Title: Estimation Issues in Software Project Management

Course:
Year: 1989
Author(s): J. Johnson, B. Schroeder and M. Veeramoney

Report No: P89018

Abstract: This report studies the estimation techniques used for sizing software development efforts and identifies significant factors that must be considered when estimating a software project. The issues identified in the report can be categorized into four areas: project goal, project complexity, work environment and human environment. The results of the report indicate that the existing estimation models need to reflect human estimation parameters. Estimation is typically performed based on past experience. A knowledge based system is a very useful tool for estimation.

ESTIMATION ISSUES IN SOFTWARE PROJECT MANAGEMENT

J. Johnson, B. Schroeder,
and M. Veeramoney

EMP - P8918

ESTIMATION ISSUES IN SOFTWARE PROJECT MANAGEMENT

PROJECT EMGT 541

Murali Veeramoney
Bryce Schroeder
Jerry Johnson

TABLE OF CONTENTS

i. Abstract

An attempt is made to study and survey the estimation techniques used for sizing the software development effort. The objective is to identify the significant factors that should be considered while estimating the software development process and to collect in one place the list of these factors. A combination of literature search and a questionnaire survey with the professionals in the field is performed to analyze the problem. The estimation issues fall into four broad areas: project goals, project complexity, human environment and work environment. The results of the project indicate that the existing estimation models need to reflect human estimation parameters. Further, the models should be tailored to specific software design areas. In the industry estimation is performed based on past experience. Hence, a knowledge based system that has the ability to predict the current estimate based on past estimates would be a very useful tool.

I. Introduction

Problem definition

Every software engineering manager has a need to estimate accurately the schedules of the project he/she is responsible for. It is has been our experience that it is often difficult to estimate the software development costs. Furthermore, some of the time these estimates are erroneous and capricious. We believe that this problem exists because there are many factors that effect the software development process and many managers are not aware of (1) the factors themselves and (2) how these factors effect the development process. We undertook this project to investigate and identify some of the factors that should be considered to estimate the software development process.

Our objective, is to collect in one place the list of factors that need consideration while estimating software costs. The process of formulating a theoretical/empirical/mathematical model that uses these factors is out of scope of this project.

We, however, believe that for the industry, understanding the software development process together with understanding the factors that effect this process, is the first step for an engineering manager to get a better handle on the estimation process. Our hope is that a software engineering manager will keep these factors in mind, and perhaps add to them, and utilize them while estimating the development process.

Problem Approach

Our approach towards the problem was two pronged. First, we attempted to understand the nature of the software development process and what methods existed for developing software. Second, we attempted to identify the factors by:

a) Literature Search: Several excellent articles have been published in the area of estimating software development costs. Some these articles are listed in the bibliography of this document. There is a clear indication of continued research in this area. Considerable amount of research activities concentrate on the software engineering process itself. As of today, this process is not understood and understanding of this process is the key to estimating it. Hence, research on the software development process and the methods for software development is relevant for the estimation process.

Most of the research use and/or formulate mathematical models to estimate software development. As mentioned above, we are not interested in the models but the factors that are used by these models. Hence, we have sifted through these models and identified the key factors that are primarily considered for estimation.

b) Survey: We formulated a questionnaire to survey managers responsible for estimating the software development process, in the Portland suburbs. Further, a copy of the questionnaire was published in the USENET bulletin board throughout the US. A copy of the questionnaire is available in the Appendix for reference. A summary of the answers is provided elsewhere is this document. The objective for this survey is to identify the techniques and factors used by people in the industry.

c) Experience: Though, we have attempted to disregard our experience in this subject, we believe that in a project of this nature we have could have been biased by own experience. Such bias is unintentional.

II. Development Methods and Tools

Some of the basic philosophies of program design embody step-by-step instructions to be performed manually. Other philosophies of development implement their methods with semi-automated, computerized methodologies that provide a language for expressing the system or program design requirements. This allows a method of prompting the designer interactively, while still in the program development process, design errors are detected. The semi-automated systems also provide various listings and analytical reports which may be used for control purposes.

The first step in estimating software development costs is to size the work to be done. This requirements definition and planning process is not the focus of this paper, however certain considerations deserve some mention. The software under-sizing problem is one of the most critical roadblocks to accurate software cost estimates. Tarek K. Abdel-Hamid and Stuart E. Madnick take note that, "Past experience indicates a strong bias among software developers to underestimate the scope of software projects.". Tom DeMarco gives a possible reason for this bias in his statement, "When your ego is involved in performance of a task, your ability to make a reasonable assessment of how long that task will take you is impaired." [5] This bias to underestimate the size of a program development project can lead to a geometrically increasing error factor. Ware Myers, contributing editor of IEEE Software journal, demonstrates that "productivity in terms of source statements per man-month declines rapidly as system size increases." [18]. Errors in project sizing can lead to unexpected additional costs, especially when pressure to make deadlines increases.

Studies indicate that the production of software involves "diseconomies of scale," [29]. This means that doubling the size of a software product more than doubles the cost of producing it.

Barry W. Boehm suggests some conditional probabilities that should be included in an expected cost model, which he called the value-of-information approach. By this approach, Boehm suggests ways of obtaining information to reduce uncertainty about the likely success or failure of a project. His five value-of-information guidelines to investing more in additional information are:

1. There exist attractive alternatives whose payoff varies greatly, depending on some critical states of nature.
2. The critical states of nature have an appreciable probability of occurring.
3. The investigations have a high probability of accurately identifying the occurrence of the critical states of nature.
4. The required cost and schedule of the investigations do not overly curtail their net value.
5. There exist significant side benefits derived from performing the investigations.

That is, it is cost-effective to investigate significant alternatives, and it is possible to "calculate the expected payoff from a software project as a function of our level of investment in a prototype or other information-buying option." [2].

Boehm suggested five pitfalls to avoid by using his value- of-information approach. The commonly recommended procedures that he suggests managers avoid are:

1. Always use a simulation to investigate the feasibility of complex real-time software.
2. Always build the software twice.
3. Build the software purely top-down.
4. Every piece of code should be proved correct.
5. Nominal-case testing is sufficient.

By following Boehm's [5] recommendation of avoiding these suggestions in the software development process may cause reductions in the project time and cost estimates.

Yeh [31] has suggested that program development techniques are inadequate in the following areas:

1. The program requirements could not be fully described.
2. There is no provision for reusing code.
3. There is a lack of tools for the critical requirements and design stages.

Merwin stated that although there are numerous technical tools and techniques which contribute to the production of reliable software, " what is still missing is the overall management fabric which allows the senior project manager to understand and lead major data processing development efforts." [9]. Fathi and Armstrong point out "problems that tend to complicate the development process are underestimation of software complexity, misinformed management, confusion of terminology, lack of comprehensive integrated support tools, lack of standards for language implementation." For the development of microprocessor software, it is recommended that techniques and methodologies used in development of software for larger machines be used [9].

Design Approach and Methodologies

The Software Life Cycle is a common model used in software estimating. It is used as a basis for describing the phases and the economic implications of each phase of software development. The Software Life Cycle is used to divide the software project into manageable sub-projects, with the completion of each phase or sub-project indicated by the completion of a milestone end-of- phase review. Excluding the program feasibility study and the software requirements definition, the typical phases are the product design, a detailed design, the coding and testing, integration and testing, implementation and systems testing, and maintenance. Some methodologies include enhancements and other program changes in the maintenance phase.

Wasserman notes that the new design philosophy of software engineering was that increased effort in the earlier stages of development can reduce testing and maintenance costs. If a software requirements error is detected and corrected during the planning and requirements phase, its correction is a relatively simple matter, but if the error is not corrected until the maintenance phase, the correction involves a much greater cost. Wasserman concludes that this design philosophy is having an effect in the following ways:

1. Organizations are identifying the design process as a unique phase in the development process.
2. Organizations instituting formal reviews of specifications and designs in an attempt to detect potential errors at an earlier stage of the development cycle.
3. The cost distribution for the phases of the software life cycle is changing in that specification and design phase costs were about 50% and testing cost was about 30%.

This compares with the 1970's estimates of 30% for analysis and specifications, 50% for testing, and 20% for design and coding [9].

Cost estimation techniques can be considered as falling into seven major categories: algorithmic models, expert judgment, analogy, Parkinson, price-to-win, top-down, and bottom-up [2]. Algorithmic models output cost estimates as a function of cost drivers, of which thirteen are used in the SDC model. SDC was developed by System Development Corporation and considers lack of requirements, stability of design, percent math instructions, percent storage/retrieval instructions, number of subprograms, programming language, business application, stand- along program, first program on computer, concurrent hardware development, random access device used, different host-target hardware, number of personnel trips, and development by military organization. The TRW Wolverton model is another algorithmic model which uses software cost per object instruction as a function of relative degree of difficulty. Expert judgement is critical in the estimation process, but is not sufficient by itself to provide accurate estimates. Some projects require outside consultants to perform design and estimating functions. Analogy is useful in determining lines-of- code, which forms the basis of many estimating models. Analogy is especially useful in establishing software design functionalities in research and development projects. The Parkinson law refers to the fact that projects will expand to fill all available time. Price-to-win refers to the approach of estimating the project cost at a level necessary to be awarded the development contract. Top-down refers to design structuring where global software properties and requirements are determined and divided into manageable sub-projects, each of which develops a program module; the developed modules are then integrated into a functional program. Bottom-up refers to the estimation of separate modules independently, with these estimates being aggregated into one overall estimate.

N. A. Vosbury of System Development Corporation analyzes several design methodologies, the use of which will improve productivity and reduce software development costs. Among the existing design tool methodologies they cite are top-down design, structured design, the Jackson method, SADT, and composite- structured design. Among the automated systems they list are PSL/PSA, SSL, PDS, HOS, and designer/verifier assistant [29].

Structured design is necessarily top-down in that it involves analysis of the functions of a process and their refinement into small modules. The key design concept is the module with the goal of modularization being "a high level of strength [in each module] and a low level of coupling." [29] Module strength has to do with singleness of purpose, while module coupling defines that amount of interaction between modules. The strategies for decomposing processes into modules include source-transform-sink (STS), transactional, and functional. STS iterates on four steps: outlining the problem structure in terms of data and operations, identifying the streams of data and sequencing the operations on them, finding points in the data streams where the data becomes useless, and using these points of uselessness to divide the problem into functions. The transactional approach determines the needs by various data types or input records. The transactional approach is useful in transaction driven business systems. The functional approach is used in the design of data base management systems,and compilers. It isolates single functions to become immediate entries in an information strength module.

The Jackson method assumes that the nature of the data determines the characteristics of the functions. Three steps are involved: design the structure of the input data, design the structure of the output data, and define the correlation between the input and the output. This design is done top-down, and a program development language is later selected for the code.

HOS/AXES is a sophisticated automated methodology from Higher Order Software, which is based on the language AXES and builds upward from a set of primitives. INA JO is another auto- mated methodology from System Development Corporation. INA JO includes a requirements modeling stage, a design specification stage, a verification of specifications stage, a program design specifications stage, and verification of implementation stage. SSL provides a language and translator that perform design consistency checks on the decomposition of a process to be solved on a virtual machine. SSL is from Science Applications, Inc. of Huntsville, Alabama.

HESS (Hybrid Expert Simulation System) introduced by Levary and Lin, uses four subsystems: the simulator (software life cycle simulation), an input expert system, an output expert system, and a knowledge-based management system. The objectives of simulating the software development life cycle process are:

1. to experiment with the model in order to text various management actions, policies, options, and procedures,
2. to test the impact of various assumptions scenarios, and environmental factors,
3. to predict the consequences of management actions on the components and flows,
4. to examine the sensitivity of the process to various internal and external factors.

The input expert system is used to check the compatibility of the components, E.g. project size for each phase, estimated schedule plan for each phase, cost per person work year, personnel hiring delay, average staff productivity rate, nominal attrition rate, staff training, error generation rate and error detection rate. The output expert system makes recommendations regarding the software development process characteristics or process modifications based on the output from the simulation, the characteristics of the input, and the initial design of the system. The knowledge base is composed of input data and the set of associated recommendations. The knowledge base eliminates the possibility of reproducing simulation runs, stores information for future use, and inspects the knowledge based records [15].

A set of five different life cycle models was proposed by Alan Davis, Edware H. Bersoff, and Edward R. Comer. Their goal was to provide new, standard methodologies for five different types of software estimating in order to improve reliability in the software life cycle. Automated software synthesis describes a family of tools that transforms requirements or high-level design specifications into operational code. Some of these products translate high-level code into machine code. Incremental development tools provide a way to design and implement an operational system in parts, adding increased functionality or performance at each stage. The rapid throwaway prototyping approach addresses the issue of ensuring that the software product being proposed really meets the users needs; a "quick and dirty" partial implementation of the system is used to help determine user requirements. In evolutionary prototyping, the developers construct a partial implementation of the system which meets known requirements and use the prototypes to learn more about the future system requirements. Reusable software is the discipline of attempting to reduce development costs by incorporating previously proven designs and code into new software products [4].

Yeh [31] identified some reasons for the low utilization of various software tools, methodologies and techniques. These include:

1. the current corporate management has little experience with the tools and is not sympathetic to their use.
2. there is no centralized corporate entity that provides the tools.
3. the learning curve cost compared to long term benefits is high.
4. most companies have little time to apply new methodologies to new programs, because they are busy maintaining old code.

The design methodologies and automated tools all have six elements in common:

1. organizing.
2. functional decomposing.
3. data-flow analyzing and decomposing.
4. documenting.
5. requirements tracing.
6. interface checking.

Any good design approach will include the above functional areas. Some features to look for in modern design tools include:

1. the number of detailed decision points in the design language should be less than the number of decision points in the implementation language.
2. the level of desirable transparency should rise.
3. complex data structures should be handled as entities.
4. data and associated operations should be handled as entities.
5. functions may be generalized and manipulated.
6. documentation should be enforced.
7. the language should enable incremental design verification.
8. organization and management visibility should be emphasized.
9. the transition from requirements to design to implementation should be smooth with automated evaluation. [29]

III. Quality and Reliability

Quality and reliability are two factors which can greatly increase the development time needed for completion of the pro- gram. Quality refers to the absence of programming errors, while reliability measures the extent to which the program performs its intended functions. Quality and reliability problems increase with the complexity and length of the program. Reliability measures vary from low to high, with high reliability demands being made only where the effect of a software failure can be a major financial loss or massive inconvenience. Some results of program failure include additional debugging and testing costs, rework of specifications, code rewrite, rework of one or more prior software development phases, and lost opportunity owning to delayed product introduction or deployment. At worst case, it may result in loss of human life.

Appraisal costs are incurred during product development for verification and validation. After each phase of the development cycle, verification ensures that cycle incorporates the intentions of the previous step. Validation comprises those techniques used to ensure that the product functions correctly and contains the features prescribed by the requirements. Appraisal includes testing, simulation, management reviews, in- process development inspections, technical audits, and compliance audits.

Current software engineering management should be concerned with three issues central to having a successful quality program:

1. The ability of the company's accounting program to present appropriate cost summaries.
2. The capability of identifying all costs associated with the quality function.
3. The quality function's ability to control quality costs [12].

Hollocker describes formal organizational control functions to ensure quality. Quality assurance includes manual maintenance and overall program support, including definition of data acquisition, analysis and reporting standards, and monitoring operation of products in use. Quality control includes program support at the product center, including data collection and analysis. Process control includes project element tracking and coordination, status and progress reporting, acquisition analysis, and productivity reporting. Configuration management refers to definition and maintenance of standards for configuration item identification and change, and change control includes general support to the technical staff and coordination at the product center level [12].

Quality software design leads to quality software development, and investments in requirements validation activities improves project efficiencies. Clear definitions of program functionalities and specifications help avoid the frequent misinterpretations, underestimates, over expectations, and outright buy-ins which can plague a software development effort. Barry Boehm comments that "Until a software specification is fully defined, it actually represents a range of software products, and a corresponding range of software development costs." Estimates are that some software organizations spend as much as 70% of their time correcting errors and in making modifications and enhancements to existing programs [29]. Some project may use prototypes (mentioned above), which provide data for complete specification and estimation of the full-blown project.

Some developers refer to quality and reliability problems as defects, where a defect is defined to be any deviation between the planned execution and the actual execution. If zero defects are allowed, the development time will certainly be longer than if some defects are allowed. If no bugs are to be allowed in the program, where a bug is an accidental flaw in the program code, then time is needed for follow-up work in removing all errors from the code.

Software quality has been defined by Tom DeMarco as "absence of spoilage." [5]. In general, quality can be defined as the cost to attain some predefined level of defects and bugs, or in other terms, quality can be defined as the cost to attain some predefined level of absence of defects and bugs. Thus, a quantifiable measure of quality is:

QUALITY = (diagnosis cost + correction cost)/program volume.

Defect clustering is a phenomenon that has been reported in software development projects. This refers to the fact that most of the defects will occur in clusters, that is, modules that have been found to contain some defects will probably contain more. If the defect rate has been found high in a module, it may be cost effective to simply rewrite the module, rather than attempt to devise tests to locate defects and then correct and re-test the code. T.C. Jones's data on IBM's IMS product shows that of the 425 modules that make up the product, 300 were defect free, while 7.3% of the modules accounted for 57% of the defects [13].

T. C. Jones [13] reports some typical software quality data from different sources:

1. 10-50 defects per thousand lines of executable code average for American-produced code.
2. 0.4 defects per thousand lines of executable code target quality for major American manufacturer.
3. 0.2 defects per thousand lines of executable code average for Japanese code.

Estimation of a company's average defect rate can be determined from historical data. A profile of defect rates can be determined for each employee to assist in estimating the costs of defect detection and correction. A company Metrics Department can determine cost factors for product defect rates based on the type of product and the project teams involved in the development work.

On-line testing uncovers only about fifty percent of the defects, so that structured walkthroughs and batch testing can still be important [5]. Having a separate testing and debugging department can improve efficiency in defect detection, correction, testing and debugging. It is DeMarco's experience that "if you want to take advantage of the efficiency of an on-line testing tool, you have to take it out of the hands of the coder." [5].

Modern quality assurance support tools can be divided into two categories: static tools and dynamic tools. Static analyzers examine programs systematically, while code and standards enforcers force adherence of single-statement and Multiple-statement syntax to defined rules. Some tools produce proforma source modules, highlighting syntactical errors.

IV. Manpower

Because of the lack of experienced personnel both in the management and technical aspects of software development, many companies are resorting to using less qualified people, which results in projects being late and over budget. Fathi and Armstrong point out that companies deal with the talent shortage by stretching existing staff and setting up ways to motivate, educate, monitor and assist people who are performing unfamiliar jobs [9].

Anticipated project costs can increase due to unforeseen events, such as employee nonproductive time. Some project loss factors which should be considered that are based on employee nonproductive time can have a significant influence on the completion date and budget. Some items to consider are coffee breaks, non project related phone calls, washroom breaks, travel time, meetings, office parties, availability of required tools and equipment, changes in work procedures, learning curve effects, personal fatigue, weather conditions, misinterpretation of activity definition, efficiency of change control system, relocation time for new employees, availability of support functions, communicating instructions, maintenance, setup and change-over, and inoperable equipment [25].

Nonprofit loss factors can result in lost time, but may not be costed directly to the project. Such factors as vacations, employee turnover, sick leave, attendance at seminars or meetings not related to the project, maternity leave, and holidays may significantly affect the completion schedule for the development effort [25].

When mistakes and errors occur and rework is necessary, the experience and ability of the individuals performing the tasks becomes important. Maintenance programming and testing and debugging can be handled more efficiently by a specialized maintenance staff. Kerth has staged that "Good senior software engineers are hard to find. Nevertheless, they are very important to the success of a project. Their guidance and ability to solve difficult problems rapidly is critical." [9].

Productivity

Several factors affecting programming productivity were listed by Magers, including:

1. vague hardware specifications.
2. changing hardware designs.
3. lack of experience with the application.
4. no previous experience with the microprocessor.
5. low-quality or no software development tools.
6. inadequate time alloted for software design, testing and documentation.
7. underestimation of the amount of memory required.

Because of these factors, Fathi and Armstrong recommend that "it is always safe to double the size of the initial memory estimate." [29].

Software estimating models regard the effectiveness of the programming staff by using a productivity factor in the basic estimating relationship:

Product = Effort * Time * Productivity

where productivity is based on empirical observation. Since an historical relationship is used to evaluate the effectiveness of the development organization as a measure of productivity, the experience of the estimator in working with the staff should be considered an important factor in the accuracy of the estimate. The availability of workstations, software tools, and other capital equipment, and the extent to which the staff has received adequate training in their use, all enhance the productivity of the staff. Some authors have identified as many as 100 factors that affect productivity.

CASE tools assist in the development process and enhance productivity. A curve relating software productivity to percentage of support software available was developed by System Development Corporation for the U.S. Army. The curve has a productivity range of 1.93 between its 30th and 70th percentiles, and a range of 2.66 between its 20th and 80th percentiles. The Constructive Cost Model (COCOMO) developed by Barry Boehm estimates the productivity range is 1.49.

The basic Putnam equation for evaluating product size is based on productivity, effort, and time. It is:

Product size = (Effort)**0.33 * (Time)**1.33 * Productivity

[18]. This relationship is limited to systems of no more than 2,000,000 lines of source code. Thus, larger systems will have to be partitioned into smaller systems of 2,000,000 lines each, and an integration factor included -- 50% of the original estimates is added by Myers for integration of subsystems. Myers found that staff time required an additional 30% when integration was required. The Putnam equation explicitly treats estimation risk and uncertainty, and can be used for estimation of cash flow, major-milestone schedules, reliability levels, computer time, and documentation costs as well as manpower. "There appears to be a minimum development time for a given project at a given organizational-productivity or cost-driver level below which it is unlikely to be completed successfully no matter how much labor effort is applied to it." [18].

Researchers have also noted that software costs increase when too tight of time limits are placed on the process, since added time for debugging and redesign efforts must later be included. Abdel-Hamid and Madnick concluded that their "Research findings indicate that decisions people make in project situations, and actions they choose to take, are significantly influenced by pressures and perceptions the project schedule produces.".

Some estimating techniques use an effort/schedule trade-off technique to estimate manpower requirements, while other techniques begin with an estimate of the lines-of-code. The functional model uses an effort/schedule trade-off approach. Most of the recent software cost estimation models tend to follow a scaling equation of the form

$$\text{Man-months} = c\ (\text{KIDS})^{x}$$

where KDSI represents thousands of delivered source instructions. Fathi and Armstrong give the figures Magers uses for estimating lines-of-code. They are:

1. 15 to 20 lines per day using a high level applications language.
2. 8 to 12 lines per day using assembly language in a microprocessor program.
3. Real-time microprogramming will average 2 to 5 lines per day.

The Bailey-Basili meta-model uses a variation of the lines- of-code model[1], but adds a factor for methodology level and complexity. The Grumman SOFCOST model, the Tausworthe Deep Space Network and COCOMO use an estimate of lines-of-code. COCOMO also uses a hierarchy of three increasingly detailed models which range from a single macro-estimation scaling model as a function of product size to a micro-estimation model with a three-level work breakdown structure and a set of phase-sensitive multipliers for each cost driver attribute. The model begins with an estimate of the product-size in delivered source instructions in
thousands. A set of fifteen cost driver attributes are then used to determine development effort. These cost drivers fall into four basic categories: product attributes, computer attributes, personnel attributes, and project attributes. Project attributes use factors discussed in the development section of this paper, E.g. software tools, etc. Personnel attributes include programmer capability, programming language experience and analyst capability. Computer attributes include mass storage constraint and computer turnaround time. Product attributes include data base size and product complexity [2].

The Putnam-Norden manpower equation assumes that one already has a forecast of the man-months of effort required for the project and that one now needs to determine how manpower requirements vary over time. The Putnam-Norden manpower loading equation as given by DeMarco is: [2]

$$\text{Manpower}\ (t) = 2Kat\ \text{expel}\ (-at\ )$$

where a is the acceleration factor (a shaping parameter that establishes the initial slope of the curve), and K is the total manpower cost in man-months. Since the equation is based on a lines-of-code metric for volume of work, it is not recommended. Within a given domain, acceleration should not vary much. The derived time to deliver is:

$$\text{Time to deliver} = 2.5 * (\text{MM})^{.33}$$

where MM is the estimated man-months. This equation is used by TRW, IBM, and RADC [2].

Some typical methodologies for estimating system or program development costs use a Rayleigh curve representation of the software life cycle. The Rayleigh curve (see Illustration 1) is a well-known model for estimating project costs which was based on an analysis of project data collected by Putnam and Norden at the Army Computer Systems Command [23]. The Rayleigh curve looks like a probability distribution curve with resource consumption distributed heaviest during the middle of the project life. An alternative theoretical model was developed by F.N. Parr, which rates the rate of progress which can be achieved in developing software to the structure of the system being developed [21]. One specific concern of Parr's hyperbolic secant model is the evaluation of whether the derivation of a work profile is the rule for deciding whether the most recently solved problem makes other work visible. This can make a major difference in the research and development environment, since planning and requirements definition is difficult at best.

Changes in staff level will affect both schedule completion time and project cost, but reach a point where additional manpower does not reduce costs or time. Any change in staff which necessitates the hiring and training of new personnel can cause delays that may be extremely difficult to estimate. The SD (System Dynamics) model, developed by Forester, considers such factors as physical, social, and informational flows through the system, and uses feedback loops and non-linear interrelationships to aid in project control. System dynamics attempts to deal with various management actions, policies, options, and procedures by testing the impact of various assumptions, scenarios, and environmental factors on the model.

V. Human Environment

As a group, engineers and scientists have a high need for achievement (n_Ach) as shown in a study done by Stahl [26]. It is this need for achievement that creates a distinct set of motivational needs that seem to be unique to the engineering environment. In three separate studies, done by Thamhain [27] [28], Orpen [20], and Griggs and Manring [11], a set of distinct motivational needs was identified for engineers and scientists. All three studies came up with the same needs as being most important: the need for challenging work and a stimulating work environment. Other needs identified include: professional growth, overall leadership, management direction, tangible rewards, job security, open communication, good interpersonal relations, and clear role definition. Five of these needs are of special concern as they relate to Software Project Estimation. They are:

Open Communication.
Challenging Work and a Stimulating Work Environment.
Overall Leadership and Management Direction.

Tangible Rewards.
Job Security.

Open Communication

Open communication as defined by Thamhain [27] is, "the free flow of information both horizontally and vertically, keeping employees informed of technical and organizational developments." This applies not only to the immediate management, but to the entire organizational structure.

In a large firm, all the employees may not be in a central location. Corporate newspapers, executive status letters, and electronic mail, can go a long way in keeping employees informed. The vertical hierarchical organizational structure can experience delay and distortion on policy and information coming down the chain of command. Frequent meetings between the Executive level of management and the line will ease rumors and create a more open flow of information. A project based organization, can be impacted by duplication of effort. This can happen by there being no clear means to see what engineering achievements can be shared in other similar applications throughout the company.

"The more negative the feedback, the less likely it is to be given, and more likely to be distorted or delayed." [24] Rumors of a negative corporate change, such as layoffs, can be detrimental to productivity. Keeping communication open from top to bottom of an organization will prevent productivity loss, as well as increasing the confidence in management.

Challenging Work and a Stimulating Work Environment

Engineers and Scientists are motivated by the nature of the work itself [11]. Being on the forefront of technology, designing what no one else has designed, creating something out of nothing, all this appeals to the high achievement need of an engineer. The key then is to provide and define projects that can meet the engineer's need for challenges [11] [27] [28]. Studies have found these projects to have the following characteristics:

- challenging work.
- the work leads to professional growth.
- the engineer is involved in the decision making process.
- good leadership, proper planning, project goals.
- clarity of roles.
- a cohesive team, with mutual trust, and open communication.
- tangible rewards.
- sufficient resources (financial and personnel).
- sense of accomplishment.

The work environment supports the project and fulfills those qualities not directly related to the project. "A work environment which is professionally stimulating and challenging, fulfills the esteem needs of the people such as recognition, accomplishment and pride; people are involved, motivated and interested in the work itself. The work environment is described by the structure of the organization, its facilities and management style." [27]

Overall Leadership and Management Direction

Gooch and McDowell [10] maintain that managers must come to the realization that they are not motivators but "change agents". As change agents a manager has the following duties:

1) Create an understanding of the desired action.
2) Make the employee aware of desired change in relation
to current practices.
3) Facilitate answers to who, what, when, where,
why, and how questions related to desired action.

This does not mean the manager cannot use other motivational techniques. Mutual feedback, between the manager/project leader and the engineer or scientist is essential in maintaining a supportive environment (Achievement Theory) [14]. Goal setting theory can be applied to stimulate productivity. It has been especially successful when the person involved is allowed to have input in setting the goal [14] [7] [28].

The engineering manager plays a very important role in the effectiveness of the engineering staff. Promotion of engineers to engineering manager, that have not shown high needs for power (n_Pwr) should be discouraged [26]. Upon becoming a manager, a high achievement, low power need engineer will be ineffective at delegating and facilitating. The problem arises in that the engineer who has a high need for power may not be considered a "good" engineer and therefore will not be promoted. Engineering managers must be technically credible, good problem solvers, goal setters, and able to give recognition. Above all the engineering manager must be able to define and keep a sense of direction and purpose in the engineering team [28].

Tangible Rewards

Money is not a prime motivator of engineers [26] [27]. However, money and title (e.g.. SOFTWARE ENGR III) do give status to the engineer. Comparison of the same job (status) at another firm or the same firm where there's more money being payed can have a de-motivating effect (Equity Theory) [14] (Herzberg Hygene Theory).

A Profit Sharing System for a growing, profitable company, can provide incentive for the engineer. However, if the firm is not stable or barely making a profit, there is no longer a personal sense of ownership of profits being made. The engineer cannot see his contribution making a difference.

Drennan's [7] hypothesis, based on two motivational studies; when companies limit their total amount of merit award (salary increases) year after year, that company policy gets blamed for low raises for high performers. This can be a primary source of apathy directed toward corporate management. Instead of continuously rewarding for past performance, Drennan suggests incentives be set for reaching future goals. The incentives should be structured as follows:

1) They should motivate on a broad front, encourage everyone.
2) Money incentive should be endlessly renewable.
3) Money should be payed in a lump sum.
i.e.. $500 payed in one lump sum is more effective than
$1000, payed out over the whole year.
4) "Winning posts" should be clear and unambiguous.
5) System should encourage appraisals.
6) Avoid de-motivation.

Job Security

Thamhain and Wilemon, while conducting their studies relating to the motivational needs of engineers [27] [28], had an unexpected finding: the primary motivation of engineers who have job security is their work, those engineers who do not have job security (whether real or imagined), their primary motivation is to achieve job security.

VI. Program Complexity

The complexity of the project is one of the major factors that need attention for estimation. Complexity is defined by Basili [1] as the measure of the resources expended by another system in interacting with a piece of software [1]. Further, there are two types of complexity: process complexity and product complexity. Process complexity deals with the complexities involved in the process of developing software. Here the metrics are based on time to develop, number of errors, etc. Product complexity is the inherent complexity of the software product to be developed. Basili [1] classifies product complexity into (a) size and (b) structure. One of the common metrics for size is lines of code [2]. Lines of code includes both code and comments. Almost all estimation models uses an estimate of the number of source or object instructions to be developed, and this is quite difficult to quantify.

DeMarco [5] provides an informal model of estimating the structural complexity. He argues that the software must follow a specify design format based on the specifications. Hence, the estimates of the structural complexity is based on the properties of the software specifications and the requirements. He outlines a design methodology that describes the characteristics of the software, in terms of its data elements, input elements, output elements, functional primitives, transitions between states, relations, modules, data tokens and control tokens. Thus, viewing the software in terms If the above characteristics is an informal model for estimating its complexity.

As seen from above, the specifications of the software is an important factor in estimation.

Putnam [22] identifies imprecise and continually changing system requirements and specifications as one of the ever present problems of software. Since it is very important to understand what is the nature of the software to be developed, the lack of clarity of requirements must be factored in for estimation. Barry Boehm [2], terms the change in the software requirements as "requirements volatility and observes its significant impact on productivity. He points out that some large projects with frequent changes in requirements have experienced productivity penalties of factors greater than four. Lack of clarity of requirements has, therefore, a direct effect on the complexity and on the productivity of the software project. The important point is that when the product that is being built is not known then it is impossible to estimate how to build it.

VII. Interview Summary responses

A questionnaire was developed for the purpose of conducting interviews/surveys. Interviews/Surveys were conducted with a variety of respondents, who had worked on many different types of software projects. Software projects included; Fourth-Generation Language, Operating Systems, Computer Aided Software Engineering Tools, Unmanned Space Module Control, Diagnostics, Business Information Systems, Artificial Indolence, Compilers and other development tools, and embedded systems. What follows is a summary of the findings from the survey. A copy of the questionnaire can be found in Appendix C. Copies of the respondents answers can be found in Appendix D.

The Interview Survey projects' size and budgets cover a wide range. Respondents had worked on projects as small as a one person/one man month project up to 100 people spanning many years. The budgets ranged up to 100 million. Target customers for the projects were commercial, military, as well as private.

Identified Factors

Many important factors in estimating software projects were identified by the interview survey respondents. Factors are listed in the hierarchical order. The most frequently identified factors first. Each factor is followed by a brief explanation.

Complexity/Novelty

The complexity and novelty of the project, was whether or not the project or something similar had been done before, and the technical difficulty of the project.

Staff Characteristics

Staff Characteristics included; the amount of experience of the staff, how quickly staff members could tackle a new project and climb the learning curve, the ability to mesh individual personalities to form a project team.

Clarity of Project Requirements

The respondents generally held that unclear, incorrect, or misinformation in the project requirements would have a sever effect on the project outcome. Several respondents mentioned that they always had a requirements walkthrough at the end of each phase in development in order to determine whether the project still met the requirements, and whether or not the requirements had changed.

Development Environment

The development environment included such things as; the host computer, the availability of software debug tools, software design (CASE) tools, project implementation language, and the availability of prototype hardware.

Performance Constraints

Speed and Size were the two most mentioned performance constraints. Most respondents said that performance constraints were dictated by the customer or by the target hardware.

Size of Project

The size of the project, typically the number of lines of code to be written, tested, and debugged, was one factor few respondents gave consideration. In question eight of the survey, interviews were asked if they used lines-of-code as an estimation factor, the resounding answer was "no". On the whole, project size appeared to measured in functionality.

Human resource availability

The ability to add more manpower in the event of schedule slippage, was a factor.

Support Commitments

Commitments to other or previous projects was a factor that concerned several respondents.

Target Audience

The complexity of the user interface changes based on the degree of sophistication of the end user.

Vacation/Holiday/Sickness/Turnover

Long term projects need to factor in sick time, vacation, and holidays. Turnover was mentioned as one of the prime factors in schedule slippage.

Number of Interruptions on Task

Some work environments are not conducive to concentrating on a single task for any length of time.

Phase of the Moon

One respondent commented that estimation of software projects might have a higher degree of accuracy, if the current phase of the moon taken into account.

Rule of Thumb Estimation

Interview respondents were asked what their "rule of thumb" estimation technique was. Answers ranged from "best guessing" based on experience to no method at All techniques were based on past experience with some "fudge" factor. One of the more interesting team estimating techniques assigned a portion of the project to each member, then each member estimated his portion. A percent fudge factor was added to each estimate based on the experience of that team member.

Software Lifecycle

Interview/Survey respondents were asked if they considered the software lifecycle (waterfall) when estimating a software project. Most of the queries returned said that they did indeed use some form of the software lifecycle when doing project estimation. It is interesting to note that several projects did not fit into the lifecycle model, they appeared to be research or maintenance based.

Accuracy of Estimation

Respondents were queried as to the accuracy of their estimations. None of the respondents said that they were on target with their estimates, although many were fairly close. The most typical reason for being off was due to some portion of the project that was not clearly enough defined or well understood. The next most popular answer was "due to factors beyond our control".

Action taken in the event of Incorrect Estimation

Respondents were asked what action was taken if an estimate was discovered to be incorrect. Answers varied from, cutting features and functions, to doing nothing and taking this into consideration on the next project. Answers to this question point to possible differences in management styles and corporate cultures.

Project Goals

Interviewees were asked what projects goals were optimized for, and given a choice of five; speed of project completion, development costs, manpower, quality and reliability, or a combination of the four. The majority of the respondents had project goals set out for them, rather than selecting the goals themselves. Speed of completion was the number one goal, followed closely by quality and reliability. The following sentence summarizes the responses nicely:

"The project has to be completed yesterday, have the highest quality possible, and with no staff or overhead cost".

VIII. Straw-man proposal for an estimation tool

It is apparent from the survey and the literature that there is no clear and correct answer for estimating the software development process. Academicians formulate a mathematical model based on certain metrics, which is primarily based on lines-of-code, and suggest to use this model for estimation. Our surveys indicate that such models, for. e.g. lines-of-code, are rarely used in practice. In the industry estimation is attempted via the ever popular "divide and conquer" rule. The development process is divided into phases and an attempt is made to estimate each of these phases. If a new project is in question, the water-fall model [2] is used to divide the development process into phases. Each of these phases and then sized by the individual using his/her own past experience.

It is our belief that, in practice, estimation of a software development process is based heavily on past experience of the individual in estimating similar projects. To aid the engineer manager in estimating a software development project, we propose a systematic heuristic approach to the problem. In this approach, the manager performs the following steps:

a) Divides the software project into the different phases.
b) Identifies the factor that need to be considered for each of these phases.
c) Using his/her knowledge base (experience) estimates the phase based

on the identified factors. That is, correlate the factors to a
project performed in the past with has similar characteristics.
d) Totals the estimates for each phase.

In order, to assist an engineering manager to estimate a knowledge base system can be developed that can use some "rules" to recommend a estimate. Of course, further research is required to identify how one can build such a knowledge base of past experience, how one can use a knowledge base and how one can add to the knowledge base.

```
                                       +-----------+            +-----------+
                                       | PROJECT   |            | PROJECT   |
                                       | ESTIMATES |            | KNOWLEDGE |
                                       | DATABASES |            |    BASE   |
                                       +-----------+            +-----------+
                                             ^                        ^
                                             |                        |
                                             |                        |
                                             |                        |
                                             |                        |
                                             V                        V
                  +--------------+         +-----------+            +-----------+
                  |              |         |           |            |           |
Software          |    Factor    | list of |           | estimates  |           |
          <-->|Identification|-------->|Co-relation|----------->|Adjustments|-+
Manager           |              | factors |           |            |           | |
                  |              |         |           |            |           | |
                  +--------------+         +-----------+            +-----------+ |
                                                                                  |
                                                                                  |
                                                                                  V
                                                                        Recommended
                                                                           Estimate
```

Figure 1. Knowledge based system for estimation

A straw-man design of one such a knowledge based system is outlined in Figure 1. Here, the individual interacts with the system to select the factors that effect a particular phase of the development process. Since the system is providing a list of all possible factors, it is unlikely that the individual will forget to include a factor. Further, the factor identification process can be customized to include other related factors. Once the project characteristics have been identified via the list of factors, the system consults a database to correlate this data to identify the estimates of a project with similar characteristics. Having arrived at the

ıe c'   c-
re pıojject
ules from

the soft-
r to look
etter than
the esti-
ssful esti-

intent of
hile esti-
consider
e identi-
omplexi-

deve
ıe inter-
oughout
vas per-
ant fac-

mation.
ped by
project
ing his-

survey
plexity
he sur-
ct life-
results
lexity.
timate
in the
The

n is a difficult task and, like software engineering, is definitely an art. Past experi-
ıe people play an important part in estimation. Hence, a knowledge based tool that
the past experience and characteristics of a software project and uses this knowl-
uggest the estimate of a current project could be used by a software engineering
for a first cut estimate. A straw-man proposal for such a system is proposed.

Appendices

A. Bibliography

Key: [#] - reference number for article.

[1] Basili, Victor R., Quantitative software complexity models: A Panel Summary, Survey results

[2] Boehm, Barry W., "Software Engineering Economics," IEEE Transactions on Software Engineering, Volume SE-10, No. 1, January, 1984.

[3] Coblentz and Gordon, "Plateaued People: A Present Problem," Training, August 1986, pp. 10 - 12.

[4] Davis, Alan M., Bersoff, Edward H., and Comer, Edward R., "A Strategy for Comparing Alternative Software Development Life Cycle Models," IEEE Transactions on Software Engineering, Vol 14, No. 10, October, 1988.

[5] DeMarco, Tom, Controlling Software Costs, Yourdon Press, 1982.

[6] DeMarco, Tom, Controlling Projects, New York: Yourdon, 1982. Team schedules for this project

[7] Drennan, "Motivating the Majority," Management Today, March 1988, pp. 87 - 92.

[8] Farquhar, J.A., "A Preliminary Inquiry into the Software Estimation Process," Tech. Report AD F12052, Defense Documentation Center, Alexandria, VA, August, 1970.

[9] Fathi, Eli T. and Armstrong, Cedric, V.W., Microprocessor Software Project Management, Ontario Centre for Microelectronics, Marcel Dekker, Inc., 1985.

[10] Gooch and McDowell, "Use Anxiety to Motivate," Personnel Journal, April 1988, pp. 50 - 55.

[11] Griggs and Manring, "Money Isn't The Best Tool For Motivating Technical Professionals," Personnel Administrator, June 1985, pp. 63 - 78.

[12] Hollocker, Charles P., "Finding the Cost of Software Quality," IEEE Transactions on Engineering Management, Vol. EM-33, No. 4, November, 1986.

[13] Jones, T.C., " Programming Productivity: Issues for the Eighties," IEEE Catalog No. EHO 186-7, IEEE, 1981.

[14] Kilduff and Baker, "Getting down to the brass tacks of Employee Motivation," Management Review, September 1984, pp. 56 - 61.

[15] Lin, Chi Y. and Levary, Reuven R., "Computer-Aided Software Development Process Design," IEEE Transactions on Software Engineering, Vol. 15, No. 9, September, 1989.

[16] Linder, "Computers, Corporate Culture and Change," Personnel Journal, September 1985, pp. 48 - 55.

[17] McKinnon, "Steady-State People: A Third Career Orientation," Research Management, January - February 1987, pp. 26 - 32.

[18] Myers, Ware, "Allow Plenty of Time for Large-Scale Software," IEEE Software journal, July 1989.

[19] Ofner, "Keeping Your High Achievers Motivated," Management Solutions, July 1987, pp. 35 - 39.

[20] Orpen, "Individual Needs, Organizational Rewards, and Job Satisfaction Among Professional Engineers," IEEE Transactions on Engineering Management, Vol. EM-32, No. 4, November 1985, pp. 177 - 180.

[21] Parr, F.N., "An Alternative to the Rayleigh Curve Model for Software Development Effort," IEEE Transactions on Software Engineering, Vol SE-6, No. 3, May 1980.

[22] Putnam, Lawrence H. A general Empirical Solution to the Macro Software Sizing and Estimating Problem, IEEE Transactions on Software Engineering, July 1978.

[23] Putnam, L.H., "Example of an Early Sizing, Cost and Schedule Estimate for an Application Software System," Proceedings of COMPSAC '78, New York; IEEE, 1978.

[24] Seashore, "Group Cohesiveness in the Industrial Workgroup," People and Productivity, Sutermeister, pp. 352 - 356.'

[25] Smith, Larry A., and Mandakovic, Tomislav, "Estimating: The Input into Good Project Planning," IEEE Transactions on Engineering Management, Vol. EM-32, No. 4, November, 1985.

[26] Stahl, "When Selecting and Training Managers, Remember Power Motivates," Research Management, July-August 1986, pp. 26 - 27.

[27] Thamhain, "Managing Engineers Effectively," IEEE Transactions on Engineering Management, Vol. EM-30, No. 4, November 1983, pp 231 - 237.

[28] Thamhain and Wilemon, "Building High Performing Engineering Project Teams," IEEE Transactions on engineering management, August 1987, pp. 130 - 137.

[29] Vick, C.R. and Ramamoorthy, C.V., Handbook of Software Engineering, Van Nostrand Reinhold Company, 1984.

[30] White, "Corporate Culture and Corporate Success," Management Decisions, April 1984, pp.. 14 - 19.

[31] Yeh, R., "Software Engineering," IEEE Spectrum, Vol 20, No 11, November, 1983.