than just function point summaries in our database. If you can prepare a detailed data model, identifying all the main data tables and all their fields, you may have enough information to base an estimate on. In our model, effort has been correlated with UFPs, table elements, and table elements + report elements + screen elements.

To start, you need to evaluate each form, report and table in the specification and assign the simple, average or complex score for each, depending on the number of fields in each. At Kallista, we periodically reviewed our project database and determine what constitutes an average, simple or complex form, report and table. Our findings suggested the following tables are good indicators:

Table 2. Forms scoring

Number of Fields	Score
0-5	Simple
6-17	Average
>17	Complex

Table 3. Reports scoring

Number of Fields	Score
0-8	Simple
9-34	Average
>34	Complex

Table 4. Tables scoring

Number of Fields	Score
0-5	Simple
4-22	Average
>22	Complex

Tables 2, 3 and 4 represent the form elements, report elements and table elements we defined earlier. Our automatic counting program scans each application and counts up all of these elements and creates detail records representing each table, form and report. Each table, form and report record is assigned a score and then all the scores are totaled.

Once you have scored the report, form or table as complex, lookup the appropriate function point weight in Table 1. Unfortunately there are differences between completed projects and new projects that haven't been built. Completed projects often have a series of reporting tables, which you usually don't know about in the specification. You may want to assume that each report will eventually have one reporting table with the same number of fields as the report and add this total into your estimated projects UFP.

After you have scored all the forms, reports and tables in the specification, add up the weights assigned to each. This is the unadjusted function point count (UFP). You can also add up the total number of TEs, SEs and REs as well and store them in the project master table.

Once one or all of these numbers is obtained, the number can be compared with other projects that have a similar score. This comparison can be done two ways: rank analysis and correlation analysis. Ranking the projects from lowest to highest by each of the scores (UFP, FP or TE) can help estimate how long this project will take and is easy to do. You just sort the existing projects by the ranking parameter and compare your estimated project with actual projects.

In addition, calculating the best-fit regression formula on one or all the three scores can be used to calculate the man hours estimate. Often the calculation takes the following form (these are sample formulas, not actual ones):

a) Effort = .4959 * FP ^ .98 or

b) Effort = 110 + .38 * FP

You will have to perform regression analysis on the current project database to determine the regression formula. The number returned by the formula represents the estimated effort to build the system in question. The number suggesting by the ranking and regression analysis is NOT 100% reliable. In fact, it is possible that the margin of error can be quite large. It is important to keep in mind that function point analysis is used to assist you in making an estimation effort, not make it for you.

Linear equations always give statisticians more comfort than non-linear ones. Hence, equation b) suggests a better correlation than equation a). In addition, you should perform an analysis of variance on your regression model so that you can determine the accuracy of the regression equation. For those developers who are new to statistics, there are several good introductory books on statistics out there which can help you.

At Kallista we ranked the estimated project to see how it's characteristics compare with other applications we've done. We then took the UFP count and plugged it into our regression formula that predicts how much effort the application may require. The derived effort is then evaluated and included as a factor which drives the actual bid amount.

The Real Benefit: Tracking Project Effort

Once the project has been estimated, it is theoretically possible to track projects by the number of function points delivered or the number of hours expended. Since we wereis in a competitive situation where clients are billed by the hour, tracking projects by man hours makes sense. If the bid you estimated is accepted, you will then want to track your time and effort on the project.

Why? To better predict how much the software will cost and when it will be done. In addition, your project estimation database will increase in size, and hopefully, accuracy.

The project needs to be broken down at least to the same level of granularity as the project estimate. The time required to build each form, report, and table should be estimated separately. Every distinct task that you feel is important to track should be setup in the project tracking file. It is not uncommon that even simple projects can be broken down into dozens of sub tasks. We have some projects broken down into hundreds of components. At the very least, the detailed project plan should be broken down by the object type: reports, forms and tables.

You can use this detailed component-level plan to verify your function point analysis estimates. It is different from the function point approach because it is a "bottom-up" approach where each individual item is compared with a component database, estimated and then the aggregate of each item is a project estimate. Function point analysis simply counts the complexity of each component and uses regression analysis to derive the effort required. The function point approach does not attempt to estimate the time to develop each item. Both approaches have value. In fact, in some instances, we have performed separate, independent bottom-up and function point analysis estimates to assure ourselves that the estimated effort is reasonable.

Once you have the project estimated and a detailed component-level estimate plan identified, it is a simple matter to begin recording how much time you spend on each component.

We recorded all our time on projects that exceed a certain size down to a component level and down to as little as five minute increments. The reason is two-fold. First, if we need a quick estimate on how long it may take to build a certain component, such as a report or a few data entry forms, we can look quickly at our project tracking records and find dozens of similarly sized components. We can then quickly give estimates without going through a formal function point analysis. We use this approach for handling small changes and enhancements to systems.

Second, since most components in Today's environments are built rather quickly, we avoid any gross overestimation of time. The goal of high granularity applies to not only components, but the time interval applied to the component. Although you may bill clients in hour, half-hour, or quarter-hour increments, it is a good idea to track your time at lower levels of granularity. Sometimes an application will be composed of 15 components, which during construction, you find a way to apply generic coding principles, reducing the time to build each component down to minutes. By recording time on a minute by minute basis, you can see the effect of such productivity gains in your project tracking.

While some people may scoff at the notion of recording time on such a detailed basis as too much work, our experience is otherwise. On one large project, we diligently recorded how much time it took to record our time at such a detailed level. It never took more than 2% of the total project time. That 2% effort pays off handsomely, however, when we can predict accurately when a project will be finished.

Please note that there are some tasks which may not be attributable to any one object, such as design effort, etc. These items should be recorded separately, nonetheless. We break such system-wide efforts into their own categories. Once you have the project plan broken down and time assigned to each component, the sum of the effort should be the same as the effort indication by the function point analysis.

By breaking the project down into hundreds of small tasks, you can record your progress and keep track of how much of the project is complete and how much is not. In addition, you have a database which contains the amount of time required to build each object. This may be valuable for predicting how long it will take to build similar objects on new project. Analyzing this data can serve as a powerful alternate to the function point analysis model.

The project tracking database can also be used to produce reports to determine if you are over or under the budgeted man-hours to date. We use ours to determine scheduling and ship dates. At any point in the development of the project, we can compare the original estimate with actual numbers. If we are behind schedule, we know soon enough to reallocate resources and people to get the project on track.

In addition, if the detail breakdown also includes the number of function points for each component, it is possible to produce completion reports based on a function point total. With this you can determine how many function points should have been delivered and how many actually

were. Tracking components by function points will help validate and refine the estimation model.

Our reports highlight the following key numbers:

Percent completion How much of the project is completion

To-date budget comparison On a to-date basis, is the project under or over budget

Function points delivered How many function points have been delivered so far?

If the project plan is detailed enough and the average size of each task is small enough, the variance between the budgeted amount and the actual amount of time spent can be quite high, which does not necessarily concern us. What's more important is that the number of tasks be high enough so that the risk of error is spread out over lots of tasks and hence greatly minimized. it's just as likely that the programmer will be over on any given task as under. In other words, we may lose the ability to accurately predict a single task, but we gain in the ability to predict the effort for the project.

Additionally, if the project is going over budget early, you can reevaluate design decisions and implement lower-cost solutions to bring the project on track. In fact, programmers intuitively do this as they program. If they have less time, they usually find a lower-cost solution. In this way, the detailed project plan can become a self-fullfilling prophecy, which is exactly what we want. Now if your programmers are implementing lower-cost solutions that don't meet customer specifications in order to get a project on track, that's a problem. But since software development is a complex process, there are always a myriad of alternatives to the design problems, each with a different cost.

Finally, we record defects in our shipped applications and record which form, report or table the defect was found in. We then can look at a particular form, report or table object and have the following numbers:

1. The original estimate for the object
2. The actual hours expended for that object
3. The number of bugs reported for that object

Having these numbers attributed to the original estimate helps us identify where we can improve our estimation model.

Summary

The following steps are needed to estimate and track projects:

1. Build a project database maintaining various application counts

2. Backfill the database by evaluating several completed applications

3. Get as complete a project specification as possible to application function points and table, screen and report elements. At the least you will need TE.

4. Using either FPs, UFPs, TE or TE+FE+RE, generate an effort estimate by ranking the estimated project with the project database or use regression analysis to derive an effort estimate. Use the generated effort number as a guide in your decision, not a replacement.

5. Develop a detailed, component-level project plan for the estimated project.

6. Track your time spent on each component.

7. Compare your project estimate with actual time totals and adjust your schedule and/or designs as needed.

8. Add the project to the project database when it is completed.

9. Review and refine your function point simple-average-complex counting formula and adjust as needed.

10. Review and refine your project tracking and reporting mechanism routinely.

Understand that you need to continually review and improve your estimation and tracking methods. The more projects you add to your project database and the more experience you gain in estimating and tracking, the easier it becomes. Discipline is a key word. It requires work to build and take care of a project database. It will be easy to conveniently forget to keep your time tracking database up to snuff, thinking that a few minutes missed here or a few minutes missed there won't hurt. Nothing could be further from the truth. The estimation and tracking system's accuracy and utility will slowly degrade as this attitude creeps in.

For those that persevere, enjoy the vacation on the beach without the laptop computer.

Appendix

Table 5 lists a sample of 14 various projects from our project database.

Table 5. Some sample project statistics

Project	Inputs	Outputs	Files	UFPs	SLOC	Effort	TE
1	78	31	130	239	7239	163	271
2	388	332	1789	2509	45830	1011	3567
3	40	0	73	113	3435	22	58

4	63	146	169	378	3037	73	607
5	10	17	24	51	2172	39	82
6	337	170	518	1025	30086	240	1009
7	20	14	121	155	1057	41	227
8	24	64	138	226	5778	53	281
9	398	82	802	1282	15122	595	1139
10	412	105	633	1150	10203	410	1355
11	349	297	650	1296	26264	572	2508
12	72	161	321	554	7341	198	1788
13	81	120	385	586	8326	301	1128
14	74	57	138	269	15456	113	405

Legend

Inputs	Forms
Outputs	Reports
Files	Tables
UFP	Unadjusted Function Points
SLOC	Source lines of code, including comments
Effort	Man-hours of effort
TE	Total number of fields in all tables

Between these 14 projects and about a couple of dozen more, our regression analysis indicates that UFPs and TEs, as we have defined and counted them, are highly correlated with the effort required to build the application.

Table 6 and 7 lists the table structures for our project master and project detail tables.

Table 6. Project master structure

Field	Type	Comments
Project id	A8*	Project identifier
Inputs	N	Count of external inputs
Outputs	N	Count of external outputs
Files	N	Count of logical internal files
Interface files	N	Count of external interface files
TCF	N	Technical complexity factor adjustment

UFP	N	Unadjusted function point total
FP	N	Adjusted function point total
Procs	N	Count of procs/methods in app
SLOC	N	Count of source lines of code
Effort	N	Count of man-hours expended
Screen elements	N	Count of screen elements
Report elements	N	Count of report elements
Table elements	N	Count of table elements
Main password 1	A20	Password used to automate counting password protected tables
Main password 2	A20	A second password used to automate counting password protected tables
Location	A125	The directory location of the application
Time file	A125	The location of the time tracking file which records actuals for [Project id]

Table 7. Project detail structure

Field	Type	Comments
Project id	A8*	Project identifier
Item id	A15*	Item identifier
Function type	A2	TE, RE, SE, EI
Complexity	A1	A=average, S=simple, C=complex
Total elements_links	N	Total number of elements + links
Total elements	N	Total number of elements in this item
Total links	N	Total number of embedded forms or lookup links or tables in the data model
UFP	S	Unadjusted function point total

References

Albrecht, A. "Measuring Application Development Productivity," *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium*, October, 1979, pp 83-92.

Albrecht, A. and Gaffney, J. "Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 6, November 1983, pp. 639-648.

Beohm, B. *Software Engineering Economics*. Prentice-Hall, 1981.

Jones, C. *Applied Software Measurement.* McGraw-Hill, Inc., 1991.

Kemerer, C. and Porter, B. "Improving the Reliability of Function Point Measurement: An Empirical Study," *IEEE Transactions of Software Engineering*, Vol. 18, No. 11, November, 1992, pp. 1011-1024.

Symons, C. "Function Point Analysis: Difficulties and Improvements," *IEEE Transactions of Software Engineering*, Vol 14, No. 1, January 1988, pp. 2-11.

Verner, J. and Tate, G. "A Software Size Model," *IEEE Transactions of Software Engineering*, Vol 18, No. 4. April 1992, pp. 265-278.

Yourdon, E. *Decline and Fall of the American Programmer*, Yourdon Press/Prentice Hall, 1992.

Bibliography

Brooks, F. *The Mythical Man-Month*, Addison-Wesley Publishing Company, 1982.

Conte, S., Dunsmore, H., Shen, V. *Software Engineering Metrics and Models*. Benjamin/Cummings Publishing Company, Inc. 1986.

DeMarco, T. *Controlling Software Projects*, Yourdon Press/Prentice-Hall, 1982.

Gilb, T. *Principles of Software Engineering Management.* Addison-Wesley Publishing Company, 1988.

Grady, R. *Practical Software Metrics For Project Management and Process Improvement*, Prentice-Hall, 1992.

Humphrey, W. *Managing the Software Process.* Addison-Wesley Publishing Company, 1990.