International Journal of Engineering Research & Management Technology
May- 2015 Volume 2, Issue-3
ISSN: 2348-4039
Email: editor@ijermt.org www.ijermt.org
"Sharpening Skills..... Serving Nation"

Special Issue: 2nd International Conference on Advanced Developments in Engineering and Technology
Held at Lord Krishna College of Engineering Ghaziabad, India

Survey of Recent Software Cost Estimation

Madhur
M.Tech Scholar
Department of Computer Science & Engineering
Al-Falah School of Engineering and Technology
Faridabad, Haryana

Rupak Kumar
Assistant Professor
Department of Computer Science & Engineering
Al-Falah School of Engineering and Technology
Faridabad, Haryana

ABSTRACT
This paper summarizes several classes of software cost estimation models and techniques: parametric models, expertise-based techniques, learning-oriented techniques, dynamics-based models, regression-based models, and composite-Bayesian techniques for integrating expertise- based and regression-based models.
Experience to date indicates that neural-net and dynamics-based techniques are less mature than the other classes of techniques, but that all classes of techniques are challenged by the rapid pace of change in software technology.
The primary conclusion is that no single technique is best for all situations, and that a careful comparison of the results of several approaches is most likely to produce realistic estimates.

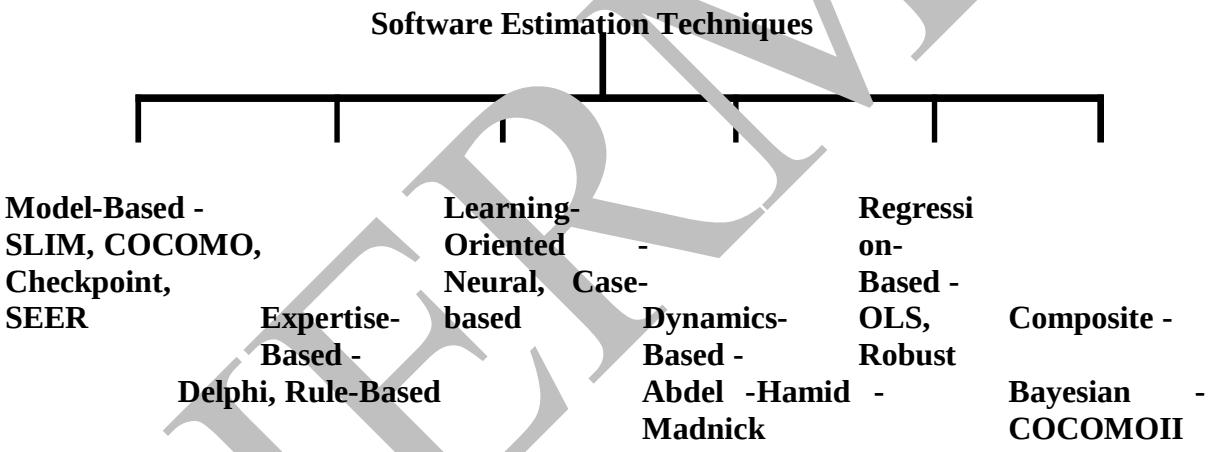
1. INTRODUCTION
Software engineering cost (and schedule) models and estimation techniques are used for a number of purposes. These include:
Budgeting: the primary but not the only important use. Accuracy of the overall estimate is the most desired capability.
Tradeoff and risk analysis: an important additional capability is to illuminate the cost and schedule Sensitivities of software project decisions (scoping, staffing, tools, reuse, etc.).
Project planning and control: an important additional capability is to provide cost and schedule breakdowns by component, stage and activity.
Software improvement investment analysis: an important additional capability is to estimate the costs as well as the benefits of such strategies as tools, reuse, and process maturity.
In this paper, we summarize the leading techniques and indicate their relative strengths for these four purposes.
Significant research on software cost modeling began with the extensive 1965 SDC study of the 104 attributes of 169 software projects [Nelson 1966]. This led to some useful partial models in the late 1960s and early 1970s.
The late 1970s produced a flowering of more robust models such as SLIM [Putnam and Myers 1992], Checkpoint [Jones 1997], PRICE-S [Park 1988], SEER [Jensen 1983], and COCOMO [Boehm 1981]. Although most of these researchers started working on developing models of cost estimation at about the same time, they all faced the same dilemma: as software grew in size and importance it also grew in complexity,

making it very difficult to accurately predict the cost of software development. This dynamic field of software estimation sustained the interests of these researchers who succeeded in setting the stepping-stones of software engineering cost models.

Just like in any other field, the field of software engineering cost models has had its own pitfalls. The fast changing nature of software development has made it very difficult to develop parametric models that yield high accuracy for software development in all domains. Software development costs continue to increase and practitioners continually express their concerns over their inability to accurately predict the costs involved. One of the most important objectives of the software engineering community has been the development of useful models that constructively explain the development life-cycle and accurately predict the cost of developing a software product. To that end, many software estimation models have evolved in the last two decades based on the pioneering efforts of the above mentioned researchers. The most commonly used techniques for these models include classical multiple regression approaches. However, these classical model-building techniques are not necessarily the best when used on software engineering data as illustrated in this paper.

Beyond regression, several papers [Briand *et al.* 1992; Khoshgoftaar *et al.* 1995] discuss the pros and cons of one software cost estimation technique versus another and present analysis results. In contrast, this paper focuses on the classification of existing techniques into six major categories as shown in figure 1, providing an overview with examples of each category. In section 2 it examines in more depth the first category, comparing some of the more popular cost models that fall under Model-based cost estimation techniques.

Figure 1. Software Estimation Techniques



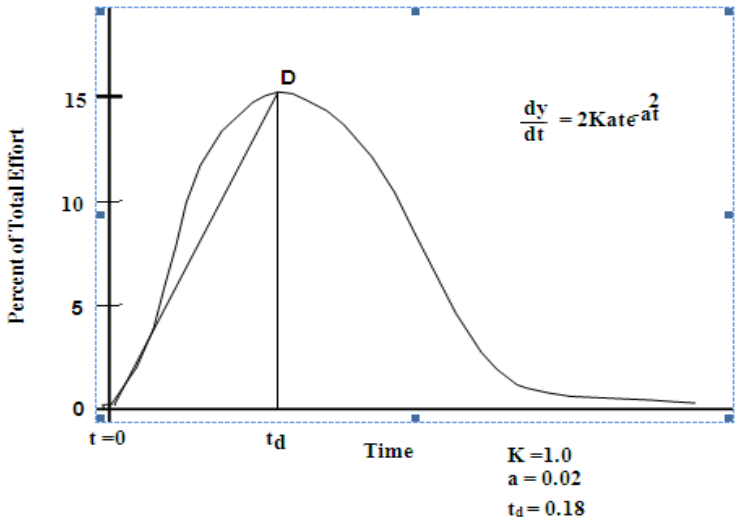
2. Model-Based Techniques

As discussed above, quite a few software estimation models have been developed in the last couple of decades. Many of them are proprietary models and hence cannot be compared and contrasted in terms of the model structure. Theory or experimentation determines the functional form of these models. This section discusses seven of the popular models and table 2 (presented at the end of this section) compares and contrasts these cost models based on the life-cycle activities covered and their input and output parameters.

2.1 Putnam's Software Life-cycle Model (SLIM)

Larry Putnam of Quantitative Software Measurement developed the Software Life-cycle Model (SLIM) in the late 1970s [Putnam and Myers 1992]. SLIM is based on Putnam's analysis of the life-cycle in terms of a so-called Rayleigh distribution of project personnel level versus time. It supports most of the popular size estimating methods

Figure 2. The Rayleigh Model.



including ballpark techniques, source instructions, function points, etc. It makes use of a so-called Rayleigh curve to estimate project effort, schedule and defect rate. A Manpower Buildup Index (MBI) and a Technology Constant or Productivity factor (PF) are used to influence the shape of the curve. SLIM can record and analyze data from previously completed projects which are then used to calibrate the model; or if data are not available then a set of questions can be answered to get values of MBI and PF from the existing database.

In SLIM, Productivity is used to link the basic Rayleigh manpower distribution model to the software development characteristics of size and technology factors. Productivity, P, is the ratio of software product size, S, and development effort, E. That is,

$$P = \frac{S}{E} \qquad \text{Eq. 2.1}$$

The Rayleigh curve used to define the distribution of effort is modeled by the differential equation

$$\frac{dy}{dt} = 2Kate^{-at^2}$$

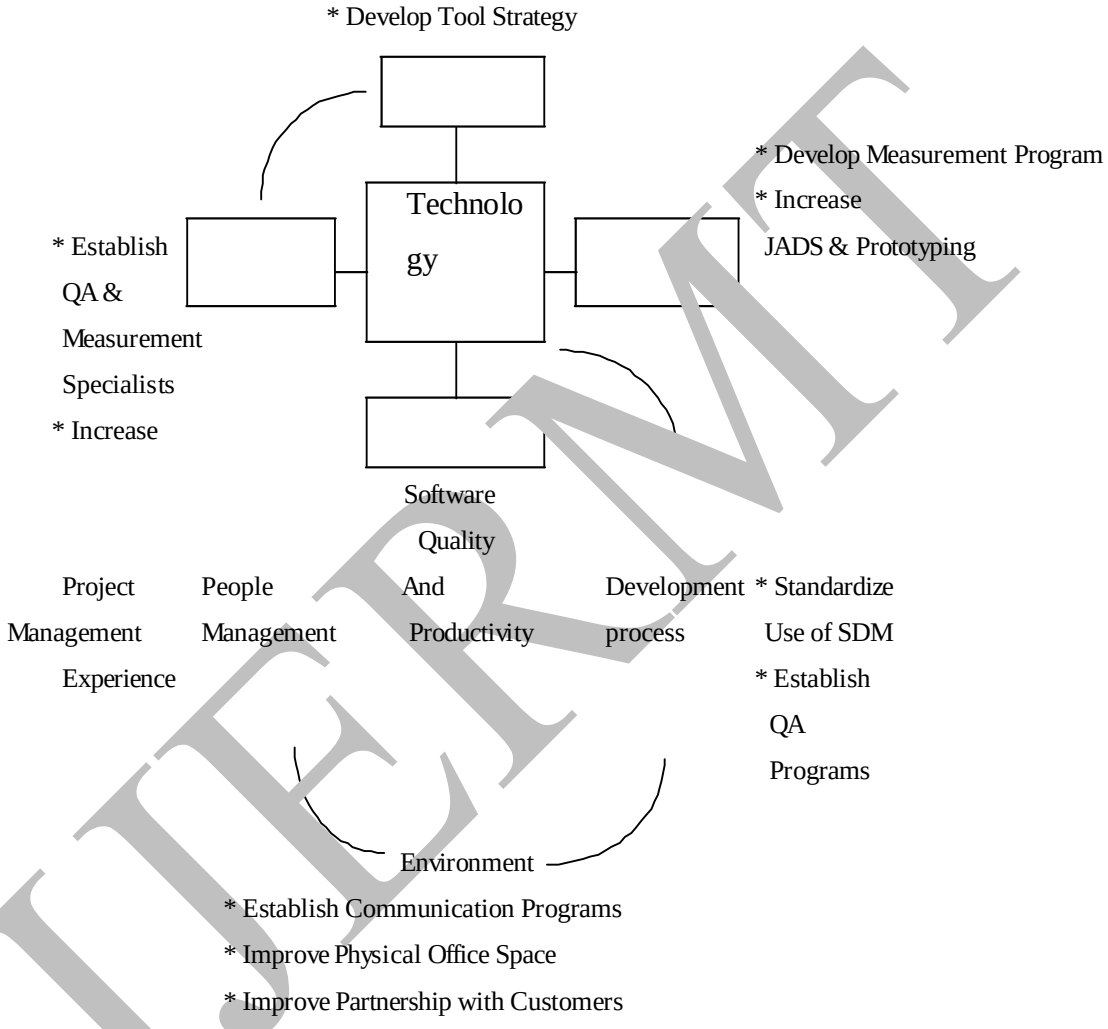
An example is given in figure 2, where $K = 1.0$, $a = 0.02$, $t_d = 0.18$ where Putnam assumes that the peak staffing level in the Rayleigh curve corresponds to development time (t_d). Different values of K, a and t_d will give different sizes and shapes of the Rayleigh curve. Some of the Rayleigh curve assumptions do not always hold in practice (e.g. flat staffing curves for incremental development; less than t^4 effort savings for long schedule stretch outs). To alleviate this problem, Putnam has developed several model adjustments for these situations.

Recently, Quantitative Software Management has developed a set of three tools based on Putnam's SLIM. These include SLIM-Estimate, SLIM-Control and SLIM-Metrics. SLIM-Estimate is a project planning tool, SLIM-Control project tracking and oversight tool, SLIM-Metrics is a software metrics repository and benchmarking tool.

2.2 Checkpoint

Checkpoint is a knowledge-based software project estimating tool from Software Productivity Research (SPR) developed from Capers Jones' studies [Jones 1997]. It has a proprietary database of about 8000 software projects and it focuses on four areas that need to be managed to improve software quality and productivity. It uses Function Points (or Feature Points) [Albrecht 1979; Symons 1991] as its primary input of size. SPR's Summary of Opportunities for software development is shown in figure 3. QA stands for Quality Assurance; JAD for Joint Application Development; SDM for Software Development Metrics.

Figure 3. SPR's Summary of Opportunities.



Estimation: Checkpoint predicts effort at four levels of granularity: project, phase, activity, and task. Estimates also include resources, deliverables, defects, costs, and schedules.

Measurement: Checkpoint enables users to capture project metrics to perform benchmark analysis, identify best practices, and develop internal estimation knowledge bases (known as Templates).

Assessment: Checkpoint facilitates the comparison of actual and estimated performance to various industry standards included in the knowledge base. Checkpoint also evaluates the strengths and weaknesses of the software environment. Process improvement recommendations can be modeled to assess the costs and benefits of implementation.

2.3 PRICE-S

The PRICE-S model was originally developed at RCA for use internally on software projects such as some that were part of the Apollo moon program. It was then released in 1977 as a proprietary model and used for estimating. The tool continued to become popular and is now marketed by PRICE Systems, which is a privately held company formerly affiliated with Lockheed Martin. As published on PRICE Systems, the PRICE-S Model consists of three sub models that enable estimating costs and schedules for the development and support of computer systems. These three sub models and their functionalities are outlined below:

- **The Acquisition Sub model**
- **The Sizing Sub model**
- **The Life-cycle Cost Sub model**

2.4 ESTIMACS

Originally developed by Howard Rubin in the late 1970s as Quest, it was subsequently integrated into the Management and Computer Services (MACS) line of products as ESTIMACS. It focuses on the development phase of the system life-cycle, maintenance being deferred to later extensions of the tool.

ESTIMACS stresses approaching the estimating task in business terms. It also stresses the need to be able to do sensitivity and trade-off analyses early on, not only for the project at hand, but also for how the current project will fold into the long term mix or "portfolio" of projects on the developer's plate for up to the next ten years, in terms of staffing/cost estimates and associated risks. The basic premise of ESTIMACS is that the gross business specifications, or "project factors," drive the estimate dimensions. Rubin defines project factors as "aspects of the business functionality of the of the target system that are well-defined early on, in a business sense, and are strongly linked to the estimate dimension." Shown in table 1 are the important project factors that inform each estimation dimension.

The items in table 1 form the basis of the five sub models that comprise ESTIMACS. The sub models are designed to be used sequentially, with outputs from one often serving as inputs to the next. Overall, the models support an iterative approach to final estimate development, illustrated by the following list:

1. Data input/estimate evolution
2. Estimate summarization
3. Sensitivity analysis
4. Revision of step 1 inputs based upon results of step 3

The five ESTIMACS sub models in order of intended use:

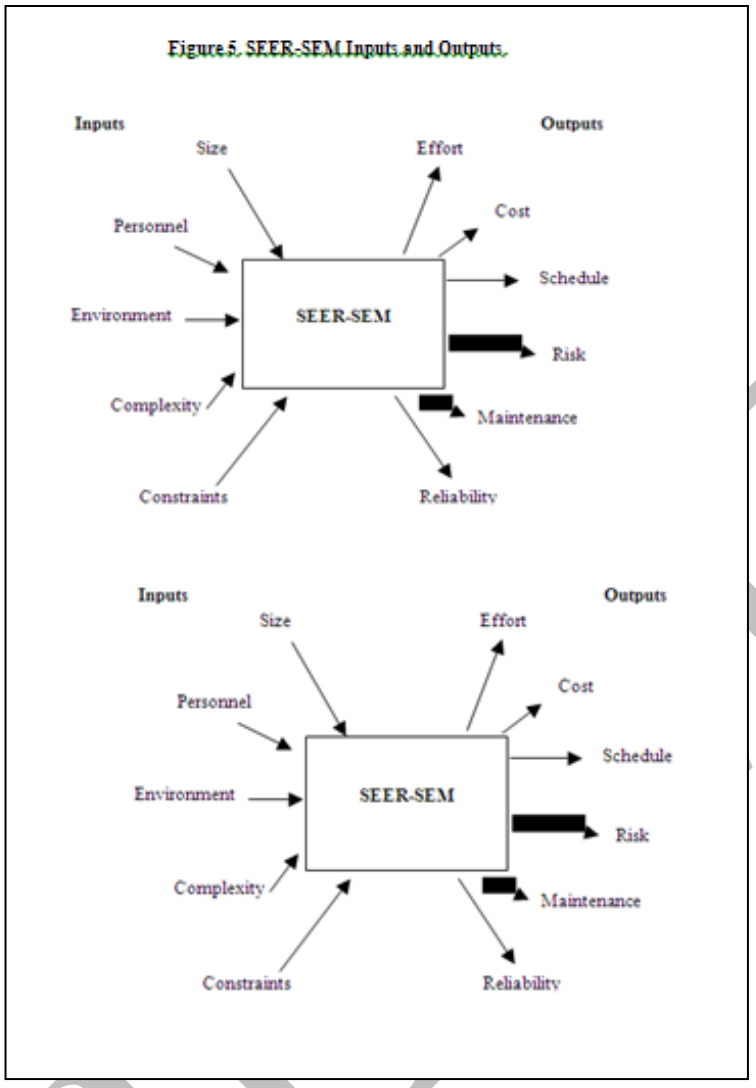
System Development Effort Estimation: this model estimates development effort as total effort hours. It uses as inputs the answers to twenty-five questions, eight related to the project organization and seventeen related to the system structure itself.

Staffing and Cost Estimation: this model takes as input the effort hours distributed by phase derived in the previous model. Other inputs include employee productivity and salaries by grade. It applies these inputs to an again customizable work distribution life-cycle database.

Hardware Configuration Estimates: this model sizes operational resource requirements of the hardware in the system being developed. Inputs include application type, operating windows, and expected transaction volumes.

Risk Estimator: based mainly on a case study done by the Harvard Business School, this model estimates the risk of successfully completing the planned project. Inputs to this model include the answers to some sixty questions, half of which are derived from use of the three previous sub models.

Portfolio Analyzer: a long range "big picture" planning tool, this last submodel estimates the simultaneous effects of up to twenty projects over a ten year planning horizon.



Features of the model include the following:

Allows probability level of estimates, staffing and schedule constraints to be input as independent variables. Facilitates extensive sensitivity and trade-off analyses on model input parameters.

Organizes project elements into work breakdown structures for convenient planning and control. Displays project cost drivers.

Allows the interactive scheduling of project elements on Gantt charts. Builds estimates upon a sizable knowledge base of existing projects.

Model specifications include these:

Parameters: size, personnel, complexity, environment and constraints - each with many individual parameters; knowledge base categories for platform & application, development & acquisition method, applicable standards, plus a user customizable knowledge base.

Predictions: effort, schedule, staffing, defects and cost estimates; estimates can be schedule or effort driven constraints can be specified on schedule and staffing.

Risk Analysis: sensitivity analysis available on all least/likely/most values of output parameters; probability settings for individual WBS elements adjustable, allowing for sorting of estimates by degree of WBS element criticality.
Sizing Methods: function points, both IFPUG sanctioned plus an augmented set; lines of code, both new and existing.
Outputs and Interfaces: many capability metrics, plus hundreds of reports and charts; trade-off analyses with side-by- side comparison of alternatives; integration with other Windows applications plus user customizable interfaces.

2.6 SELECT Estimator

SELECT Estimator by SELECT Software Tools is part of a suite of integrated products designed for components- based development modeling. The company was founded in 1988 in the UK and has branch offices in California and on the European continent. SELECT Estimator was released in 1998. It is designed for large scale distributed systems development. It is object oriented, basing its estimates on business objects and components. It assumes an incremental development life-cycle (but can be customized for other modes of development). The nature of its inputs allows the model to be used at any stage of the software development life-cycle, most significantly even at the feasibility stage when little detailed project information is known. In later stages, as more information is available, its estimates become correspondingly more reliable.

Project Architect: this module scopes and qualifies the major software elements of a project. This information is then fed into the Estimator module to estimate development schedule duration and cost. Scoping (or sizing) is done in terms of the following elements:

Applications – software subsystems supporting a business area.
Classes – representing business concepts.
User cases – business requirements defined from the user's point of view.
Packages – supporting frameworks with clear responsibility for an aspect of the infrastructure.
Components – abstractions of lower lever business services.
Services – common system features available across applications and components.

Qualifiers to software elements, rated on a scale from low to high, include the following:

Complexity – covering computations and the number of relationships.
Reuse – covers both COTS and other existing software.
Generosity – whether the software needs to be built to be reusable.
Technology – the choice of programming language.

2.7 COCOMO II

The COCOMO (Constructive Cost Model) cost and schedule estimation model was originally published in [Boehm 1981]. It became one of most popular parametric cost estimation models of the 1980s. But COCOMO '81 along with its 1987 Ada update experienced difficulties in estimating the costs of software developed to new life-cycle processes and capabilities. The COCOMO II research effort was started in 1994 at USC to address the issues on non- sequential and rapid development process models, reengineering, reuse driven approaches, object oriented approaches etc.
The Early Design model involves the exploration of alternative system architectures and concepts of operation. Typically, not enough is known to make a detailed fine-grain estimate. This model is based on function points (or lines of code when available) and a set of five scale factors and 7 effort multipliers.

2.8 Summary of Model Based Techniques

Table 2 summarizes the parameters used and activities covered by the models discussed. Overall, model based techniques are good for budgeting, tradeoff analysis, planning and control, and investment analysis. As they are calibrated to past experience, their primary difficulty is with unprecedented situations.

Table 2. Activities Covered/Factors Explicitly Considered by Various Cost Models.[2]

Group	Factor	SLIM	Checkpoint	PRICE-S	ESTIMACS	SEER-SEM	SELECT Estimator	COCOMO II
Size Attributes	Source Instructions	YES	YES	YES	NO	YES	NO	YES
	Function Points	YES	YES	YES	YES	YES	NO	YES
	OO-related metrics	YES	YES	YES	?	YES	YES	YES
Program Attributes	Type/Domain	YES	YES	YES	YES	YES	YES	NO
	Complexity	YES	YES	YES	YES	YES	YES	YES
	Language	YES	YES	YES	?	YES	YES	YES
	Reuse	YES	YES	YES	?	YES	YES	YES
	Required Reliability	?	?	YES	YES	YES	NO	YES
Computer Attributes	Resource Constraints	YES	?	YES	YES	YES	NO	YES
	Platform Volatility	?	?	?	?	YES	NO	YES
Personnel Attributes	Personnel Capability	YES	YES	YES	YES	YES	YES	YES
	Personnel Continuity	?	?	?	?	?	NO	YES
	Personnel Experience	YES	YES	YES	YES	YES	NO	YES
Project Attributes	Tools and Techniques	YES	YES	YES	YES	YES	YES	YES
	Breakage	YES	YES	YES	?	YES	YES	YES
	Schedule Constraints	YES	YES	YES	YES	YES	YES	YES
	Process Maturity	YES	YES	?	?	YES	NO	YES
	Team Cohesion	?	YES	YES	?	YES	YES	YES
	Security Issues	?	?	?	?	YES	NO	NO
	Multisite Development	?	YES	YES	YES	YES	NO	YES
Activities Covered	Inception	YES	YES	YES	YES	YES	YES	YES
	Elaboration	YES	YES	YES	YES	YES	YES	YES
	Construction	YES	YES	YES	YES	YES	YES	YES
	Transition and Maintenance	YES	YES	YES	NO	YES	NO	YES

3. Expertise-Based Techniques

Expertise-based techniques are useful in the absence of quantified, empirical data. They capture the knowledge and experience of practitioners seasoned within a domain of interest, providing estimates based upon a synthesis of the known outcomes of all the past projects to which the expert is privy or in which he or she participated. The obvious drawback to this method is that an estimate is only as good as the expert's opinion, and there is no way usually to test that opinion until it is too late to correct the damage if that opinion proves wrong. Years of experience do not necessarily translate into high levels of competency. Moreover, even the most highly competent of individuals will sometimes simply guess wrong. Two techniques have been developed which capture expert judgment but that also take steps to mitigate the possibility that the judgment of any one expert will be off.

4. Learning-Oriented Techniques

Learning-oriented techniques include both some of the oldest as well as newest techniques applied to estimation activities. The former are represented by case studies, among the most traditional of "manual" techniques; the latter are represented by neural networks, which attempt to automate improvements in the estimation process by building models that "learn" from previous experience.

4.1 Case Studies

Case studies represent an inductive process, whereby estimators and planners try to learn useful general lessons and estimation heuristics by extrapolation from specific examples. They examine in detail elaborate studies describing the environmental conditions and constraints that obtained during the development of previous software projects, the technical and managerial decisions that were made, and the final successes or failures that resulted. They try to root out from these cases the underlying links between cause and effect that can be applied in other contexts. Ideally they look for cases describing projects similar to the project for which they will be attempting to develop estimates, applying the rule of analogy that says similar projects are likely to be subject to similar costs and schedules. The source of case studies can be either internal or external to the estimator's own organization.

- identify the data or features to collect
- agree data definitions and collections mechanisms
- populate the case base
- tune the estimation method
- estimate the effort for a new project

4.2 Neural Networks

According to Gray and Mc Donell [Gray and Mac Donell 1996], neural networks is the most common software estimation model-building technique used as an alternative to mean least squares regression. These are estimation models that can be "trained" using historical data to produce ever better results by automatically adjusting their algorithmic parameter values to reduce the delta between known actual and model predictions. Gray, *et al.*, go on to describe the most common form of a neural network used in the context of software estimation, a "back propagation trained feed-forward" network (see figure 9).

The development of such a neural model is begun by first developing an appropriate layout of neurons, or connections between network nodes. This includes defining the number of layers of neurons, the number of neurons within each layer, and the manner in which they are all linked. The weighted estimating functions between the nodes and the specific training algorithm to be used must also be determined. Once the network has been built, the model must be trained by providing it with a set of historical project data inputs and the corresponding known actual values for project schedule and/or cost. The model then iterates on its training algorithm, automatically adjusting the parameters of its estimation functions until the model estimate and the actual values are within some pre-specified delta. The specification of a delta value is important. Without it, a model could theoretically become overstrained to the known historical data, adjusting its estimation algorithms until it is very good at predicting results for the training data set, but weakening the applicability of those estimation algorithms to a broader set of more general data.

Wittig [Wittig 1995] has reported accuracies of within 10% for a model of this type when used to estimate software development effort, but caution must be exercised when using these models as they are often subject to the same kinds of statistical problems with the training data as are the standard regression techniques used to calibrate more traditional models. In particular, extremely large data sets are needed to accurately train neural networks with intermediate structures of any complexity. Also, for negotiation and sensitivity analysis, the neural networks provide little intuitive support for understanding the sensitivity relationships between cost driver parameters and model results. They encounter similar difficulties for use in planning and control.

5. Dynamics-Based Techniques

Dynamics-based techniques explicitly acknowledge that software project effort or cost factors change over the duration of the system development; that is, they are dynamic rather than static over time. This is a significant departure from the other techniques highlighted in this paper, which tend to rely on static models

and predictions based upon snapshots of a development situation at a particular moment in time. However, factors like deadlines, staffing levels, design requirements, training needs, budget, *etc.*, all fluctuate over the course of development and cause corresponding fluctuations in the productivity of project personnel. This in turn has consequences for the likelihood of a project coming in on schedule and within budget – usually negative. The most prominent dynamic techniques are based upon the system dynamics approach to modeling originated by Jay Forrester nearly forty years ago [Forrester 1961].

5.1 Dynamics-based Techniques

Dynamics-based techniques explicitly acknowledge that software project effort or cost factors change over the duration of the system development; that is, they are dynamic rather than static over time. This is a significant departure from the other techniques highlighted in this paper, which tend to rely on static models and predictions based upon snapshots of a development situation at a particular moment in time. However, factors like deadlines, staffing levels, design requirements, training needs, budget, *etc.*, all fluctuate over the course of development and cause corresponding fluctuations in the productivity of project personnel. This in turn has consequences for the likelihood of a project coming in on schedule and within budget – usually negative. The most prominent dynamic techniques are based upon the system dynamics approach to modeling originated by Jay Forrester nearly forty years ago [Forrester 1961].

5.1 System Dynamics Approach

System dynamics is a continuous simulation modeling methodology whereby model results and behavior are displayed as graphs of information that change over time. Models are represented as networks modified with positive and negative feedback loops. Elements within the models are expressed as dynamically changing levels or accumulations (the nodes), rates or flows between the levels (the lines connecting the nodes), and information relative to the system that changes over time and dynamically affects the flow rates between the levels (the feedback loops).

Figure 10 [Madachy 1999] shows an example of a system dynamics model demonstrating the famous Brooks' Law, which states that "adding manpower to a late software project makes it later" [Brooks 1975]. Brooks' rationale is that not only does effort have to be reallocated to train the new people, but the corresponding increase in communication and coordination overhead grows exponentially as people are added.

Madachy's dynamic model as shown in the figure illustrates Brooks' concept based on the following assumptions:

1) New people need to be trained by experienced people to improve their productivity.
2) Increasing staff on a project increases the coordination and communication overhead.
3) People who have been working on a project for a while are more productive than newly added people.

As can be seen in figure 10, the model shows two flow chains representing software development and personnel. The software chain (seen at the top of the figure) begins with a level of requirements that need to be converted into an accumulation of developed software. The rate at which this happens depends on the number of trained personnel working on the project. The number of trained personnel in turn is a function of the personnel flow chain (seen at the bottom of the figure). New people are assigned to the project according to the personnel allocation rate, and then converted to experienced personnel according to the assimilation rate. The other items shown in the figure (nominal productivity, communication overhead, experienced personnel needed for training, and training overhead) are examples of auxiliary variables that also affect the software development rate.

Where

$\boldsymbol{x}$ = a vector describing the levels (states) in the model

$\boldsymbol{p}$ = a set of model parameters

$\boldsymbol{f}$ = a nonlinear vector function

t = time

6. Regression-Based Techniques

Regression-based techniques are the most popular ways of building models. These techniques are used in conjunction with model-based techniques and include "Standard" regression, "Robust" regression, etc.

6.1 "Standard" Regression – Ordinary Least Squares (OLS) method

"Standard" regression refers to the classical statistical approach of general linear regression modeling using least squares. It is based on the Ordinary Least Squares (OLS) method discussed in many books such as The reasons for its popularity include ease of use and simplicity.

More recently, Mad achy used system dynamics to model an inspection-based software lifecycle process. He was able to show that performing software inspections during development slightly increases programming effort, but decreases later effort and schedule during testing and integration. Whether there is an overall savings in project effort resulting from that trade-off is a function of development phase error injection rates, the level of effort required to fix errors found during testing, and the efficiency of the inspection process. For typical industrial values of these parameters, the savings due to inspections considerably outweigh the costs. Dynamics-based techniques are particularly good for planning and control, but particularly difficult to calibrate.

6. Regression-Based Techniques

Regression-based techniques are the most popular ways of building models. These techniques are used in conjunction with model-based techniques and include "Standard" regression, "Robust" regression, etc.

6.1 "Standard" Regression – Ordinary Least Squares (OLS) method

"Standard" regression refers to the classical statistical approach of general linear regression modeling using least squares. It is based on the Ordinary Least Squares (OLS) method discussed in many books such as The reasons for its popularity include ease of use and simplicity.

The OLS method is well-suited when

(i) A lot of data are available. This indicates that there are many degrees of freedom available and the number of observations is many more than the number of variables to be predicted. Collecting data has been one of the biggest challenges in this field due to lack of funding by higher management, co-existence of several development processes, lack of proper interpretation of the process, etc.

(ii) No data items are missing. Data with missing information could be reported when there is limited time and budget for the data collection activity; or due to lack of understanding of the data being reported.

(iii) There are no outliers. Extreme cases are very often reported in software engineering data due to misunderstandings or lack of precision in the data collection process, or due to different "development" processes.

6.2 "Robust" Regressions

Robust Regression is an improvement over the standard OLS approach. It alleviates the common problem of outliers in observed software engineering data. Software project data usually have a lot of outliers due to disagreement on the definitions of software metrics, coexistence of several software development processes and the availability of qualitative versus quantitative data.

There are several statistical techniques that fall in the category of 'Robust" Regression. One of the techniques is based on Least Median Squares method and is very similar to the OLS method described above. The only difference is that this technique reduces the median of all the r^2.

i

7. Composite Techniques

As discussed above there are many pros and cons of using each of the existing techniques for cost estimation. Composite techniques incorporate a combination of two or more techniques to formulate the most appropriate functional form for estimation.

7.1 Bayesian Approach

An attractive estimating approach that has been used for the development of the COCOMO II model is Bayesian analysis is a mode of inductive reasoning that has been used in many scientific disciplines. A distinctive feature of the Bayesian approach is that it permits the investigator to use both sample (data) and prior information in a logically consistent manner in making inferences. This is done by using Bayes' theorem to produce a 'post-data' or posterior distribution for the model parameters. Using Bayes' theorem, prior (or initial) values are transformed to post-data views. This transformation can be viewed as a learning process. The posterior distribution is determined by the variances of the prior and sample information. If the variance of the prior information is smaller than the variance of the sampling information, then a higher weight is assigned to the prior information.

Table 3. Prediction Accuracy of COCOMO II.1997 vs. COCOMO II.2000

COCOMO II	Prediction Accuracy	Before Stratification	After Stratification
1997	PRED(.20)	46 %	49 %
	PRED(.25)	49 %	55 %
	PRED(.30)	52 %	64 %
2000	PRED(.20)	636 %	70 %
	PRED(.25)	68 %	76 %
	PRED(.30)	75 %	80 %

8. CONCLUSIONS

This paper has presented an overview of a variety of software estimation techniques, providing an overview of several popular estimation models currently available. Experience to date indicates that neural-net and dynamics-based techniques are less mature than the other classes of techniques, but that all classes of techniques are challenged by the rapid pace of change in software technology. The important lesson to take from this paper is that no one method or model should be preferred over all others. The key to arriving at sound estimates is to use a variety of methods and tools and then investigating the reasons why the estimates provided by one might differ significantly from those provided by another. If the practitioner can explain such differences to a reasonable level of satisfaction, then it is likely that he or she has a good grasp of the factors which are driving the costs of the project at hand; and thus will be better equipped to support the necessary project planning and control functions performed by management.