

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9218221

**Information commodities and the production process: How the
introduction of CASE tools affects the software development life
cycle**

Bellin, David, Ph.D.

City University of New York, 1991

Copyright ©1991 by Bellin, David. All rights reserved.

U·M·I

**300 N. Zeeb Rd.
Ann Arbor, MI 48106**

A

**INFORMATION COMMODITIES AND THE PRODUCTION PROCESS:
HOW THE INTRODUCTION OF CASE TOOLS
AFFECTS THE SOFTWARE DEVELOPMENT LIFE CYCLE**

by
DAVID BELLIN

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1991

(c)1991

DAVID BELLIN

All Rights Reserved

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

1/18/91
Date

Abbe Mowshowitz
Dr. Abbe Mowshowitz
Chair of Examining Committee

1/25/91
Date

T. Wesselkemper
Dr. T. Wesselkemper
Executive Officer

Prof. Daniel McCracken

Dr. George Schneller

Supervisory Committee

The City University of New York

Abstract**INFORMATION COMMODITIES AND THE PRODUCTION PROCESS:
HOW THE INTRODUCTION OF CASE TOOLS
AFFECTS THE SOFTWARE DEVELOPMENT LIFE CYCLE**

by

David Bellin

Adviser: Professor Abbe Mowshowitz

The goal of this research is to model the software production process in such a way that the effect of the introduction of CASE tools can be analyzed. The production process will be represented by graphs, with cost relations connecting the production steps shown as nodes.

The model and the effect of the introduction of the CASE tool will be placed in a theoretical framework. Specifically, CASE tools are an example of the general category of information commodities, and the software production process is an example of manufacturing processes in general.

Such an analytical framework is intended to contribute to the understanding of the software production process, the effects of automated tools on software production, and the effects of the introduction of information commodities into production processes in general. There have been attempts to understand the software production process abstractly, and to discuss the value of information. However, little theoretical study has been done to create a framework for understanding change to the production environment, especially from the user's viewpoint.

This analytical framework will enable a potential user of a CASE tool to

estimate the value of the tool to their organization, independent of any claims to value made by the tool producer. A relationship between the use of the tool and organizational flexibility and hierarchy may be shown. And finally, such ordered knowledge may be useful in strategic planning and policy making.

Acknowledgments

A doctoral dissertation is an undertaking which often seems without end and of perhaps questionable significance. While commitment and perseverance have to come from within, outsiders who are willing and able to participate in an exchange of ideas are crucial to the process.

Such participation was freely given by Dr. Abbe Mowshowitz, who performed as advisor, pusher to clearer analysis, and colleague beyond any formal demands of his time. Frequent discussions in his home and office were intellectually stimulating and were of inestimable assistance. Dr. Dan McCracken and Dr. George Schneller, the other members of the committee, asked hard questions at every stage, which was their "job", and thus forged clarity about the contribution of this research. Others who assisted as sounding boards for my ideas included friend, colleague, and consultant Dr. Susan Suchman, friend and computer consultant Ed Levy, and friend and colleague Dr. Jeremy J. Shapiro.

Various other friends and colleagues heard bits and pieces of my formulations along the way, although I cannot hope to list or remember them all.

I hope all of those mentioned, by name or inference, will feel appreciated in knowing that a work of some small assistance to the development of better software has finally been completed.

Table of Contents

Abstract iv

Acknowledgments vi

Introduction 1

Chapter 1: The Commoditization of Information 2

Roots of Commodity Production 2

Definition of Commodities 2

Effects of Commodity Production 3

Effects of Information Commodity Production Today 4

Competing Views of Information Commodities 5

"Information Value": A Clarification 7

The Economic Value of Information Commodities 11

User Interest in Information Commodities 13

Chapter 2: The Software Production Process 22

Introduction 22

Defining Software Development 23

Review of Software Development Models 26

Waterfall Model 27

Prototyping 27

Evolutionary Development 29

Spiral and Black Box Models 30

Process Maturity Architecture 31

Summary 34

Software Cost Estimation 36

CASE Tools 39

CASE Tool Productivity 42

Chapter 3: A Model of Software Production Tasks and Their Costs 46

Model Objectives 46

Modelling 52

A Sample Production Cost Graph 56

Application of Graphs to Software Development 60

Methodology for Constructing a Software Production Cost Graph

66

Summary 74

Chapter 4: Applications of the Cost Graph Model 76

Interview Process 76

Company Profiles 80

Company A 80

Company B 81

Company C 83

Company D 84

Company E 85

Classes of Production Cost Relationships	85
Cost Graph Analysis of Typical SDLC Detailed	89
System Analysis	91
Rework	97
System Design	98
Program Analysis	100
Coding	101
System Integration	103
CASE Tools: Model Change Analysis	104
Graph Comparison	106
Sample Costs and the Transition to CASE	111
Conclusion	112
Chapter 5: Conclusion	114
Cost Graphs and Alternative Phase Costs	114
Fleckenstein	115
Interviewee Assumptions	116
Utility of Cost Graphs for Modelling Software Production	117
Research Directions Using Production Cost Graphs	120
Appendix: Figures	123
References	166
Bibliography	174

List of Figures

Figure 2-1: IEEE Waterfall SDLC Standard	124
Figure 2-2: Pressman's Protoype SDLC Model	125
Figure 2-3: Pressman's "Combined Prototype" SDLC Model	125
Figure 2-4: StarSys Inc. MacBubbles Suggested SDLC	126
Figure 2-5: Hewlett-Packard Metrics Oriented SDLC	127
Figure 2-6: Boehm's Spiral SDLC Model	128
Figure 2-7: Mills' Spiral SDLC Model	129
Figure 2-8: Box Diagram of SDLC and Business Relationship	129
Figure 2-9: Humphrey's Universal Level of the SDLC	130
Figure 2-10: Humphrey's Universal Level SDLC Cell Specification	130
Figure 2-11: Humphrey's Atomic Level (example)	131
Figure 2-12: Intermediate COCOMO Cost Driver Attributes	132
Figure 2-13: Analysts' Rankings of 17 CASE Functions	133
Figure 3-1: "Dura-Bar" Example: Preliminary Graph	134
Figure 3-2: "Dura-Bar" Example: Production Cost Graph	135
Figure 3-3: "Dura-Bar" Example: Annotated With Dollar Amounts	136
Figure 3-4: A Simple Cycle	137
Figure 3-5: Dummy Node to Remove Looping	138
Figure 3-6: A Cycle	139
Figure 3-7: Cycle Removed	140
Figure 3-8: SDLC Overview Based on Waterfall Standard	141
Figure 3-9: Participant Firm SDLC Standard	142
Figure 3-10: Draft Cost Graph of Program Analysis Sub-tasks	144
Figure 3-11: Draft Cost Graph of Program Analysis Eliminating Cycles	145
Figure 3-12: Cost Graph for Program Analysis	146
Figure 4-1: Interview Background Questions	147
Figure 4-2: Illustrative "Cost Graph" Used During Interviews	148
Figure 4-3: Company Data Summary	150
Figure 4-4: Company by Cost Graph Group	150
Figure 4-5: Cost Graph for Group 3 Structured Methodologies, CASE Use	151
Figure 4-6: Cost Graph for Group 2 Structured Methodologies, No CASE Use	152
Figure 4-7: Cost Graph for Group 1 No Structured Methodologies, No CASE Use	153
Figure 4-8: SDLC Overview	154
Figure 4-9: One Way to Graph Rework Without Cycles	155
Figure 4-10: Group 3 with Changes from Group 2 Highlighted	156
Figure 4-11: Spreadsheet Dollar Assumptions	157
Figure 4-12: Spreadsheet Printout of Group 2 Cost Graph	158
Figure 5-1: Fleckenstein Phase Cost Assumptions	160
Figure 5-2: Spreadsheet Printout Using Fleckenstein Assumptions	161
Figure 5-3: Interviewee Cost Assumptions	163
Figure 5-4: Graph Printout Using Interviewee Assumptions	164

Introduction

The objective of this thesis is to show that a new approach to modelling production costs can be used as an effective tool for managing the software development process. Called "cost graphs", this approach provides a graph-theoretic means of analyzing production cost relationships in any industry. The argument in pursuit of this objective is developed in five chapters as follows:

Chapter 1: The Commoditization of Information, sets the theoretical basis for this study by examining the economic and historical context of the rise of information commodities. It also provides a definition of such commodities.

Chapter 2: The Software Production Process provides a comprehensive literature review covering software development life cycle models. A brief review of relevant literature in software cost estimation and tool productivity is also provided, closing with a definition and classification of CASE (Computer Aided Software Engineering) information commodities.

Chapter 3: A Model of Software Production Tasks and Their Costs, provides the basis in graph theory for the cost graph model, as well as the decisions used in the representation of certain production tasks.

Chapter 4: Applications of the Cost Graph Model, applies the cost graph model to the set of companies interviewed in the course of creating the analytical framework of the thesis.

Chapter 5: Conclusion, examines the potential utility and possible limitations of the cost graph approach, not only for the software industry but for production settings in general.

Chapter 1: The Commoditization of Information

Roots of Commodity Production

Commodities, or goods produced for exchange instead of direct use, developed during the transition of feudal society into capitalism. An agriculture based economy developed into the industrial capitalism we are familiar with today. "As barter became more frequent, the producers began to produce goods regularly for the purpose of exchange. Production for use was superseded by production for exchange."[1]

With this transition, for the first time, whole production organizations came into existence whose sole purpose was the generation of exchange-value through the transformation of nature via industrial production.

Definition of Commodities

Exchange value is realized through sale of this production in the marketplace. The exchange medium of the marketplace is money, and the goods exchanged are called commodities. There are two important characteristics of a commodity:

1. A commodity must imply legal ownership, the right to freely buy and sell at will.
2. The price of a commodity is not necessarily correlated with its "use-value", and may vary with market conditions.

It is useful to note that the size of the equivalence class will have an effect upon the valuation of commodities. Economists typically discuss commodities such as "#2 Fuel Oil" or "wheat", which contain a large number of goods which are the same. Other equivalence classes might include "personal stereos", in

which the individual commodities are considered "essentially the same."

Equivalence classes for information commodities are generally smaller than those of more traditional commodities.

Effects of Commodity Production

The rise of commodity production caused a great transformation of the economy, and thus of the social fabric, reaching into every corner of human life. Even a person's very ability to work became commoditized as defined above: available for sale in the marketplace in ways it had never been before[2]. The impact of this transformation of labor has been profound. Skilled craft workers freed themselves from old ties to the guilds and masters, which were hampering the development of the productive processes, but at the same time they became involved in a new system of relationships over which they had less and less control. Through the very act of the sale of their labor power, the workers lost control of their social relations. In terms of production processes, the result was an increase of flexibility in the organization of production.

The purchase and sale, the commoditization, of human labor itself, made it perhaps the most important commodity in the rise of commodity production. Work was transformed from an individual undertaking into a mechanized process wherein the worker's individuality no longer mattered. Human labor was thus itself transformed, from something intimately connected with a particular human being, into a good which can be bought and sold in the marketplace. In this way, labor power was no different than any other commodity which came to be bought and sold through the exchange medium of money. The price of a worker's labor no longer related, however tenuously, to the direct physical needs

of the laborer and his family for survival, but instead came to be regulated by market forces. As this process developed, it became apparent that the buying and selling of labor power was no longer controlled by the laborers themselves, but instead was controlled by the marketplace.

The new, more complex, and increasingly flexible organization of production made possible by the commoditization of society has continued to this day, reaching a new peak with the recent transformation of information into a commodity.

Effects of Information Commodity Production Today

A process analogous to that which transformed labor into a commodity is taking place with respect to information in the age of mature capitalism. The means of production for creating information commodities are large mainframe computers, personal microcomputers, complex database software, and operating systems. The latter means of production are themselves information commodities.

Just as the commoditization of labor brought about the replacement of handicraft and feudal production with the labor commodity, now we see the replacement of labor power in production by information commodities. Draftsmen are replaced by computerized design systems. Robots replace welders in the assembly line. Automated numerical control machine tools replace skilled machinists in the shop[3]. A central result for the organization of production is a level of flexibility never seen before, as not only labor, but now knowledge itself is separated from the individual.

Further, just as labor did produce commodities before its own

transformation into a commodity, so too has information been the basis of other commodities. Newspapers and books, to cite only the two examples with greatest historical longevity, have existed as commodities whose composition was largely informational. In one sense, they have also exhibited the properties of a commodity as described earlier. However, they are themselves only partial examples of commoditized information in the sense we are speaking of. For newspapers and books cannot themselves transform the process of production of other commodities. And it is only in the most rudimentary sense that they can be repackaged, transmitted, transformed, and integrated into new process control products. The information commodity of today has potential to add value when part of a production process.

It has been pointed out that the:

profusion of information commodities is driven by recognition of information as a separable component of production and by the existence of the means to exploit this recognition. ...the market value of information behaves much like that of other commodities. Some information products and services are valued according to their age...others according to their innovative design...and all information commodities are subject to the laws of supply and demand.[4]

This view has been largely ignored in the literature, but it is the view that forms the basis for this study. In other words, I feel that the information commodity is the correct analytical concept, rather than the value of information in general.

Competing Views of Information Commodities

This view of the commoditization of information is by no means the accepted view in computer and information science. In fact, it is often argued that information is unique and unlike any other commodity [5][6][7]. Such a

view argues that information is, unlike other commodities, not consumed with its use. It can thus be re-used infinitely. As well, it is argued, the format in which the information is disseminated can be changed without changing the underlying informational content. Resting on communications theory as instantiated by Shannon, the entropy or information content of any message may be calculated apart from its commodity form [8]. The pervasion of such information throughout our working and social lives is thus said to be the causal agent of change, rather than any specifically economic transformation in the commodity sense.

Such a perspective can be useful in evaluating the flow of information throughout the macro-society, and perhaps also in measuring the volume and error rate of such information flows. However, it does little to help establish a clear picture of the value of information or information's impact upon production in the micro-economic sense. These views are therefore of little use in this research.

I do not argue that all information appears in commoditized form, even under advanced capitalist economic structures. We may again analogize to the commoditization of labor to see why this can be. Although labor power in general has become commoditized, there are still forms of labor outside of the exchange system. This unpaid labor includes for example that of a housewife. While it is socially necessary and important labor, the work of a housewife is not bought and sold in the marketplace. Allowances given to children are another such example.

In the information sector, examples of non-commodity forms of information would include a program I write for the use of myself and my

friends, a poem or letter written to a loved one, and songs recited in the shower. However, none of these aspects of information are of interest in this study.

"Information Value": A Clarification

There is a body of literature in information science and economics which addresses the value [9] of information generally [10]. While this literature is helpful in establishing a wide "macro" level picture, it is too general to be of help in the analytical framework pursued in this study. We intend to restrict the notion of value to the economic value of information as a commodity. This section briefly summarizes this literature, before the next section which details the value categorization to be used in this research. First information value as utility is reviewed, after which information value as a macro-economic measure is summarized. It is hoped that through such an exposition the reasons for the framework chosen will be clarified.

A prevailing view speaks of the "value of information" in the sense of a denotation solely expressing how "useful" it is to the recipient [11]. The value of data is not regarded as primarily economic [12], but is abstracted from economics completely. Informational value is relative to utility to a decision maker [13][14][15][16], interest to a user, or a potential future use thereby. Information value is frequently written of as "communicational utility". This communications approach is most recently expressed by the examination of data availability and flow within an organization to the members of that organization, as well as by an analysis of the impacts of that data upon the structure of the firm [17][18]. Included in this view as valuable information are the transmission

of messages up and down the managerial hierarchy embodied in the flow of communications within the organization [19][20][21].

Taylor of the Department of Information Studies at Syracuse University declares that

...there are no accepted or standardized ways of approaching the value of information. Indeed it is necessary at this stage to distinguish between two interpretations of value: the potential value of the information/knowledge content of the message, and the value of the service that provides the message.[22]

This distinction between the value of information content to the user, and the value of the provider service, is examined in a subsequent section. In terms of the overall value of information which we are now examining, Taylor later made explicit his view, which is already implicit in this quote. Information value is to be determined solely by the utility which the recipient feels it has [23]. In this way, economic value is only tangential to use value. While not explicitly stated in this form by Taylor, such an approach leads to the conception of economic value growing solely out of use value, with little regard to the exchange value of information in the marketplace. For example, if a library were to evaluate the value of installing terminals for online access to bibliographic databases, the primary factor to examine would be the utility of such a service to the patrons. If an examination of patron profiles reveals high value of such information, it is worth obtaining. Budgetary constraints are not a prime factor in such a valuation assessment. In fact, economic factors are often viewed as impediments to and constraints upon information value.

More sophisticated variants on this theme emerge, for example in Gotlieb [24]. He states that

Where there is a genuine market for information or information

services, it is not unreasonable to regard price as a measure of the value (utility) of the information.(page 114)

This would be close to the meaning of information commodity value, which we will examine shortly. However,

...a manager who receives a report on some activity in the company in which the interest is only peripheral is definitely *not* in a market situation. This point is emphasized by the observation that not only does the value of the information level off as the quantity delivered increase..., but it may well be that more is *less* valuable, ...(page 115)

The discussion continues with an examination of information economics as a subdiscipline of decision theory, complete with mathematical formulas which calculate the internal (use) value of information in the sense quoted. However even proponents of such theories agree that they are flawed and generally inaccurate, as they require a great deal of data regarding past performance of informational use which itself may not be similar to the situation being evaluated. Additionally, they require detailed data regarding the past performance and probabilities of the internal and external market. Finally such formulas make arbitrary definitions of the objectives to which the information use evaluated will be put, and there is no agreed upon systematic method for the creation of such objectives before the mathematical analyses are applied. Finally, by excluding the outside free market, such analyses do little to help with pricing decisions. They do not evaluate the relation of information commodities in the production process.

Indeed, information is available in many forms, not just computerized, and there have been many attempts to map its societal usage, flow, and value.

Machlup is justly credited with fathering this field of study with his book *The Production and Distribution of Knowledge in the United States* [25]. This was the

first detailed attempt to classify knowledge production in society, including education, R&D activity, and communications media. Machlup then attempted to calculate the portion of the national production figures classifiable as the total knowledge production of the United States. He further delved into the occupational structures thereby engendered, demonstrating the increasing shift away from blue-collar manufacturing work and towards white-collar office and service work.

Porat continued Machlup's pioneering work in a more meticulous fashion in his 1977 study for the US Department of Commerce [26]. Narrowing the study of information economics, Porat found that portion of the national economy involved with information to be large and rapidly growing. He estimated at the time that "46% of the Gross National Product is bound up with the information activity; earning 53% of labor income", this having risen from 30% of the work force in 1950. The trend is identical in all the industrialized countries [27]. A closer analysis of the Porat report shows that computerized information activities accounted for 25% of the GNP in 1977. This report has continued to attract attention, for not only is the portion of the GNP centered on information surprisingly large, but no quantitative study since Porat has attempted to duplicate such a detailed economic measure.

A more recent attempt to value information has arisen within the online database industry itself. Although the revenues attributable to this industry group are not recorded directly in government economic indicators, we are now able to estimate them. The industry itself reports to the Information Industries Association, which has indicated that revenues are growing at an annual rate of twenty-five percent, and that by 1990 revenues should be at \$40 billion as

compared with \$12.5 billion in 1983 [28].

Another approach attempts to quantify the value of information to society by measuring the flow of information through a "census of communications flows". The foremost proponent of this measure of information value is de Sola Pool [29]. Through the development of unique measurement indices for 17 communications media, de Sola Pool argued that there has been a very great rate of growth in electronic dissemination of information, even above that which was consumed! This he found to justify the socio-psychological concept of information overload. He also found a decrease in informational flows in print media. This however, does little to explain the contradiction between the high cost of this apparently overabundant supply of electronic information.

The Economic Value of Information Commodities

In the previous section we first discussed information value defined as uncertainty of content. However, it can be assumed that information has some use value. Otherwise it would not be sought after, accumulated, or purchased by otherwise sane individuals. Information content, by itself, without economic factors, is of little help in our analysis of the information commodity. This is due to the fact that exactly what we seek to measure is economic value, which assumes use value.

We then examined attempts to estimate of the economic value of information by measuring its "traffic", or flow through the general economy. Unfortunately, these estimators have operated at the macro level, and are thus of little help in the micro-economic sphere where day to day operational decision making takes place. Global calculation of the portion of the national

economy devoted to information related activities will not help explain the value of individual units exchanged for money. Thus, we must find some way to measure the micro economic valuation which takes place when an information commodity is bought and sold in the marketplace. Barter exchanges, which do not take place through the marketplace, are assumed to be negligible and are not relevant to the discussion. Many economists would argue that such barter is controlled by market forces in any case.

In defining the characteristics of commodities earlier, I argued that many (although not all) forms of electronic information are commodities. An information commodity is an economic commodity whose core is information. Information is the organization, storage, processing, and presentation of data, which is for us the primitive element. Information commodities are therefore available for purchase in the marketplace. Just a few examples from a long list of such commodities include:

- purchased software packages (WordPerfect, rBase, etc.)
- leased software programs (VSAM, IMS, etc.)
- bibliographic retrieval services (Dialog, D&B, etc.)
- scientific data retrieval services
- governmental data services
- data networking services (Telenet, Tymenet)
- computer network services (CompuServe, Bitnet, Arpanet)

Accepting that information commodities are bought and sold on the marketplace, which serves as the arbiter of the contradiction between use and exchange value, we are left with the question of the measurement of value. Price is an accepted measure of value in a stable marketplace.

Even so, the valuation process as reduced to price itself contains two aspects. Value has one nature in the eye of the producer, and another in the eye of the user. To the user, the valuation process will include factors such as

cost benefit analyses, payback periods, feature analysis[30], cost avoidance, and internal rates of return. Harder to quantify aspects to the valuation process might include factors such as customer satisfaction and organizational cultural ambience.

In contrast, for the producer, the determinants of price are quite different. Producer valuation factors include external return on investment, costs of production, market timing, competition, profitability, and related factors. Thus, the commodity valuation process is quite different depending upon whether information is priced by the producer or the consumer. The producer's interests are served by the creation of a commodity with as high a price as possible, while the consumer interests are clearly in maximizing use value at the lowest price, the converse. While this may result in temporary situations in which price is not congruent with value, these variations should disappear over time. For example, a producer who makes a pricing decision too far above the user calculation of value should eventually go out of business due to a lack of exchange activity! In other words, price is generally a good indicator of the value of economic commodities in general, and of information commodities in particular.

What remains to be seen is why the purchaser has any interest in the acquisition of the information commodity, at any price. How does the user approach the buying of an information commodity?

User Interest in Information Commodities

We have just established how to evaluate the economic value of information commodities, and we have decided in the preceding discussion that

the measure of economic value that we will use is the price of the commodity. That being the case, we must still address why any user would want to pay that price charged by a vendor for an information commodity they wish to introduce into a production process. The vendor's price, or selling price, translates for the user into costs. To determine how much to pay for an information commodity, the user must assess how production costs will be affected by the introduction of that information commodity.

What is the economic value of an information commodity to a user organization? There are several reasons why managers of production processes are interested in introducing such tools. These include [31][32]:

- increased productivity
- improved quality
- better managerial control
- production flexibility
- increased data on production costs

It is my view that most managements will invest in an information commodity for a combination of all these reasons, so each should not be viewed in isolation. However, it is useful to briefly analyze each reason for the acquisition of an information commodity in turn.

If productivity is increased, with all other factors being held equal, it is apparent that the cost of fabricating the end product will decline. If the production process can be analyzed so that one can apply the information commodity to labor intensive tasks it is often presumed that productivity will increase further. The additional advantage of an information commodity in a production process is that presumably the application of it to small sub-tasks will

enable the identification of production tasks which cross product lines. In other words, one should be able to apply the information commodity to similar sub-tasks in the production of different products of a similar type. To the user, this will decrease costs by spreading the acquisition and implementation investment across different product lines. In fact, as we shall see later, this is exactly the situation in many software development environments. In the organizations we interviewed for this study, the acquisition costs of CASE tools were not applied to any single project. Rather, the cost of tool acquisition was considered part of the overhead costs of the organizations as a whole, similar to costs incurred for hardware development platforms, office furniture, and the like, which we found generally not to be charged to specific projects.

Improved quality is another benefit of the introduction of information commodities, a benefit which has received much attention. Much of this attention is due to perceived higher quality of Japanese products.[33] In fact, improved quality may lead to greater production costs. However, this is often acceptable with the argument being that improved quality will lead to greater market share and competitive advantages. Remember however that in this study we do not primarily discuss environments which produce mass-market commodities, even information commodities such as software packages for the mass market. Our study primarily addresses software development organizations which produce products for internal use. Quality here is important also, since software being produced will impact many diverse areas of the organizations interviewed. Therefore, improved quality of the end product, especially if the end product is itself software, can be a powerful inducement to the introduction of an information commodity.

Production flexibility is another item of great interest to users in considering the acquisition of information commodities. If the acquisition can itself give the company greater ability to respond to market changes (or user requirements), by changing its production organization, the user has an increased ability to produce new or changed products on short notice. This itself is a great advantage in the current business economy. This is one of the advantages received from the introduction of computer controlled machine tools in many companies. This ability to rapidly shift production to new product lines, to change the production task relationships, and to increase managerial control of the production process has been well documented.[34][35][36][37][38][39]

One of the most touted advantages from the introduction of information commodities into production environments is the fact that in the very course of getting the benefits discussed in the preceding paragraphs, detailed quantitative information becomes available. Never before obtainable, this data details the production process itself. This is a major selling point for the introduction of such a tool. Since many managers feel that they operate "in the dark" much of the time, more data on their production operations is always of interest. Unfortunately, many managers do not realize that much of this data may be useless. They will often have to deal with a vast quantity of data produced as a result of automated production process monitoring. This is not the place to debate the nature of this data, its utility, its legality, or its social effects. These factors are, we should note, the subject of much debate. The effects of computerized monitoring of individual production workers has in particular received a great deal of scrutiny.[40]

On the other hand, it can reasonably be said that data which directly

controls the product as it moves through the production process has been found to be very useful. Such data ranges from software which tracks machined parts through the different machining processes, to code library and configuration management routines which maintain control over computer software. As well, increased data on production costs, in particular on material usage, and labor time, has been widely accepted in many production environments, especially in the traditional manufacturing sector.

Just as there are diverse reasons for the installation of hardware, it is also difficult to do a cost-benefit analysis for the introduction of information commodities. Much of the economics literature on this issue, especially in recent years, has focused on a quantitative "feature analysis". In such an approach, each portion of the commodity to be purchased is named and given a rating out of a total feature merit scale (usually on a base of 10 or 100). The product of each vendor is then rated against this scale, with the highest scoring vendor offering a candidate for acquisition. In the field of software economics, Boehm describes a detailed feature analysis system for installation of computer systems.[41] Included in his analysis are systems software, cpu, peripheral equipment, and various manufacturer (vendor) attributes. A discussion of similar systems is given in Gotlieb [42] and in Phister [43].

However, both Boehm and Gotlieb agree that such feature analysis systems are based on very subjective factors used in the construction of the feature lists and the weighting given to each feature on the list. Therefore such an approach can be used as a guide, but cannot be considered an objective quantitative cost-benefit analysis. Even the user interests we have just discussed for information commodities are not strictly quantifiable. For example, increased

productivity is often said to translate into decreased labor costs. However this seems often not to be the case. As pointed out by Gotlieb: "Often, total labor costs do not really go down - they are held constant even though the volume of work increases considerably. Thus the labor savings may take the form of cost avoidance." [44] Gotlieb does go on to show various mathematical formulas for user cost-benefit analysis, but he does point out that these calculations hold only if the benefits can be estimated with certainty, which is generally not the case. He goes on to say that "The main driving force for automation in an industry may simply be that it is 'in the air' and that any organization that fails to make the change senses itself to be, and may really be, at a serious competitive disadvantage." [45]

However, I believe there is still one more important, non-quantifiable reason for the introduction of information commodities. It is not only that they are "in the air". Rather it is that the use of information commodities results in increased managerial control over the production process. I would argue that this is true across all industry sectors, as has been documented by Shaiken in the machine industry, by Garson in various industries, and by Howard. However, I think it is especially true in the realm of software production. Whereas the cost to performance ratio has improved hundreds of times over for computer hardware in the last few decades, programmer productivity has risen only minimally, and this appears to remain true in the era of fourth generation languages. Therefore, in software development, it is becoming apparent that labor costs are increasing as a percentage of the total costs of production. This has been well documented in the data processing field, and the impact in terms of management control has been analyzed particularly incisively by

Greenbaum.[46] Greenbaum shows the growing routinization and specification of the work of systems analysts and programmers. She points out that such routinization does not always occur with strictly economic justification. Rather, for data workers, as for other workers, introduction of information commodities occurs so that management will have greater control over not only the production process in general, but over the people who engage in that process. In fact, Greenbaum argues that software production automation will primarily affect programmers, but that this doesn't really matter, since the earlier stages of systems analysis affect the organization of production more than the introduction of technology itself.[47]

In any case, given more data, given increased productivity, and given increased flexibility in the organization of the production process, management does *de facto* obtain better control, stricter control, over that process. And the allure of such increased control, real or imaginary, is a strong incentive to users toward the acquisition of information commodities. If we review a few information commodities which have been adopted by users, we will see that the introduction of information commodities in the software production process differs little from that which occurs in other fields. For example we stated earlier in this chapter that computer controlled automated machine tools represent information commodities introduced into the machining process. Architects have introduced computer aided design packages into the building design process.

In the computer field, information commodities are being applied to the production of software itself. The first examples of information commodities applied to the software production process were of course commercially available

compilers, language translators, and database management systems. These did have an effect upon small portions of the production process, namely the programming part of the process. Recently a new class of information commodities called Computer Aided Software Engineering (CASE) tools are being sold which hold the promise of automating a wider portion of the software production process. In the next chapter, we will review common models of the software production process, and at that point we will define CASE tools. However, at this point we merely seek to justify our contention that CASE tools are in fact an information commodity. Referencing our earlier discussion of commodities, we said that they exhibit two important characteristics. First, a commodity must imply legal ownership. Second, the price of a commodity may vary with market conditions, but it must be bought and sold. I think it is clear that at the present time, within the legal and economic structure of the United States, that one can develop a program which will automate part of the software production process, buy and sell it at will in the marketplace, and its price will in fact vary with market conditions. Price variance is often particularly erratic when new commodities are introduced into the market, as can be observed with CASE tools. Wild price fluctuations are common, since pricing decisions are made not only on the basis of investment costs, but also on an evaluation by the producer of the cost benefit to the user (use value). As well the producer must take into account the market's competing commodities. Presently many CASE products exist varying in price from \$100 to well into the six figures. It is clear that this class of software embodies the replacement of labor power, and knowledge power, replacing human activities with those of a computer program. In particular, CASE tools obviously embody the aspect of information

commodities in that they are able to transform the process of production of other commodities. The information commodity, as stated earlier, has the potential to add value when part of a production process, and this is the situation we find with CASE tools.

In summary, we have reviewed the roots of commodity production, defined commodities and information commodities, disputed commonly held views of information value and the economic value of information commodities, and tried to establish the motivations of users for the introduction of information commodities into production processes. In this research, we will examine the impact of CASE tools, a particular information commodity, on the software production process, which is detailed in the next chapter.

Chapter 2: The Software Production Process

This chapter reviews the literature covering software production process models (the "software development life cycle"), as well as that in the related areas of software cost estimation and tool productivity. We establish a definition of CASE tools and categorize existing tools.

Introduction

In order to understand the interest of a user in introducing an information commodity into her production process, we must be sure we understand the general scope and nature of that production process. In this research, we examine the production process of software fabrication, the end product being a set of modules which will actually be implemented on a computer system in a business environment. Our focus is on large products created by multiple teams of programmers (a subsequent chapter details the environments of the participating companies). Software production in other environments, such as scientific, military, or home environments, may exhibit different production characteristics.

In order to understand the specifics of the software production process, we will first define the nature and scope of software engineering and software development. We will then review current descriptive models of the software development process, generally known in the literature as Software Development Life Cycle (SDLC) models. These range from the traditional "waterfall" model, through prototyping and Harlan's boxes, to the recently introduced process architecture model. The information commodity which we will be analyzing is defined. This class of information commodities is known as CASE (Computer

Aided Software Engineering) tools. We then review literature in the related areas of software cost estimation and software tool productivity.

Defining Software Development

Information systems exist to support three major areas of business functions: transaction processing, managerial reporting, and decision support.[48] Over the past two decades organizations of all kinds have invested growing amounts of money in the development of information systems to meet their business needs. As the years have gone by, the systems which have been produced have become larger, more complex, and had a more significant dollar impact on the bottom line of the companies introducing the systems. Another trend which has been noted is the increasing proportion of information systems development devoted to the software component, as opposed to the hardware, of such systems. Software costs are now the major factor in most information systems, and because of this a discipline called software engineering has grown up to embody the set of knowledge necessary for modern software production practices. There is general consensus that software engineering focuses on the software development process in a macro sense. That is to say, software engineering encompasses an approach to systems development which is interdisciplinary, focussed on more than the technical skills necessary to produce software. While technical people are needed during the software production process, it is now recognized that today's complex systems are increasingly driven by market requirements, customer demands, and industry sector trends[49]. In this atmosphere software engineering, encompassing software development, has been defined with different degrees of nuance:

Software engineering can be characterized in a number of different ways. When The chronology of the software development process is examined, software engineering flows through three generic phases: *definition*, *development* and *maintenance*. The phases are often modeled using one or more *paradigms* that describe the individual steps associated with each phase. When the pragmatic elements of software engineering are considered, the discipline may be viewed as a set of procedures, methods and tools. Finally, when the end-product (working, maintainable software) is the focus, software engineering may be viewed as a set of *deliverables* that encompass documents, programs and data. In actuality, each of these views overlaps the others.[50]

In addition to being such a set of tools and deliverables, software engineering also encompasses management activities:

The system development process is a management paradigm for defining and scheduling work in investigation, specification, and implementation activities.[51]

This means that more than technical matters are involved with software development. Management is concerned with containing the rapidly growing cost of information systems project development. In short, software engineering may be best described as "The disciplined application of engineering, scientific, and mathematical principles, methods, and tools to the economical production of quality software." [52]

Let's place the software development activity in the context of the economic activity of the United States in general. It has been estimated that in 1975 there were 220,000 programmers in the country, and that the number is now close to one-half million. Fourteen years ago \$ 41 billion were expended annually for data processing. Now the figure is well over \$ 200 billion, accounting for approximately 10 percent of the gross national product.[53] The market for tools to automate the development process is now \$ 1.4 billion (1989) and is forecasted to grow to \$ 4 billion by 1993.[54] This is but one way of looking at the importance of software development to the economy. In the

previous chapter we discussed the importance of information generally to the economy. It would be no exaggeration to say that the daily functioning of our society would be drastically altered without computer and communications systems. All this means that controlling the costs of software production is an increasingly important task, not only for any particular organization seeking to save money, but also for the economy as a whole.

There are many means through which one may seek to decrease costs and increase productivity. One can focus on creating an environment which fosters effective work habits and constructive interpersonal relationships. One can focus on ergonomics, the physical arrangement of the workplace: better lighting, well planned work stations, and the like. One can focus on data processing project management techniques: how to plan, the role of Critical Path methods. One can discuss the provision of resources to the personnel: are there enough terminals, what is the turnaround time on compiles. One could also focus on training: how to provide the necessary knowledge and skills for the performance of a particular production task. And one can focus on more subjective attributes, such as how to provide personal satisfaction to programmers and analysts. The rewards which software practitioners seek may be different than rewards sought by other workers. One could also focus on communications techniques, both formal and less formal, which will increase both productivity and efficiency and decrease the cost of software development.

We have chosen to focus on the identification of the particular sub-tasks which make up the development process. It is our approach to define work requirements and develop an analytical approach to the evaluation of the costs of production at each sub-task, along with the costs of the input necessary to

production at each sub-task. Such a view will assist in the containing of software development costs. Therefore, a survey of the major models of the production process is in order.

Review of Software Development Models

Humphrey has pointed out that process models can be used either as a description of actual practices, or as a suggestion for what those practices should be.[55] The models we are concerned with in this work are those which are primarily descriptive models of what actually takes place during the development of new software products. Our cost graph analytical framework is itself primarily a descriptive model, since the construction of the graph is based upon the actual tasks carried out during software production and the cost relationships between these tasks. This will be shown in detail in the next chapter, where we will examine the theoretical basis of cost graphs, as well as a methodology for their construction. The cost graph analytical technique should not be confused with a prescriptive model which would attempt to suggest the way in which the production process *should* be organized. We make no such evaluation in this work. To some extent however, it must be admitted that any model is by nature a combination of description and prescription. This is because a model adopted in practice by an organization is introduced as a standard, which is meant to both describe the way production is organized, as well as to be a standard from which deviations can be measured. There are several accepted software development models in use today by commercial software development organizations. The usefulness of any model, in particular what are called "life cycle" models, has been questioned.[56] Such criticisms are generally based upon

the use of such models as prescriptive devices. In other words, critics of life cycle models regard them as impediments to a better organization of software production. This may in fact be the case, however, such a discussion is tangential to the purposes of this paper, in which we seek to examine software production through a more descriptive approach based upon actual production practices.

Waterfall Model The most common software development models in current use are variants on what has been called the "waterfall" life cycle model. In fact, the Institute of Electrical and Electronics Engineers has developed a standard for this model, shown in Figure 2-1. It is a sufficient *description* of the software development process, in that it does describe at a general level the basic steps which are engaged in during software production. This model does give some guidance as to the relative order of performance of the major tasks. This model has been justly criticized however, for the implication of a strictly sequential view of production tasks. The model can lead one to ignore the complexity of, and the inter-relationships between, the sub-tasks of software production.

Prototyping If improperly applied, it is possible that users of the waterfall model will ignore early prototyping and early development techniques. Therefore, one variant of the waterfall model was based upon an approach which emphasizes the benefits of early prototyping. Such a model is discussed by Pressman, who introduces a prototyping model for software engineering [57]. Shown in Figure 2-2, prototyping moves the developer towards an early implementation of partial products. Pressman argues that object-oriented programming is particularly amenable to prototyping. And this approach gives

users the benefit of getting an early working model which can be used as the basis for a revised needs assessment. In this way, Pressman argues that the actual developed software product will be of higher quality. However he also sees that prototyping may be problematic. Users tend to believe that prototype screens mean that a complete system can be completed very easily and quickly. Second, when using prototypes, system architects have a tendency to develop poor quality products, using short cuts in order to complete production as quickly as possible. Lastly, prototyping is most applicable to software which includes a high degree of user interfacing. Such an approach is of limited value for applications of embedded systems, transaction processing, or batch processes.

In his book, Pressman argues that there are other "fourth generation" and "combined paradigm" models which can obviate the problems of both the waterfall and the prototyping models. However, he does not explain them in detail, and it is left to the reader to do so and test his argument. Pressman's "combined paradigm" model is shown in Figure 2-3.[58]

More sophisticated variants on the waterfall model exist, and have been widely adopted in commercial settings. The largest selling CASE tool for the Apple MacIntosh[59] provides a suggested SDLC model to users which is based on a prototype oriented waterfall model (see Figure 2-4). This SDLC is typical of those widely accepted by industry, although the figure shown is more graphically oriented than many such models. Many approaches have attempted to annotate the prototype and waterfall models, with the specification of points at which data should be collected for further process model study within an organization. An example in Figure 2-5 is that provided by researchers at Hewlett Packard [60].

Evolutionary Development As an alternative to these approaches, Gilb has introduced the "evolutionary development" model as a prescriptive approach to software engineering. "The evolutionary delivery method...is based on the following simple principle: deliver something to the end user. Measure the value to the end user in all critical dimensions. Adjust both design and objectives based on observed realities."[61] The difference from earlier approaches stressed by Gilb is the implementation of many early delivery dates, instead of planning based upon a single delivery date which is often in the "distant future". Instead of the long life cycle, sequential waterfall model, Gilb advocates building partial products early and delivering them when they are built. In other words, he argues for a life cycle in miniature, which is repeated many tens, dozens, or (perhaps?) hundreds of times before production is complete. He calls this planning for multiple objectives, permitting your objectives to change and be revised into new ones as software development progresses. The key to this is early and frequent iteration of the analysis, design, and delivery cycle at each step along the way. This approach is attractive and well argued, and it is very user oriented. Evolutionary design is a business information systems approach, rather than a more technical orientation towards algorithm design or a particular implementation technique. To his credit, Gilb realizes that even his proposed evolutionary model has drawbacks.[62] The disadvantages of evolutionary development include extensive training needs, a long maturity process, need for extensive management support, and the need for progress measurement techniques. However, he sees these drawbacks as minor. Gilb's approach can be summarized as an excellent prescription about what needs to be done by software engineering management. Unfortunately, it is not a good description of

practices of most organizations, nor is it used by any of the companies we interviewed for this paper.

Spiral and Black Box Models It is the spiral model of software development, introduced by Boehm, which has attracted much attention in the computer science community [63]. Growing out of Boehm's work for the TRW Defense Systems Group, the spiral model combines risk analysis with the "specification driven" waterfall model, and with other "prototype driven" production models. Essentially, in this author's view, the spiral model itself could include any of the production models already reviewed within it. The spiral model provides a framework for deciding which prescriptive model should be enforced in a given software production environment. A graphic portrayal of Boehm's spiral model is shown in Figure 2-6.

Mills has used his black box technique to model the systems development process. He recognizes that while software production proceeds through all the tasks shown in the waterfall model, it does not engage in these tasks in a simple sequential manner. Mills categorizes the activities of software production into three general classes: investigation, specification, and implementation. An investigation is a fact finding study before specification, which focusses in on a specific information system. Implementation converts the specification into an operational system. These last two classes of specification and implementation seem to be what are commonly referred to as systems analysis, systems design, and programming. Mills uses his box structure to provide a modelling form for the system development process. He alleges that each activity goes through the three stages of planning, performance, and then evaluation. While not to be confused with Boehm's approach, Mills also organizes the development process

into a spiral, shown in Figure 2-7.[64] The relationship he sees between system development and the general activities of a business, is shown using his black box technique in Figure 2-8.[65] The Mills approach can be seen as both descriptive and prescriptive, since there are some companies who do follow his approach, although none were among the five companies interviewed during this study.

Process Maturity Architecture Perhaps the most important and well described new model of the software development process has been that provided by Humphrey in his recent book *Managing the Software Process* [66]. Humphrey makes what appears to be a very sensible division of the software development process into a three level architectural model. The topmost level is the Universal level, which describes the policies meant to be enforced during the production of software. This is a very abstract level, meant to guide the entire production process, and is somewhat similar to that seen in Figure 2-1 for the waterfall process model. His next level down from the universal level is the Worldly level, which spells out the work procedures which embody the policies of the universal level. These procedures define who does what tasks, and when they are supposed to do them. The worldly level of the development architecture defines the results of each subtask, appropriate measurement techniques for quantifying the work done, and selected managerial control checkpoints. The most itemized level of this model is the Atomic level, which can be very detailed. In particular when tasks are to be automated, Humphrey states that a particularly high level of detail is required. The detailed definitions of atomic processes will generally be embodied in process standards written at the atomic or worldly level. The worldly level is where methodologies are

defined, along with the means by which their implementation can be assured. And it is the worldly level of Humphrey's model which provides the most insight into the software production process for the descriptive model which we develop later in this study. An example of Humphrey's universal level is shown in Figure 2-9 [67], with its accompanying cell specification shown in Figure 2-10.[68] A possible drawback of Humphrey's approach is that the atomic level process models can be overly complicated. As an example, he discusses a model of regression testing in his book. Figure 2-11 shows an incomplete atomic level diagram for regression test refinement, only a small part of regression testing in total.[69] However, even this is not the most detailed level of the model. To fully construct a model of his architecture, one would have to further decompose the cells shown in Figure 2-11. Even Humphrey does not attempt this in his book, which may be understandable given that his goal is the prescription of a better way to manage production. One of the refreshing aspects of Humphrey's approach to development architectures is his recognition of the reality that no single model is "best". The model chosen must depend on the needs of the organization performing the modelling. Humphrey advocates the use of early prototyping, although at the same time stating that there is no one correct way of doing prototyping. He also advocates the inclusion of entity process models into the development process architecture.

Humphrey's emphasis is on a state or process model view of the software development environment. However he also recognizes that substantial model differences will be produced if one tries to establish a production model when taking an organizational point of view, or a managerial control viewpoint. For example, from an organizational point of view, an organization chart indicating

the hierarchy of responsibilities for each employee position might be the proper modelling format, even though such a view of the development process will not help a programmer carry out his or her tasks very well. A managerial control view is particularly relevant when process control is of concern. Here one will have to decide when quantitative control data should be collected, how it can be collected, how it can be reported, and in particular where control checkpoints for managerial decision making are indicated. All these aspects are, according to Humphrey, important parts of a development model, and should be included when a complete software development architecture is put into place.

Humphrey himself emphasizes the software process from an entity and task point of view, which he calls a "state view". Humphrey feels it is very important that a process architecture be put in place, that process tasks are defined in a consistent manner, and also that the process architecture itself be amenable to change as the development process itself changes. However, he does state that all deviations from the process model must be documented, reviewed, and formally approved by management.

It was Humphrey who was quoted earlier in this chapter, raising the important concern that we must be careful to choose the most appropriate model for our purposes, rather than being concerned with "correct" modelling techniques abstractly. As indicated earlier, it is Humphrey who also raises the issue of prescriptive versus descriptive uses of models. While Humphrey's model does have a descriptive component, his whole approach is essentially a prescriptive one, and he admits this in his book. His intention is the definition of how software production processes *should* be conducted, suggesting where standards and policies can help guide the work. This is the way in which

Humphrey feels the major concerns of software production will be most effectively addressed. He lists these concerns as quality, the inherent instability of requirements, and the growing complexity of modern software products. To this end, he advocates a software process assessment procedure for organizations, which has three purposes. The first purpose is to learn what is actually going on during software production in a particular company. The second purpose is the identification of the major problems in production, and the third is the creation of change in the production process.

It is the area of managing production process change which particularly concerns Humphrey. In his view, organizations which engage in software production can be divided into five levels of what he calls "process maturity". These range from the most chaotic and poorly managed organizations to what he calls the "optimizing" maturity level, in which a company has a solid foundation for improving an already sound and well managed production process. In his own words "Transitions from level one to level two, or from level two to level three, take from one to three years with a dedicated management commitment to process improvement. To date, I have observed only few software teams at level four or five, and no complete organizations." [70] The argument in favor of using such a software development process maturity architecture as a prescriptive model, is that similar approaches have worked for many years in scientific and engineering environments. Given the problems in modelling software production, it is the best alternative that exists. Such is Humphrey's contention, and in general I am inclined to agree with him.

Summary In surveying the major software production models in the previous sections, it is perhaps surprising that while they have contrasting approaches to

the description of, and the ordering of, the various sub-tasks of software production, they all seem to agree upon the general scope of activities upon which that production is based. It is unlikely that the adherents of any of the models mentioned would not include the following activity paths as part of the software production process: feasibility analysis, requirements definitions, graphic modelling, information and either process or entity modelling, both logical and physical modelling of the desired system, some form of program analysis and modelling, some actual coding along the way, and testing and systems integration as well as (once the system is in operation) software maintenance. There is no disagreement that these are the basic tasks which form the software production process.

Disagreements arise in the discussion of the relationships among the production sub-tasks. Which tasks can produce partial products to be sent on to other sub-tasks, as contrasted to production tasks which must be completed in their entirety before output is sent on, if in fact there are any such sub-tasks. There are obvious disagreements regarding sub-task ordering: which tasks depend on which others, and in what order must they be performed? This debate has in fact extended to questioning whether any prescriptive model at all serves a useful purpose in advancing the state of software production processes. I feel however, that these debates have little bearing when one is trying to construct a *descriptive* model of *actual* production processes.

As will be demonstrated in the next chapter, this research formulates a description of the software production process which has as its goal a definition of the sub-tasks engaged in during software development, and an analysis of the cost relations among these tasks. We call this the "cost graph" model approach.

It is important to note at this stage that the cost graph model is not intended to be a prescriptive model which states how the production process *should* occur, nor does it try to indicate the most rational organization of that production. Rather, it attempts to model the actual production tasks which take place in a given organization (or set of organizations) and provides a way to analyze sub-task cost relationships within the production process. The goal of such an analysis in our case is to gain insight into the application of a class of information commodities (CASE tools) in this specific production process. Therefore, the cost graph technique is not dependent upon any particular software development life cycle model. In fact organizations which have chosen to implement any of the previously discussed SDLC models would have no problem applying cost graph analysis to their production process, as we will see in the next chapter. However, since the cost graph model, as we can tell by its name, does address costs, it is important to review the related literature of software cost estimation and software tool productivity before we can go on to define the class of information commodities known as Computer Aided Software Engineering (CASE) tools, and show how to apply cost graph techniques.

Software Cost Estimation

While the purpose of cost graph analysis is to model costs, not necessarily to predict them, many models do exist for the estimation of software project costs. The most widely used such model is Boehm's COCOMO (CONstructive COst MOdel) [71], which will be summarized momentarily. COCOMO is the basis of several commercial products, is widely used by US military organizations, and has been adopted by numerous businesses. Other important cost and

models include those of McCabe[72], Halstead[73], and Putnam[74]. These are summarized in Boehm, who includes some of their work in his, and will not be separately reviewed here.

The purpose of software cost estimation can be seen as reducing risk and increasing control. Accurate prediction of schedules, resource requirements, productivity, are all presumed to reduce the risk of software which is of low quality, late, or over budget. Increased control over production is desirable for it too will result in lower costs and a better quality product, with a concomitant ability to reduce production risk and gather data which will lead to more accurate future cost prediction. Boehm attempts to address all these factors in his COCOMO model, which includes over 50 parameters which affect software production costs.

Many of the COCOMO attributes are attributes of the software production environment, such as programmer language experience, and the use of modern software tools and practices. Others reflect product attributes, such as required product reliability, database size, and timing constraints. The intermediate COCOMO attributes are shown in Figure 2-12[75].

COCOMO provides varying levels of detail for cost estimation, each of which is claimed to provide successively greater accuracy. For example, basic COCOMO claims an error margin of a factor of 2 (100%) in 60% of the projects which use it[76] (making it almost meaningless in the view of some). Basic COCOMO assumes "average" device constraints, using product size as KDSI (thousands of delivered source instructions) as the major basis for calculation. Months of production effort is calculated as:

$$E = a * KDSI^b$$

where a and b vary according to the general nature of the software project:

organic (small product with experienced personnel)

$$a=2.4 \quad b=1.05$$

semidetached (medium size product with average experience)

$$a=3.0 \quad b=1.12$$

embedded (high degree of product constraint)

$$a=3.6 \quad b=1.2$$

The intermediate COCOMO model achieves greater predictive accuracy by requiring a more detailed knowledge of production constraint factors (see Figure 2-12). In this way it claims to achieve estimates which are within 20% of actuals 68% of the time[77]. Detailed COCOMO makes the cost drivers even more detailed by forcing the quantification of many items by phase as well as for the whole project. However this detail only produces a nominal improvement of 2% such that estimates are within 20% of actuals only 70% of the time[78].

All the COCOMO data is constructed upon a database of 63 software projects which cannot be presumed to be representative of any particular software project[79]. This, as Boehm himself points out, is a valid criticism of the model. The model also makes implicit assumptions about resource distribution over the life cycle phases which may not be in accord with practices in the particular workplace using the model. It is also not clear that cost estimation itself will result in greater project control and the ability to quantify cost data in the future. In fact, it has been estimated that software process data collection itself adds from 5 to 15% overhead costs to a project[80]. Perhaps the most telling criticism of software cost estimation schemes such as COCOMO

are that they try to do too much. This is meant in the sense that they utilize a large number of factors based upon generalized experiences, which any given organization may have no data to support. Furthermore, many of the factors are intertwined and a change in one affects many others in subtle ways. And factors such as usability and design feature analysis are explicitly excluded from many such models, while such attributes certainly have a cost effect. They represent cost estimation as a static technique to be applied before a project begins, when it may be more useful to view cost estimation as a dynamic effort which continues during production[81].

From the standpoint of the goals of this research effort, common cost estimators such as COCOMO are problematic in that estimation models suffer from the extremes of overly detailed models which are difficult to quantify accurately, or which are too superficial to connect to specific production tasks. They may help as cost predictors, but do not clearly portray the cost connections between the actual tasks engaged in to produce software. They do not show, for example, how the outputs from a production task are valued, and how the value of such output is split between two other tasks which need them as inputs.

Since the application of automated tools to such production tasks might be expected not only to change the organization of production, but also to affect costs, we now turn to a definition and classification of such tools.

CASE Tools

CASE tools are a category of software products which assist in systems analysis, system design, prototyping, programming, documentation and testing. These are software tools which, in other words, assist in the production of other

software products. The acronym CASE stands for "Computer Aided Software Engineering". While some replace the word "Software" with "Systems",

The intent of CASE is to build "software", not "systems". They are particularly well suited for organizing programming specifications and producing the software. CASE tools include diagramming aids, screen prototyping aids, program generators,...and test/debugging aids. ...they do not alleviate the need for the important up-front systems work.[82]

Perhaps twenty years ago, the tools encompassed by this term might have been operating systems and language compilers. Today such software tools are taken for granted, while CASE is spoken of at a "higher level", which assumes traditional single-programmer coding tools. An underlying software development base of compilers, sophisticated operating systems, and even database management systems is assumed to be used in the software production process. CASE tools intend to offer automation which will even further enhance the production of correct, well documented, and maintainable software.

Such a spectrum of tools must include:

everything involved in software development - not just diagramming, or data dictionary, or code generation, but everything from strategic planning, requirements analysis and project management all the way through to re-engineering and maintenance. A real CASE toolkit...will integrate very phase and every person involved in the software development process. It makes no sense to us to speak of CASE and Integrated CASE as though they were two separate things.[83]

However this remains generally true if one speaks only of the CASE spectrum, for few (if any) such tools claim to take care of "everything involved in software development." Many feel that such tools do not exist, as evidenced by the views of a member of Fidelity Software Development Company who reported to Datamation that "we don't see one [CASE tool] that spans the development life cycle...".[84]

Most CASE tools take aim at fairly specific classes of tasks involved in

software production. Some authors have taken to dividing the tools into three categories, upper-middle-lower, each meant to indicate the spectrum of tasks the tool helps to automate.[85] Upper CASE is said to be those tools which are directed at management, both technical and general corporate. This software is said to aid in the establishment of overall company goals, objectives, resources, and strategic plans. This is where you might attempt to construct an information model of the entire enterprise. Tools such as the PRIDE system from M. Bryce and Associates, or Texas Instruments' Information Engineering, claim to assist in these tasks.

The "middle CASE" tools assist working systems analysts and designers in automating the structured methodologies they use to plan and develop software products. This includes diagramming of data flows, entity relationship diagramming, data dictionary entry, diagram checking and balancing, and the like. They assist in the sharing of these resources among groups of analysts working on the same project, as well as sometimes assisting in the creation of shareable databases of these sub-task outputs which can be re-used between projects. There are a growing number of tools available in this category, the predominant PC based product appearing to be Index Technology's Excelerator with 52% of the market sites [86]. Other examples include DesignAid and CASE-2000 from Nastec, Stradis from McDonnell Douglas, MacBubbles from StarSys, and at least a dozen more. These tools would be of particular importance for this study in that they impact directly on the first three phases of the software development life cycle.

The focus of "lower CASE" tools is the generation of program code and software testing. This category includes code generation tools which generate

COBOL, Pascal or C source, such as MicroStep from Syscorp International. Version change control packages which keep track of versions as programs are tested and modified, such as the PCVS package from Polytron are included. Software which automates the creation of test data, and includes facilities to run and report on executions against the data set, such as the "T" system from Programming Environments, runs on a range of environments directed at larger shops. Creation of user documentation is sometimes included in this category, although there is debate as to whether this facility is properly part of the software development life cycle or not.

Another popular framework for categorizing CASE tools is into "front-end" products which support the earlier phases of the SDLC such as analysis and design, and "back-end" products which support code generation and testing [87]. The latter is equivalent to the "lower CASE" area described previously, while the former encompasses "middle" and "upper" CASE tools. Whichever terminology is used, there is general agreement upon the scope of CASE as described. However, many believe that "the potential return on investment in CASE technology remains unfulfilled." [88] This is partly the case because of a continuing reluctance of corporations to invest in CASE tools, as well as reports that "only 1 in 4 of those that have implemented CASE reported effective use of the technology". [89] Therefore the actual effect of these information commodities on the software production process is open to question. There seems to be no adequate framework for such an evaluation, a gap which this work may help to narrow.

CASE Tool Productivity How productive are CASE tools? Despite widespread vendor claims of lowered production costs and increased productivity, no

quantitative studies exist to support such claims, at least as they apply to the early phases of the SDLC. Some studies do exist examining programmer productivity tools.[90][91][92][93] What reportage exists on the analysis and design tools is largely anecdotal, tending to focus on the reasons for the introduction of CASE tools rather than on the results of their actual use.[94] Quantifiable justification can only be found in the case of studies on the subjective evaluations of analyst's regarding tool usage.

The most notable work on analyst perception of tool productivity has been done by Norman [95] and by Norman and Nunamaker [96], as well as reportage by a few others.[97][98] Norman and Nunamaker concur with this writer that "we are not aware of any empirical studies in the research literature that investigate CASE technology or its effects on productivity". Instead of attempting to do same, they decided to discover "which functional and behavioral aspects of CASE technology, *from the software engineer's point of view*, contribute the most favorably toward increasing their productivity over comparable manual methods." [99][emphasis added]

Both studies use multi-dimensional scaling for the clustering of stimulus items, which analysts were asked to rate in terms of tool contribution to productivity. The pool included 91 subjects who completed the PC-based questionnaire, all of whom were users of Index Technology's Excelsator CASE tool, and two-thirds of whom were COBOL or fourth-generation language users. No attempt was made by the authors to verify the claimed productivity gains reported by analysts. In light of these facts, the studies may be seen as initial approaches to an analysis of tool productivity. It suggests areas for possible quantitative measurement of productivity gains, and the need for a comparison

between perceived productivity and actual productivity.

The results of the study are a ranking of 17 CASE functions by productivity ranking, as shown in Figure 2-13. Two tool functions are at the top of the list, data flow diagramming and the data dictionary. They were selected 79% and 77% of the time respectively, resulting in a "relative choice" rating of 1.00 and 0.98. This means that software engineers select these two tools as the biggest productivity boosters. The next level down is a group of four functions selected from 59% to 57% of the time. Note that some functions are actual tool functions, while others are effects of using the tool (such as "project standardization" or "project member's communications").

The studies cited go on to cluster CASE functions on the basis of simple versus complex modeling, and least versus most productivity, in order to come up with groups of functions which form a "productivity space" for software engineers, not discussed here. Note also that the authors do not look at any functionality not included in the Excelerator product, for example code generation, version control, or test generation. The costs of implementing and learning the CASE tool are ignored.

The authors ably demonstrate that software engineers are able to agree upon a subset of CASE functionality which they feel enhances productivity. The reader should bear these studies in mind while examining the following chapters, in order to establish any possible correlation between our interview data and the Norman and Nunamaker studies. This point will be addressed subsequently.

The Datamation-Price Waterhouse CASE survey of 1989 [100] addresses the perceived productivity improvements of data processing managers. Much of the survey considers plans for CASE use, and expectations of CASE productivity

benefits. It is interesting that over 70% of companies with fewer than 500 employees did not expect to introduce "upper CASE" tools, which may indicate that the cost/benefit ratio is prohibitive for smaller firms. However, over half the surveyed companies of all types either now use integrated CASE tools or plan to do so (although the percentages were smaller with small firms and increased with number of employees).

In terms of direct productivity benefits, the Datamation survey indicates that management opinion lists "faster development" and "longer program life" as the top two items. The latter presumably refers to decreased maintenance costs during the useful operation of the system, while the former is so vague as to give little indication as to exactly what will lead to faster development cycles. "Enforcing standards" and "better documentation" were the next two important CASE benefits listed, which do correspond to some of the functions selected by software engineers in the Norman and Nunamaker studies, specifically to "project standardization" as well as other items.

Collectively, these studies point to areas where working analysts and managers expect productivity benefits from the use of CASE tools. However, all agree that no published literature quantifies such benefits. What may be lacking is an analytical approach to an evaluation of the costs of the software production process. What are the sub-task groupings which make sense for this production process, and what are the cost relationships among them? In other words, what is the flow of costs through the production process, and how could they be affected by the introduction of a CASE tool? It is just such an analysis which this study undertakes beginning in the next chapter.

Chapter 3: A Model of Software Production Tasks and Their Costs

The model presented in this chapter is unique in its ability to clearly portray the cost interactions between sub-tasks in the production process. The costs of the inputs necessary to complete each sub-task are laid bare in a manner more evident than previous analytical techniques. An added point of value is the graphics capability of this cost graph modelling. Since the model is based on directed acyclic graphs, production sub-tasks and their cost relations can be portrayed in a manner which makes intuitive sense to managers. The model presented here is adaptable to many different production environments, since it can be derived from already existing software development life cycle standards. As well, the very process of constructing a cost graph will benefit software producers by encouraging them to clarify cost sub-centers and their interaction during production. In many situations, the need for data collection and analysis of actual costs then becomes apparent.

In this chapter, we will:

- examine the overall objectives of this model
- show how graph theory is applied to software production modelling
- explain the methodology which is used to construct cost analysis graphs

In the course of this discussion we will clearly delineate what the model constructed is intended to include and exclude, and we will justify our contention that it adds a useful and unique view of the production process.

Model Objectives

The objective we have in modelling is to describe and inform the costs

associated with the production of software systems. As well as illustrating the costs at particular production steps, our model depicts the cost interactions between production steps. In particular, we are able to show the cost of the inputs requisite to carry on production at each particular sub-task of the software production process. Such cost interactions are overlooked by common software cost estimation schemes, but are of great interest when examining the potential effects of an information commodity or CASE tool on the production process. The cost model described will be able to determine the cost of the inputs necessary to carry out a particular task, as well as the cost of performing that task. Thus, if we are considering automating a task through the use of a CASE tool, the model would help determine if the particular task should be automated, or if the graph should be searched back for other nodes with higher production costs (which might result in greater cost reductions). Ripple effects through the project cycle of cost changes will also be more apparent when modelled in this manner.

Cost is a term which must be clarified before proceeding. First, cost must be differentiated from worth, value, and quality. The model being created makes no attempt to value the quality of the software produced, only the dollar cost to the producer of creating it. The reliability of the software product, the resale value of such a product, or the importance of such a product to the user, are all aspects which are outside the scope of this study. Specifically, this means that it is assumed that cost is the only variable factor of interest in the model. Changes in cost shown by the model assume that the output produced is of the same quality. For example, we shall see in the detail of the model that often a software production project creates source code which is placed in a code

library, to be re-used subsequently by other projects. Our model does not include an analysis of the worth of such re-usable code. The error rate of the code over time is not examined. Nor do we look at its size, complexity, market value, efficiency of execution or efficiency of storage utilization. We include only the cost of creating and placing it into the library.

Software cost estimation has been discussed in the previous chapter, where it can be seen that the approach to costs just discussed has both similarities to, as well as differences from, previous software cost estimation techniques such as those of Gotlieb [101] and Boehm [102]. Gotlieb describes software cost as a function, among other factors, of complexity and programmer experience. Quality is not explicitly discussed nor measured. Gotlieb does state that over time, through an analysis of maintenance error corrections, management will be able to modify the percentages of project costs spent in each phase in a manner which will reduce maintenance work, and thus presumably produce better quality systems. But this is an implied quality measure.

Boehm's COCOMO model, as we have seen in the previous chapter, also does not contain a direct quality measure. COCOMO includes factors which indicate the experience level of the programmers and analysts, as well as a factor which indicates how familiar they are with the type of product being created. COCOMO's primary factors for these considerations are ACAP (Analyst Capability), AEXP (Applications experience), PCAP (programmer capability). High factors for capabilities and experience reduce software costs in COCOMO. One could reasonably expect that they will also increase software quality, but the COCOMO model does not include any direct measure of quality.

The COCOMO model does allude to quality in the components which cover the attributes of the software product. These are RELY (required software reliability), and CPLX (product complexity). High ratings for these "effort multipliers" result in higher COCOMO costs. Again, one would expect that products requiring high degrees of reliability would also demand high product quality, but this is not necessarily the case. COCOMO merely says that it will cost more to ensure greater reliability (or to create a more complex software product). For example, reliable software does not necessarily have to be modular, well commented, reusable, or well packaged. However, I think it is more reasonable to state that there is generally a correlation between required reliability and quality in software production. This would imply that Boehm's COCOMO does include some inference of the quality of the product, albeit an indirect indication thereof.

To summarize, the model created in this work explicitly excludes quality measures. This is an advantage in conceptual clarity compared with existing cost estimation techniques. From the above discussion it should become apparent that the model being used in this paper will be of significant value in examining the effects of CASE tool usage in production of software products which are primarily for internal use. It may be of less utility in an examination of production of software packages which are primarily for mass-market resale. This is because in the latter situation factors of quality, size, and efficiency may be more important than the production cost factor. This distinction between products for internal use, versus products for resale, is another important item not addressed by current software cost estimation strategies. The author feels that the model to be demonstrated by the current research can be enhanced to

include quality measures. This potential expansion can be obtained by replacing the cost calculation with a calculation of "added value", which would encompass more varied criteria. Such a modification is not attempted in this study, but is discussed in the final chapter of this text.

Even after the preceding examination of cost and quality, an accountant would demand a further clarification of the term "cost". For example, there are variable costs (especially labor and use of sub-contractors), overhead costs (secretarial support, office rental, electricity, phones, training, etc), and capital costs (capital goods, computers, other machinery). Our model does not attempt to include all possible costs. Most of our participant organizations do not charge projects directly for overhead costs, so such costs have been excluded. Most of the participating organizations do not include capital expenditures in project costs, so these have also been excluded. Specifically, this means that computer hardware and software support platforms for the production process are not included. Most organizations do not account for such items on a per project basis. This leaves the directly budgeted manpower costs of the project, broken down to the various sub-tasks used either as clear stages in the production process, or as cost centers for accounting purposes. If the modelling techniques described in this text are adapted to an organization which uses different accounting practices, different costing strategies can be included without affecting the overall modelling technique.

When analyzing software production, it becomes apparent that the production processes themselves are not necessarily sequential. They may occur in parallel as well as iteratively. The model created in this research draws upon the techniques of network flow theory, and is based fundamentally on the

acceptance of a software development life cycle model, at least across some phases of that life cycle. The model created for this work also differs from previous approaches in significant respects. We are interested in showing the structure of production represented by cost flows, in an organization which performs software development. Thus, we need to model the specific production processes, the engagement in which result in software systems. This is somewhat similar to the initial view embodied in a Work Breakdown Structure as used by Boehm and others. In addition to a listing of these production processes, we would like to see how they are interconnected by their associated costs.

Among others, are some of the questions we would like to answer using our production cost graphs are:

1. What sub-task has the largest production cost?
2. What sub-task has the costliest inputs?
3. What sub-task has the costliest outputs (provides the highest cost inputs to other tasks)?
4. What is the total production cost of this graph?
5. What are the costs of providing output to projects outside this one? (I.e., of contributing to the company code library or master data dictionary).
6. What are the costs of reading input from outside sources? (I.e., of getting data from or verifying the corporate data dictionary or reusable code library).

Perhaps problematical, cost flows between production processes are not always reflected in organizational hierarchies, nor are they certain to be reflected in the organization of work units. But they do leave a trail of paper flow,

telephone communications, meeting schedules, and data communications traffic. Most important, we know that cost dependencies exist when certain inputs are necessary to production at a given stage in the life cycle. The inputs may be measured, perhaps by counting lines of code, numbers of data flow diagrams, or document pages. All such input flows have associated costs to the parent organization. While the production cost graph model does not make changes in the organization hierarchy of production completely explicit, the model does give the capability of inferring factors which affect the organization of production, as will be examined in the next chapter when we see how the modelling technique is applied to software production.

It should be emphasized that the model created in this work is analytical in nature. That is to say, that the model is not meant to be useful as a tool which can direct collection of the most detailed costs, nor is it a tool to impose managerial control structures directly upon the production process. The model may assist in these tasks through its primary purpose: the provision of a greater understanding of the production process, and of the cost relations between production centers which currently exist in a production environment.

The basic unit in this model is the cost of resources, which we look at in three basic ways: as inputs necessary for a production process to do its work, as costs associated with that production task itself, and as outputs to other production tasks. Shortly a detailed presentation is given of the model of the software development process and each of its associated production tasks and their costs.

Modelling

Mathematics can offer a means of modelling a portion of the real world in a way that can provide a unique view of that reality. In a sense similar to the role of fact-based reality in science fiction [103], mathematical models at their best provide a way to expand our understanding about our reality. They can provide insight by illustrating aspects of that reality hard to explore in other ways. Graph theory, or combinatorial analysis, "is the mathematical study of the arrangement, grouping, ordering, or selection of discrete objects, usually finite in number" [104]. Such graph theoretic models have a further advantage that they often form the basis for simulating the interaction of objects under different parameters, which can also be allowed to change dynamically. Such imitation of real world behavior is the basis of many useful tools, such as cost estimation software, behavioral dynamics, and other works. In this way, "Starting with a model which identifies system parameters, it is possible to study the influence of these parameters and their mutual interactions on performance" [105]. The purpose of this research is the creation of that underlying model upon which a simulation could be built.

There exist many systems for which graph models are appropriate. The well known traveling salesman problem is one. A road map would be another, where there exists a set of cities, $N = \{City_i..City_{i+n}\}$. An arc $(i,i+n)$ exists if and only if $City_i$ and $City_{i+n}$ are connected by a road, and the arcs need no direction assuming one can always travel either way on a road. Another appropriate use of graph theory would be to model a document index system, which has a set of documents $\{Doc_i..Doc_{i+n}\}$, with an arc $(i,i+n)$ if and only if the $IndexTerm_i$ references DOC_{i+n} . Other structures well represented by graphs would be molecules and atoms, electrical networks, phone networks, and

discrete-state systems [106][107].

An entire area of graph theory is applicable to the planning and control of large scale projects. Such techniques are called activity networks or general activity networks, with extensions particularly applicable to problems of managerial cost accounting [108]. Activity network theory can be traced to both the CPM (Critical Path Method) and PERT (Program Evaluation and Review Technique), which are both based upon directed acyclic graph structures. Both methods focus on the interdependence of activities and both represent a project as a network of activities, with activities represented typically as the arcs between node "events" representing activity inception or termination. An arc exists where one activity cannot start without the termination of another activity node. However there is no reason why activities cannot be represented as nodes and dependencies represented as arcs, and in fact this is frequently done.

CPM was developed for well defined project planning and control where the time and cost relationships are direct. PERT refined the CPM method in an attempt to cope with uncertainty by introducing time estimates for each activity. An optimistic, pessimistic, and likely time associated with each activity allows the computation of both an expected project completion time as well as a variance used to give the probability that the completion estimate will be achieved. Typical data processing project applications of these graph theoretical models have included planning, scheduling, managerial control, communications, and simulation [109]. CPM and PERT techniques have proven very useful for project management.

Another useful class of activity network graph theoretic models are network flow models. The project management techniques just mentioned (CPM

and PERT) are actually applications of general network flow modelling theory [110]. In contrast to the project management application, general network flow models are most useful for analyzing the transfer of a commodity through different points in a distribution network, represented at each stage by unique finished goods [111]. Thus, network flow models may be used to decide upon an optimum distribution scheme when a company producing widgets has several warehouses and several customers, and must decide upon an optimum distribution and shipping scheme. Or an oil company may use network flow models to evaluate how much oil should be sent through different pipeline systems to the customers.

Such a network flow model is unsuitable when edges represent the non-temporal linkage of sub-tasks during production. For example, a software module sometimes represents partially constructed systems at an early stage in the production process. They are not necessarily discrete goods which are passed on to another node.

A more subtle difficulty with using the network flow model for production costing is the problem of rework during software development. Further complicating the matter is the question of rework induced by errors detected at a later stage. Thus a graph model of the software production process will conceptually contain feedback loops. Such loops can result in a complicated graph structure, and often necessitate the use of simultaneous equations. We introduce methods for the simplification of the graph structure to accommodate the problem of rework, the details of which are discussed elsewhere in this chapter.

Existing software development life cycle models are also problematic.

These models tend to be oriented towards project management aspects of production. In this respect they are similar to PERT and CPM techniques, although SDLC models are usually less complex. The aspects of the life cycle which are modelled for managerial control are not necessarily the same as those which describe the actual tasks performed by technical workers. There is no consideration given to the description of the cost flows connecting production points. Cost flows represent the portion of the output of a task which is necessary for the performance of a second task, which needs that partial production as input for its work. It should also be noted that many software life cycle models do not even specify managerial checkpoints as specific steps in the life cycle model: they are frequently left implicit.

A Sample Production Cost Graph

Now that the modelling theory has been justified, it will be helpful to scrutinize a "mini" example of a production cost graph, after which we will discuss the application of graphs to software development and a methodology for their construction.

The example production environment is the manufacture of iron window bars. These are the burglar discouraging devices so popular in today's urban areas. The production manager of the "Dura-Bar" company would like to visualize the cost flows between production sub-tasks.

The first step is a definition of the sub-tasks. The Dura-Bar production manager is not concerned with selling to the customer, or with making sure that the customers needs are really being met. The production manager is concerned with fabrication of the window bars themselves, her domain being taking the

window specifications and translating them into a complete set of window bars which have been properly bent, welded, painted, and inspected. After inspection, the production manager considers them a finished product, and they are ready for installation, which is outside the scope of interest of the production manager.

Within the production process as just depicted, the production manager is initially able to define the following sub-tasks as relevant to the costs of producing window bar grates:

- blueprint design
- bending and cutting
- welding
- painting
- quality assurance

While reviewing these sub-task divisions, the production manager realizes that during the sub-task of bending and cutting the bars, errors occasionally happen. These mistakes result in the need for rework, during which the bars are re-bent. Our manager decides that the cost of such rework is to be included as part of the cost of performing the bending and cutting sub-task itself.

To be sure these are the relevant tasks, the manager goes out on the floor, both watching production and talking to the workers. She concludes that some revision to the sub-task list is necessary. The sub-task of gathering together the raw iron pieces is an important part of the production process, and should also be included in the cost model. In addition, she had forgotten that the blueprint designers look through old files to see if there are designs they can re-use. New designs are conversely always put away into the files when the window grates are complete. Finally, she recalls that sometimes bars are bent for jobs that are cancelled. These "pre-bent" bars can be included in a new product if they conform to the specified shapes needed. This use of old designs

and pre-bent bars are considered "outside interfacing" costs, since they read from, or contribute to, overall company stockpiles. The production manager feels it will be useful to get a sense of the costs of such interfacing with other projects.

After these revisions, the complete list of sub-tasks consists of these items:

- blueprint design
- read from design files
- gather iron pieces
- find pre-bent bars
- bending and cutting
- welding
- painting
- quality assurance
- deposit blueprint in design files

At this point the production manager is able to produce a preliminary graph which shows each of these sub-tasks, with arcs delineating the cost relationships between the tasks. Her preliminary sketch is shown in Figure 3-1. This sketch shows all the production sub-tasks which this manager has decided are relevant, with the cost connections she feels are important after reviewing her experience in the production process. But it does not yet contain weights on the nodes or arcs of the graph, and these must be determined next.

A determination of the costs of several production sub-task nodes comes easily to the production manager, since during previous projects she has collected data on their approximate costs. She knows that together welding, designing blueprints, and bending and cutting account for nearly three-quarters of the cost of producing a window grate. She is less sure of the costs for painting, quality assurance, and outside interfacing, since no attempt has ever been made to collect data corresponding to these production activities. Therefore she applies heuristics to assign cost weights which seem reasonable. A similar procedure is

followed in assigning cost weights to the arcs output from sub-tasks. For example, after gathering the iron pieces, the manager knows that most of the iron must be bent and cut before welding. However, some 20% of the iron does not need bending or cutting and can pass directly to the welding task. As well, after welding, while most of the product must be painted, a small portion of a common product needs no painting (typically the portions which are fastened with cement into the building masonry for a secure attachment base).

After a review of all these weight parameters, the production manager is able to create the complete production cost graph which is shown in Figure 3-2. In this diagram, the cost of production at each node is expressed as a percentage of the cost of the entire production process. The weights on the arcs represent proportions of the total output of a node. In this manner, assuming a total production cost of \$ 10,000, dollar amounts could be calculated for the cost of production at each sub task node. As well, the manager would be able to visualize the cost inputs to that task, as well as the output of costs from the sub-task to other tasks. Figure 3-3 annotates the example graph of the previous figure to show such dollar amounts.

The resulting graph can now be analyzed, and will provide answers to many of the questions outlined in other parts of this chapter. The manager can see which nodes absorb the highest cost inputs. These nodes are thus more important to the production process than would be apparent solely by examining the cost of performing the sub-task itself. Similarly, those nodes which incur the highest production cost now stand out. And the most costly connections output to other tasks become clear. The manager could leave the basic graph intact, and adjust her assumptions about the allocation of costs to each sub-task node.

For example, if a greater percentage of the project budget were spent on design, how would the resulting graph look? The effects of introducing an information commodity into the production process can be examined. The manager can consider the effect of introducing a computer controlled bending and cutting machine. She can make an assumption about the degree to which it will lower the cost of production at node seven. The effect of such a change on the cost relationships could be examined. It could be that new production sub-tasks will be necessary, and that there will be new cost relationship arcs. Many other questions could be investigated, as mentioned in the previous sections.

Used in this way, the production cost graph is a powerful tool for analyzing the cost relationships of the production process and the effects of changing either the allocations of costs or the automation of a sub-task. This short, and simplified example provided the reader with an understanding of how production cost graphs are constructed and interpreted. We are now in a position to justify the graph theory used in constructing production cost graphs, and to demonstrate their application to the domain of software production.

Application of Graphs to Software Development

The models used in this research were created by combining the framework provided by existing activity network graph theory with existing work in developing software life cycle models. The idea of using graph theory to elicit the effects of information commodities on the production process is due to the work of Mowshowitz.[112] Existing research in software development life cycles has been summarized in the previous chapter of this work. However, as was explained earlier, this base of existing research theory is not sufficient. It

cannot be used to construct the analytical framework needed for a model of sub-task cost inter-relationships. Furthermore, production processes are not necessarily sequential, they may occur in parallel as well as iteratively. The model created in this research is based fundamentally on the acceptance of a software development life cycle, at least in miniature if not for an entire project. The model created for this work differs from standard SDLC descriptions in significant respects. We are interested in showing the structure of production in an organization which performs software development. Thus, we need to model the specific production processes, the engagement of which result in software systems. In addition to a listing of these production processes, we would like to see how they are interconnected by their associated costs. Graph theory provides the modelling framework needed for such an analytic approach.

The graph theoretic basis used in the modelling of production processes used in this study is that of directed acyclic graphs. Nodes represent production processes. A production process is a logical work step, or collection of work steps, in the creation of the product. A production process must have a definable set of outputs, as well as inputs necessary for its work. The set of all production processes from source (project inception) to sink (completion of product) is the set of nodes $V = \{v_1 \dots v_n\}$.

In this model, all steps in the production processes are connected by costs associated with inputs and outputs. This set of arcs or edges have certain characteristics. The directed arc (i,j) exists only if production process $V(j)$ depends for its own work upon some portion of the production taking place at node $V(i)$. Furthermore, this arc is a directed flow. The arcs, as well as the nodes, are weighted by cost. The cost weight of an arc (i,j) is represented by

$W(i,j)$. The weight of the outputs from $V(i)$ are computed by adding the cost of production of the node to the sum of the costs of the incoming arcs. Each arc output from a node has a weight which represents an associated cost factor connected with the necessary inputs it provides to another node. The complete set of nodes and connecting arcs collectively make up the graph.

We are now in a position to define the graph application used in this model. The production cost of a node is a predefined weight in the graph. This weight is expressed as a percentage of the costs of the life cycle phase containing the node in question, and the life cycle phase is in turn a percentage of the total project cost of 100%. The methods of assigning these weights is discussed in the next section of this chapter. In Chapters 4 and 5, examples are shown which translate actual dollar figures for the entire project into figures attached to cost weights of each node and flow. At the present time, in justifying the model itself, we can define the output costs of any node as equal to the input costs, plus the productions costs of that node:

$$P(i) = W(i) \text{ \{the production cost of node } i\}}$$

$$I(i) = \sum_k W(k,i) \text{ \{the sum of the inputs to } i\}}$$

$$O(i) = \sum_k W(k,i) + W(i), \text{ or: } O(i) = I(i) + P(i)$$

In other words, the sum of all outgoing flows from any node to all other nodes is the node's output cost. At any node we can thus say that the input cost is the sum of all output costs on arcs leading to the node, which is a restatement of the earlier conclusion that:

$$I(i) = \sum_k W(k,i)$$

The distribution of costs over the output flows is assigned when the graph is constructed. The assignment of the distribution is discussed in the methodology

section of this chapter. The source node S has no input, and the sink Z has no output. The sink has the special characteristic that its production cost is the negative sum of its inputs. This characteristic of the sink is necessary for the graph to be self contained, as the sink absorbs the total production costs of the project.

The important characteristic of the graph derived from these features is that the project cost is equal to the total sum of the inputs to the sink. If this can be proven, we can be sure that all costs reflected in the sub-tasks are properly carried through the graph without artificial inflation or loss. To prove this, it is necessary to show that the sum of the inputs to the sink does in fact equal the total production costs over all nodes:

$$\sum_i W(i,z) = \sum_{i \text{ not } z} P(i)$$

This follows from the fact that:

$$W(i,z) = P(i) + \sum_k W(k,i) - \sum_{j \text{ not } z} W(i,j)$$

{nb: production cost + sum of inputs - sum of outputs to others}

$$\cdot \sum_i W(i,z) = \sum_i P(i) + \sum_i \sum_k W(k,i) - \sum_i \sum_{j \neq z} (W(i,j))$$

Since

$$\begin{aligned} \sum_i \sum_{j \neq z} W(i,j) &= \sum_i \left[\sum_j W(i,j) - W(i,z) \right] \\ &= \sum_i \sum_j W(i,j) - \sum_i W(i,z) \end{aligned}$$

Therefore

$$\sum_i P(i) + \sum_i W(i,z) = \sum_i P(i) - P(z)$$

finally as proof:

$$= \sum_{i \neq z} P(i)$$

An interesting ancillary derivation from this proof is the fact that if the graph is subdivided, and a new sink is created to absorb all the dangling output flows, the sum of those flows will be the project cost at that point in production. Thus the cost graph is valid over both the entire project as well as subsets thereof (if one assumes all sub-task costs have been incurred at the time of the cut).

The acyclic feature of this graph model simplifies the analysis of the graph, for it makes possible the use of straightforward linear integer programming, as well as the removal of indeterminateness. Any potential cycles have to be removed in order to produce a directed acyclic graph. There are two kinds of possible cycles, each of which is resolved differently. Each of these cases is discussed here in the general graph theoretic sphere. In the subsequent section, we show how they could arise in the particular realm of software production.

A cycle of length three is shown in Figure 3-4. This cycle can be removed in one of two ways. Most simply, the weight of the edge could be incorporated into the weight of the node itself. This means that the cycle is simply eliminated. However, in certain applications, this removes analytic clarity. In these applications, we can introduce the idea of a "dummy" node. The edge would then go to a dummy node before being passed on to a further node, as in Figure 3-5 by node V(2b). These are called "dummy" nodes because they do not represent actual production tasks, and thus have no associated cost weights. They can conceptually be considered as part of the actual production node with which they are associated. It may help the reader to visualize the dummy as an expansion of a production cost node, with the two enclosed in a single phantom

node, as shown by the dotted lines enclosing them on Figure 3-5.

A similar cycle can be seen in Figure 3-6. Two nodes have flows between each other, as well as each having outgoing flows to a third node. But it introduces the same complexity. The solution is similar to the removal of loops: we must introduce a dummy node which receives the "looping" outputs, as shown in Figure 3-7. As shown, in this case two alternatives can be used to redraw the graph. As with the solution to cycles shown in Figure 3-5, the dummy nodes have no production cost and both alternatives are equivalent. Again, it may help the reader to visualize the dummy as an expansion of a production cost node, with the two enclosed in a single phantom node, as shown by the dotted lines enclosing them on Figure 3-7. In this way no cycles exist, and algorithms to calculate edge weights can remain straightforward. These dummy nodes may add complexity to the artistic representation of the model. But they do not add any intellectual complexity as they can simply be ignored conceptually. A production cost graph containing dummy nodes could be displayed to users with the dummy nodes hidden with no loss of accuracy.

We have shown that it is possible to represent a production process by a weighted directed acyclic graph. A natural extension is the attempt to use these graphs to illuminate changes to the production cost relationships induced by an outside agent. The outside agent of interest is the information commodity (defined earlier). A series of questions are of interest when an information commodity is introduced into the production process. How does the structure of the production process change? Structural change would be reflected in the disappearance of existing production processes, or the introduction of new ones. How does the cost of production change? This would be reflected in the costs

of production processes, in the weights attached to the edges, or by a shift in the edges. Edge interconnections could increase or decrease in number and complexity. The two basic components addressed in this manner are therefore structure and cost.

These are some of the questions which the model created in this research will answer. Additional questions are important, although they lie outside the scope of the model. For example, we would like to know how the required capital base is affected? In other words, are there additional infrastructure requirements? Organizational flexibility is also of interest, since it is frequently said to increase dramatically as a result of the introduction of an information commodity.

Methodology for Constructing a Software Production Cost Graph

In this section a methodology is given which may be used to create a software production cost graph for a particular organization. The methods used in this study will also be described, since the cost graphs shown here are amalgams drawn from the situations of all the participating organizations.

The methodology may be summarized by the following steps:

1. Convert the existing SDLC standard into a high level graph.
2. Decompose each phase node into its sub-tasks.
3. Make decisions regarding rework modelling.
4. Elimination of cycles, numbering.
5. Decisions on task inclusion (documentation, interfacing..).
6. Cost allocation.

In the course of carrying out these steps, both management and technical staff should be consulted. This will ensure that the resulting cost graph accurately represents reality and it not just a representation of paper standards which may

not be carried out in the actual production environment.

The starting point for the creation of the cost graph is the organizations system development life cycle (SDLC) standard. From the SDLC standard, it should be possible to draw a high level graph showing the major phases in the SDLC. We have reviewed SDLC models in the literature in the previous chapter, where we introduced the "waterfall" model and discussed its benefits and limitations. Without repeating that discussion, it was found that the participating organizations all used some variant of the waterfall model as the basis for their life cycle standard. Therefore, the author commenced by re-drawing this model as a high level graph, as shown in Figure 3-8. One participating organization's SDLC standard is shown in Figure 3-9. Identifying information has been removed at the request of the firm. For detailed SDLC standards such as that shown in Figure 3-9, the resulting cost graph will exhibit greater clarity if some phases are combined into single nodes in the initial graph.

The second step is the decomposition of major phases into their sub-tasks. This is conceptually similar to the explosion of process bubbles in data flow diagramming. For example, Figure 3-8 contains the task node of program analysis. Based upon the literature, the author's experience, and amalgamating the practices of the participating organizations, the task of program analysis was broken down into these sub-tasks:

- create structure chart
- write module specifications
- design physical files and verify the data dictionary
- assemble above for quality assurance

Each of these sub-tasks should be reviewed, with the goal of creating sub-task

nodes that add the most conceptual clarity to the graph with its purpose of cost relationship modelling kept in mind. When discussing the task of data dictionary verification, it became apparent that to accomplish the task one has to read from the organization's master data dictionary. A decision was made to show this as a separate sub-task, rather than assume it as within the node for data dictionary verification. This was done because tasks of interfacing with outside resources such as data dictionaries and code libraries is important for analyzing the impact of CASE tools. But the breakout is not fixed. This points out that the decomposition must be done with the goal of the cost graph kept in mind.

This decomposition could continue to a more detailed level. A similar task was carried out in an earlier work by this author in the creation of a set of data flow diagrams capturing the flow of data through the systems development process. However, an excessively detailed cost graph seems to add needless complexity to the model constructed for this study. Should a graph be constructed for a commercial organization, the interviews discussed in the next chapter indicate that meaningful cost estimates at a more detailed level are unlikely.

It is important that there is clarity as to what actual production work each sub-task corresponds to. I would recommend that a descriptive text paragraph or set of data flow diagrams be written in order to provide such clarification.

Once each phase has been broken down into sub-tasks, cost connections must be made. This is a crucial step and must be done carefully. If sub-task X does not start until it receives input from another sub-task U, then a cost flow should be drawn from U to X. Sub-tasks which represent only the reading from an outside database have no input cost, as the cost of writing to that database

were incurred outside either this phase, or outside the project. Continuing our example of the program analysis phase, the resulting model would appear as in Figure 3-10.

At this point the question of rework must be addressed. The model may contain no cycles, so rework can be modelled in only two ways. Rework can be shown as separate sub-task nodes at appropriate points in the graph, with the rework task having its own associated costs. This approach is difficult, and requires many more decisions. For example, one will have to determine exactly what proportion of the output is reworked. And at what level rework will be modelled: at the phase level, or at the sub-task level. At the phase level, we might show rework (in the current example) taking place after quality assurance. But rework does not only include rework of program analysis. There may be rework of output from previous phases. Thus, several rework nodes might be necessary. It is unlikely that most organizations have the data necessary to detail an allocation of rework costs so finely.

Alternatively, rework costs can be considered as part of the cost of the task as shown, in which case no distinct rework sub-task nodes are shown. The cost of each sub-task should include a factor which represents re-work. Since the graph does not imply anything about the temporal interaction of the tasks, either method for incorporating rework will suffice.

A decision on which method to include may be a question of how cost accounting is already being carried out by the firm being studied. For the amalgamated model in this study, the latter approach to rework was used. This was because no organization had data on rework costs, and some organizations had strict rules which forbid rework of particular early phases (in particular of

system analysis output).

The fourth methodology step is the elimination of cycles from the graph. This is a purely mechanical step which must be done so that the mathematics of the directed acyclic graph are intact, as discussed in the previous section. "Dummy" nodes are created, which have no production cost and represent no actual production work, to absorb inputs which would create cycles.

In the realm of software development, cycles are introduced in two ways. First, we have the possibility of rework. Rework is introduced when defects are uncovered during quality control processes, and outputs have to be sent back to the very production processes that produced them in the first place. Second, we may have the situation where two production processes must operate concurrently in order to create their outputs, which then flow out to other nodes.

Continuing with the example of program analysis, Figure 3-10 shows cycles between "create structure chart" and "write module specifications". There is another cycle between "create structure chart" and "design physical files - verify data dictionary". These cost cycles do exist in the real world. This is because, for example, module specifications cannot be written until at least some of the structure chart has been drawn. And often, the structure charts are modified as the module specifications are written and reviewed.

Both the cycles in Figure 3-10 are removed when the sub-task node "create structure chart" is expanded with a dummy node. This dummy node absorbs the inputs from the two nodes involved in the two cycles, plus the original output from the "create structure chart" node. It combines these cost flows into a final flow from the "create structure chart" sub-task. There is still only one sub-task "create structure chart" conceptually, and one can consider the

dummy as part of one single sub-task, as shown by the dotted lines in Figure 3-11, where the resulting cost graph for program analysis is shown. Each node is assigned a number sequentially at this point for ease of referencing. It may be useful to leave some numbers unassigned in each SDLC phase, so that as the graph is refined in subsequent steps the nodes already graphed will not have to be renumbered.

Methodology step 5 consists of decisions on "marginal" or often overlooked tasks. An explicit decision should be made on their inclusion or exclusion from the cost model. The basis for this should be not only the organization's current cost accounting practices, but also the software production staff's actual work practices. In the organizations interviewed this decision centered on user documentation. Two organizations explicitly included user documentation in their SDLC. The other four excluded user documentation, the production of which was the job of either the users or a separate documentation group which was cost accounted separately from software production. However, even the two firms which included it on paper did not do any user documentation during software production in practice! Therefore user documentation tasks were excluded from the cost graph created for this study.

Outside interfacing is another important area which should be given explicit review. Is there a library of re-usable code, or a master company data dictionary? If so, costs are incurred for the actual work of depositing code in the library, or reviewing the code library to see if there are routines a given software project could use. These costs, often hard to account for, are recommended for inclusion in the model. This is especially important if, as in this case, the model is a means for evaluating the impact of CASE tools, since

such tools often impact these areas.

The sixth and final step in the cost graph construction methodology is perhaps the most important. This is the allocation of costs to each sub-task. I recommend proceeding in a top-down manner, congruent with the way in which sub-tasks were decomposed. Every manager can certainly start with a gross cost figure for a project as a whole, 100% of the costs. If this is not possible, it is recommended that the cost graph approach be delayed until such time as sufficient project management controls are in place.

Starting from here, we can return to the high level phases abstracted from the organization's SDLC (previously shown in Figure 3-8). It should be possible to obtain percentage figures for costs incurred in each phase, or in some cases for groups of phases. This study obtained such percentages through reviewing the literature on software cost estimation (particularly Boehm and Grady and Caswell). These figures were reviewed with the interview subjects, none of whom had detailed cost data of their own! (This points to one benefit of the cost graph approach: it will encourage the collection of such data). As a result of these discussions, the following initial cost breakdowns were used:

100% Total Project Cost

65% System Analysis and Design

15% Implementation

20% Integration, V&V

The cost allocation strategy then should proceed down towards the individual sub-task nodes in a similar hierarchical fashion. For example, system analysis and design consists of both logical modelling and physical modelling, and the 65% of total project costs must be allocated between them. Such an approach

then produces the following cost allocations:

100% Total Project Cost

65% System Analysis and Design

60% Logical Modelling

40% Physical Modelling

15% Implementation

25% Program Analysis

75% Coding

20% Integration, V&V

One can then continue to the sub-tasks as shown on the graph. Continuing with our example, program analysis is shown above as consisting of 25% of the cost of the Implementation phase, which itself accounts for 15% of the total project cost. For the sub-task of Program Analysis, the following costs were chosen, based on interviews and heuristics:

15% Extract from Master Data Dictionary
 25% Design Physical Files - Verify Data Dictionary
 15% Create Structure Charts
 20% Write Module Specifications
 25% Assemble & Verify - Quality Assurance

In this way, each production sub-task is given a weight which represents the cost of performing that sub-task. This cost weight is a proportion of the cost of an SDLC phase, which is in turn a proportion of the entire project cost. The completed example graph for program analysis in Figure 3-12 is annotated with this information.

The final task during the construction of the model is the assignment of weights to the arcs which represent cost inter-relationships among production sub-tasks. The total output cost from any node was defined earlier as the sum of the input costs, plus the cost of production at the node. This total (100%)

would be represented as a weight of 1 on the arc, if only one cost relation exists. In other words, if only one arc exists from node v_x to v_y , that arc will have a weight of 1. On the graph no weight will be shown for such an arc, as a value of 1 can be assumed.

More commonly, a sub-task node will have cost relationships with one or more other sub-task nodes. In this case, each arc must have a weight less than 1, and the sum of all outgoing arc weights must equal exactly 1 (100% of the total outgoing costs). It is unlikely that data exists for many such cost relationships, and the arc weights may be assigned based on an analysis by an experienced production manager. This is how the weights in this study, to be discussed in the next chapter, were obtained. The exact figures can in any case be adjusted when data is collected after the model is placed in use. The arcs in Figure 3-12 are weighted in this manner.

Summary

The chapter began with a summary of our objectives in creating a model of software production tasks and their costs, namely greater conceptual clarity of cost inter-relationships and the potential impact on production costs when CASE tools are introduced. After a review of relevant graph theory, we discussed the means by which graphs can be applied to software production. We finally presented a step by step methodology for the construction of a software production cost graph.

After a final review with all concerned, the cost analysis graph is complete. Dollar data for the cost of a project may now be allocated across the phases and sub-tasks in the proportions designated on the graph. One important

result is that you are now able, at any chosen sub-task, to see the cost of the inputs necessary to complete that sub-task. You will also be able to see to which tasks the output costs are distributed. The next chapter offers a concrete example of such data. This information is crucial when one is making decisions about the cost relationship impact of automated CASE tools (information commodities).

A detailed discussion of the composite graph corresponding to the organizations interviewed in this study follows in the next chapter, along with an description of the participating companies and the impact of CASE tools.

Chapter 4: Applications of the Cost Graph Model

Having examined a methodology for the construction of a software production cost graph, as well as the graph's theoretical basis, we are now in a position to apply the methodology to actual production environments. We apply the model as a framework for large programming projects involving teams of programmers and analysts. It is not expected that this analytic framework is transferable directly to the production process engaged in by individual programmers on small projects. Sometimes described as craft or "handwork", as distinct from software manufacture, such production is outside the scope of this study (although it is discussed in the final chapter on future research). Six organizations participated in this study, granting the researcher extensive interviews with management responsible for software production. After a description of the interview process, profiles are given of each participating organization. These profiles lead to the generalization of three typical production cost graphs. The cost graph for the most typical SDLC is then presented in detail. In the concluding section of this chapter, an analysis of the impact of CASE tools is given.

Interview Process

Interviews with managers responsible for software production in large organizations were carried out for the following purposes:

- to ensure that the analytical framework of the model makes sense to practitioners;
- to ensure that the model include all relevant production tasks actually

- performed during the course of software production;
- to establish that the cost relations of the model correspond to "real world" situations;
- to begin dissecting the impact of CASE tools upon cost relations in production settings.

Since all of these goals are general and analytical in nature, the interview process was primarily an aid to the construction of the cost graph representation of the theoretical approach to modelling discussed in the previous chapter. This use of personal interviews contrasts with studies which are oriented towards data collection and quantification, which this research does not attempt. The interviews did make clear how better quantification of the software production process could be helpful within the analytical framework of our cost relationship graphs. This point will be expanded upon in the last chapter.

Personal interviews were selected as more appropriate than other methods, such as telephone interviews or mailed questionnaires. Given the complexity of the cost graph model, as well as the role of subject input into the evolution and efficacy of the model, the interaction offered by a personal interview could not be obtained by other means. Such requirements correspond to the analysis of the major factors in interview format selection discussed by Alreck and Settle [113] in one of the foremost texts on the subject. "Personal interviewing is able to win respondent cooperation and hold it for a substantial period. Non-response bias is minimal..."[114] Both these points were certainly proven true in this study. While personal interviewing does introduce a degree of interviewer bias, the advantages clearly outweigh the disadvantages for this research.

Interviews were sought with individuals in a position to evaluate the entire software production process, rather than those who carry out sub-phases of it. These individuals have a multiple project, or company wide, view rather than knowledge of single projects, as perhaps a project manager might have. Therefore we sought persons with titles such as Vice President, Information Systems Director (or Assistant Director), Standards and Controls Manager, or Director of Systems Integration.

Each interview began with the gathering of preliminary data which sought to establish the general nature of the participating company. This data established the average project size, budget, staffing, implementation of structured methodologies, and level of CASE tool use. We also reviewed the company's SDLC standard, and the hardware and software environment. Finally, if available, we reviewed the company's base of cost data regarding both software production generally as well as rework specifically. Actually, little hard cost data was available in any organization interviewed. Therefore much of this information was subjective and not precisely quantifiable. The list of background questions used by the interviewer during these discussions is shown in Figure 4-1.

After the background questioning (some of which was returned to in the course of the interview), between 1 and 2 hours were spent reviewing an illustrative cost graph shown in Figure 4-2. The author verbally walked the interviewee through the model, describing the content of the sub-tasks, their cost relationships, and their relationship to the SDLC. It will be noted that Figure 4-2 is not a true cost graph since it contains cycles, which are used to indicate rework. Figure 4-2 was drawn in this manner for ease of discussion with the

subjects. The full cost graph model presented later in this chapter does not contain cycles and fully corresponds to the modelling theory presented in the previous chapter.

Each interview subject was encouraged to express conceptions, or practices, which differed from the proposed model. If the subject found that part of the framework did not correspond to their company's practices, they were encouraged to express that view. The subject and the interviewer would then jointly discussed which changes to the graph would reflect the subject's practices. In this way, the model shown in Figure 4-2 evolved during the course of the interviews. Figure 4-2 is thus an amalgam of the sub-task cost relationships in all the organizations. No attempt was made to complete cost graphs specific to each individual organization.

After the discussion of sub-task nodes and their cost relationships, the subjects were again asked for cost data associated with each production node. Particular attention was given to the organizations project costing methods, to the information inputs required by quality control checkpoints, and to the management of rework. If CASE tools were not being used, subjects were asked for subjective impressions of the changes in sub-tasks and cost relationships expected upon CASE introduction. If CASE tools were already in use (the situation in only one organization), the subject was asked for some subjective or objective quantification of the changes induced into the development process by such tools.

After a summary of the interview, and an opportunity for an informal exchange of ideas on topics of the subjects interest, the interview was concluded.

Company Profiles

It is crucial for the reader to have knowledge of the participating organizations, and some description of the individuals interviewed, as they are so important to the manner in which the coming sample cost graphs portray sub-tasks and cost relationships. As stated earlier, all the interviewees work in large project environments with multiple programmer/analyst teams. Five firms are mainframe shops, one is large minicomputer user with a small mainframe. Two are public sector and four are private sector firms. In the following pages each of the six participating companies is described separately. Please note that the participants were promised anonymity and therefore identifying data, such as names, is not included.

Company A is a government agency with approximately 450 programmer analysts. The company is just beginning to use and enforce structured methodologies. A new Systems Development Life Cycle standard is meant to enforce the use of structured methodologies, and training in such techniques has recently been increased. No CASE tools are in use, and the company is a mainframe shop just beginning to think about off-loading development work to micro computers and LAN's. The subject interviewed holds the title Director of Systems Integration, and is committed to the use of structured techniques and the eventual introduction of CASE tools.

In this firm, approximately 220 individuals work in the applications group on new product development, with the hardware environment primarily IBM mainframe based, but also including Wang minicomputers and IBM System 38. The average project is budgeted in the range of \$ 250,000 to \$ 400,000 annually for three years. This produces a total project cost of from \$ 750,000 to \$

1,200,000 for the average software product. It was surprising to this interviewer to discover that in this environment absolutely no cost statistics are collected on an ongoing basis, and little is done to track expenditures versus estimates. Some use is made of outside contractors, however this is often restricted to maintenance rather than new product development.

The director interviewed stressed the importance of staff skills and training to the development of new software. It is unclear what proportion of this training is budgeted to particular development projects, as opposed to general overhead. Regarding the sub-task divisions, user documentation is done by the end users themselves, and is not part of the development effort. As well, this firm begins documentation in a later phase than is indicated by the cost graph shown subsequently in this chapter. Documentation efforts commence during the implementation phase. In contrast, the director stated that physical file design is done a phase earlier than our model shows. Prototyping is done on paper, for user interfaces. No formal library of reusable code exists, although some programmers maintain personal code libraries.

Regarding the important question of rework, this company does not permit rework of output from an earlier phase. In other words, rework can only occur within a life cycle phase. This means that defective products are presumably found and corrected within each phase. There are frequent "go/no-go" checkpoints as discrete tasks within phases, at which time personnel from a special department outside the development group make the evaluation. This seems similar to a separate quality assurance function, and the firm intends to continue this strict SDLC style when CASE tools are adopted.

Company B is a government agency which encompasses authority for

slightly more than 1,000 programmers, analysts, and data base analysts. The hardware environment of this agency consists of two IBM 3090 mainframes, an IBM 4381, an IBM 3884, and some IBM micro computers. The software environment is largely COBOL plus some C on the micros, with IDMS the current database (although conversion to DB2 is underway). Some use is made of a COBOL code generator (Accolade) and fourth generation languages (RAMIS and FOCUS).

The Manager of Standards and Procedures was interviewed in a crowded and busy office. This manager is responsible for SDLC definition, dissemination, and enforcement, for all software development efforts. In fact, different SDLC's are being written, each tailored to a different size software project. Project size is described solely in terms of budget allocation, rather than in lines of code or number of modules, and is estimated to be towards the high six-figure range. Subsequent to the budget approval process, little or not cost measurement is carried out, much to the dismay of upper management.

In this company, structured methodologies have been adopted and a training program is in place to ensure that technical staff continue to absorb and use such methodologies. While no CASE tools are currently being used, a committee is actively evaluating CASE tools such as Knowledgeware and IEW, and plans to make an adoption decision this year. Thus, the firm is very interested in developing methods for cost data collection and tool evaluation. The manager interviewed was enthusiastic about the cost graph model as an aid to such cost collection, and felt that the approach of costing by project phase "makes a lot of sense".

Along with the other interview subjects, the manager at Company B

quickly felt so comfortable with the cost graph approach that (s)he made several comments regarding differences in their company's sub-task orderings. For example, at this company test plans are performed during physical file design and structure chart creation. Our model had test planning in the next phase, as we will see later in this chapter. As well, implementation planning is included in the system design phase, while our model shows this in a later phase. Finally, at this company, some data modelling is done during requirements analysis, in other words, outside of our model completely! Overall however, the cost graph model presented to the subject was acclaimed as making a great deal of sense, with cost relationships which clarified the nature of production relationships and correspond to the actual software production environment.

Company C is a large for-profit company with over 150 programmer analysts, all of whom are required to use structured methodologies. As well, most of the analysts use PC based CASE tools for both mainframe and PC applications. Excelerator is the CASE tool used by this firm, a tool which runs on micros only. Average project size in dollars was estimated to be in the range \$ 200,000 to \$ 450,000.

The manager in charge of CASE tool support and acquisition was interviewed. This interview was of special interest, as Company C is the only subject which is making heavy use of CASE tools. In fact, they are known in the data processing community as one of the early adopters of such tools. Thus it was of particular importance to obtain the views of a manager on the sub-tasks necessary for software production after the tool is in place.

In this regards, it was of interest to note that the major differences commented upon by the subject were in the area of coding. We found a new

sub-task, which resulted in a more complex cost relationship structure. Other tasks, both during implementation and earlier, were said to have reduced costs. It was reassuring that the cost graph itself was still comprehensible and served as the basis for a clear and lucid discussion of the changes. It is primarily this interview that resulted in the cost graph to be shown later in this chapter which represents the production cost situation after tool adoption.

Company D is the smallest data processing shop encountered during this set of interviews. A for-profit mail order merchandiser, only approximately 20 programmer analysts are employed in a hardware environment which is all IBM based, primarily on AS/400 minicomputers. There is also an IBM 30XX mainframe on which little new development takes place. While no CASE tools are used, the management of this firm is innovative and thoughtful. The entire staff is well schooled in structured methodologies, and management appears to continually explore alternative methodologies and innovative ideas, such as Information Engineering and perhaps soon object oriented design. The average project size was not divulged, however an estimate of \$ 100,000 - \$ 250,000 would not seem unreasonable in this environment.

At this firm, as at the others, the cost graph approach made immediate and intuitive sense to the manager interviewed, who was the Vice President of Data Processing Resources. He saw no major changes in their practice as compared to the cost graph, and made many useful comments on the costs of sub-tasks and their connections. This manager claimed to have detailed cost information available, although it was not shared with this researcher. This is in contrast to the situations of the larger shops described above, which had little or no cost data although their staffs were some ten to fifty times larger!

Company E is a for-profit data processing consulting firm which employs some 60 programmer analysts. This firm was interviewed because of its experience across a range of data processing shops. While not itself confined to one company or industry, this firm's experiences in many varied situations enable it to have a comprehensive view of the software production environment.

The subject interviewed is a Senior Project Analyst employed by the firm. They engage in contracts with large firms, both private and public sector, usually for long term assignments on large (\$ 250,000 up) projects. Most contracting companies are mainframe environments with little CASE tool experience or use, although this is slowly changing as more companies acquire CASE tools. This analyst felt that the proposed cost graph adequately represented the situations his consulting firm encounters in the field. He felt that often less user participation in production takes place than would be desirable, even though such input is shown in our cost graphs. And often test planning takes place during system design, although we show it later. He concurred with the assessment of other interviewees, stating that little cost data is available in client firms. The analyst interviewed was excited by the cost graph approach as a potential means for "getting a handle on" the actual costs of production, especially if relevant data could be collected as a production takes place.

Classes of Production Cost Relationships

The above company profiles, summarized in Figure 4-3, suggest that software production practices, at least among the participating companies (who are probably a representative sample), seem to have three major points in common:

1. They all follow some SDLC model.
2. They all use, or plan to use, some form of structured methodology.
3. None collect detailed data on the costs of performing production sub-tasks.

Within this framework, three general groups of cost graphs could be discerned as a result of the interviews. Each group represents an instantiation of the general analytic model presented in the previous chapter:

Group 3: Use CASE tools, and are conversant in structured methodologies.

Group 2: Use structured methodologies, but do not use CASE.

Group 1: Use neither structured methodologies nor CASE tools.

The breakdown of interviewed organization into these three groups is summarized in Figure 4-4. Figures 4-5, 4-6, and 4-7 illustrate the cost graphs for each of these cases. We briefly discuss the categories in this section. In the next section, the Group 2 instance is discussed in great detail, where it is used as the detailed example of the implementation of the model described in this work. And in the final section of this chapter, the mapping of a transition from Group 2 to Group 3, and the implications in terms of information commodity theory, are discussed.

This classification into three graph "groups" developed as a result of the responses during the interview by Company A, the only firm not currently strictly enforcing structured development methods. The interviewee in this organization objected to the software production model proposed in Figure 4-2, stating that their production process has had, until recently, many fewer sub-task steps and less complex interdependencies. A larger amount of time and cost is spent in

rework, but at less definite points in the process. It was unknown what proportion of rework costs corresponded to analysis, to design, to code, or to other sub-tasks. With this subject, the model illustrating the "Group 1" situation was developed. This group can be described as the lowest level in software development practices. The implications of this graph for the introduction of CASE tools are discussed in the final section of this chapter.

Figure 4-4 maps each company interviewed into the classification of Group 1, 2, or 3. The majority of the organizations interviewed in this study are classified under the "Group 2" heading. They have a more complex software production process than that exhibited by Group 1 organizations. The data described in the previous section shows that project size is not the causal difference between Group 1 and Group 2 firms. For example, Company D, the smallest firm surveyed, falls into the Group 2 class. What differentiates Group 2 from Group 1 seems to be the adoption of structured development methodologies. Such adoption indicates that the production process has been standardized and codified. This may be better or worse in terms of quality; however, managers apparently believe such methodologies result in a more consistent and controllable development process. Group 2 firms all stated that they had developed clearly demarcated control points during the software production process, at which time sub-task outputs are reviewed and approved or disapproved. The role of documentation in the production process, in particular technical (system) documentation, had been standardized and clarified by Group 2 organizations. For example, data flow diagramming was defined by most as the organizational standard for graphic models created during system analysis. And some enforce Structured English when writing module specifications before

programming may begin. Such standards make it easier to define specific production sub-tasks and their inter-connections.

The transition to Group 3 is premised upon the adoption of a CASE tool. The subject at Company C, the only CASE user, agreed that some development methodology must be in place before a CASE tool can be adopted with any benefit. As we will shortly see, given a comparison of the cost graphs for each model, one has to agree. In fact, when the CASE user was interviewed, the model introduced to them (Figure 4-2) provided a good basis for discussion, as opposed to the Group 1 participant to whom this model seemed fairly irrelevant. The Group 3 subject debated the costs associated with production nodes and interconnecting flows. They did not substantively disagree with the production nodes described, with the exception of the production of actual code. Here, new production nodes appeared as a result of the interviews, as well as substantive changes in cost flow interconnectedness of the sub-tasks.

It is useful to note that this model is descriptive, not prescriptive. Our model may be seen as somewhat analogous to Humphrey's "Worldly (W) process level" of detail in his task oriented model for process management.[115] However our model is not meant to be an aid to process management or practice; as explained, it is meant to be an analytical tool providing insight into production inter-relationships and costs, and the effects of information commodities upon these factors. As Watts S. Humphrey of the Software Engineering Institute points out, "The question...is not 'what is the right way to model the process?' but 'what is the most appropriate way to model this process for this purpose?'"[116]

Cost Graph Analysis of Typical SDLC Detailed

The "typical" cost graph corresponds to the Group 2 class of companies described in the previous section. The graph itself is shown as Figure 4-6. The detailed description of this model instance is complex. We have to accomplish several tasks:

- specify what is performed by each sub-task
- specify how each sub-task is related to other sub-tasks
- apportion production costs to each sub-task
- apportion costs to the outputs from each sub-task

In creating a cost graph, specifying the sub-tasks involved in the production process, and their cost relationships, is the crucial task. These facts are the basis for the graph's structure.

In order to make the graph more understandable in the following pages, we start by reviewing the life cycle model being used as a "typical" base. We then map the sub-task definitions onto each phase of that life cycle. Cost weights and cost relationships are justified in this same discussion.

The typical life cycle used as a base for the cost graph is reproduced in Figure 4-8. This figure considers production tasks at a level similar to Humphrey's concept of a "Universal (U) level" detail of the software development process.[117] It does appear similar to a traditional, task oriented, sequential view of the development life cycle. But it should not imply a strictly sequential process view, and is useful here for clarification of the graph theoretic model instantiation which follows. Remember that this modelling technique is not dependent upon the acceptance of a particular life cycle model, as was made clear in the section on methodology in the previous chapter. However, life cycle

standards can facilitate sub-task identification. Since all the subject organizations do use life cycle models, one is abstracted in Figure 4-8 for clarity.

The overview life cycle in Figure 4-8 begins with "System Analysis". This assumes that a software system has already been requested and is desired by a user. That is to say, a feasibility study has already been completed, validated, and approved, as has a formal requirements definition. Thus, we enter development assuming that the general nature of the product to be developed is known, and that appropriate resources for its production have been allocated and planned, at least for the initial stages of production. Problems associated with such resource planning, cost estimation, resource allocation, and project management are not of explicit concern in the model being created. Only the costs of producing intermediary outputs during product development are of interest, along with the proportional weighing of such outputs as they are needed as input for other production steps.

Operations considerations of previously developed software are excluded, as are questions of simultaneous operations of the old system beside the new one (parallel processing). At the end of Figure 4-8, you will note that nothing is shown past "Verification and Validation". At this point, the software system has been designed, completed (coded), tested, and integrated. This is the point at which development is considered complete for the purpose of our model.

Maintenance is therefore excluded from the model. It has been argued that software maintenance is itself more complex than the original development life cycle. Others argue that maintenance is simply a replication of the software development life cycle, albeit "in the small". In any event, all agree that maintenance is distinguishable from the original creation process of a new

software product, and is for this reason excluded from our model.

Having defined the borders of what will be considered "software development", let us proceed to examine the "wordly" level details of each high level "universal" in Figure 4-8. Our description of sub-task activities follows the general discussions in [118][119][120][121][122] [123] [124] as well as participant organization data. The latter had a major influence upon the definition of the production processes in this graph. What follows therefore is a detailed discussion of the graph theoretic model instantiation presented in Figure 4-6.

"System Analysis" is taken as the start of software production, so, for our model, costs begin here. Specific requirements for a feasible system are needed as input, but, as stated earlier, for the purposes of our graph they are considered outside the scope of actually developing the software system. System Analysis consists of formulating a logical view of the desired system. With substantial user involvement, a graphic process model is created of the desired system's logical functionality. Along with this an information model is constructed, a detailed high level statement of purpose is written, reporting requirements are determined, and initial prototyping occurs. This is the "universal" level, or most general description of the System Analysis phase.

The detailed work of analysis is made up of several sub-tasks, shown in the typical (Group 2) cost graph model as nodes number 3 through 10 (see Figure 4-6 for reference throughout this discussion). Each of these sub-tasks is now described in detail before the next phase is discussed.

Node 1 is the source node, the beginning point for the graph.[125] It is not a production process, thus it has a cost weight of zero [$P(1)=0$]. There is no incoming cost flow as node 1 is the beginning of the graph. There is a

requirement that every network graph has only one start node (as well as only one terminal node).[126] The start node here is connected to five other nodes. The arcs from 1 to 2, 3, 4, 5, and 6 have a cost weight of zero. This conforms with our theoretical explanation of the application of graphs to the software production process, since the sum of outgoing cost flows is the addition of the costs coming into a production process with the cost of that process itself. Since the arcs all have a zero cost, there is no need to assign proportional weights to them.

While nodes 3, 4, 5, and 6 each represent actual production processes in developing software, Node 2 is a "dummy" node similar to the source node of the whole graph, which does not represent a production process and therefore has not cost associated with it. While not required theoretically or mathematically dummy nodes are useful to lend visual or conceptual clarity to a graph without affecting it in any other way.[127] In this model, Node 2 represents the source node for production steps which draw on existing resources from outside of new development production itself.

I have called this function "outside interfacing". For example, during Programming, part of the production of code is the reading of previously produced routines from a master code library. The function of getting this code from the library (finding it, making sure it is the algorithm desired, etc.) has an associated cost for the software development process. This cost, incidentally, is quite apart from the production cost of the task which created and added the source to the code library in the first place. There are other examples of such outside interfacing include reading and writing of the master data dictionary, of the documentation library and of user documentation.

Conceptual clarity is added to this model by clearly delineating where in the production cycle outside interfacing is done. For instance, the master data dictionary may be accessed both during System Analysis and during Programming. From a cost standpoint, our primary interest, the costs of such access, may not be the same in both instances. To reflect this, we do not need a dummy node. However, if we wish to account for these costs clearly, we are really talking about two production cost centers. One is reading the data dictionary during System Analysis, the other during Programming. These activities are different both conceptually and pragmatically, but more to the point the subject organizations think of these costs as part of their respective life cycle "phases". Thus it is more useful to have separate nodes representing each activity. To do this, a dummy node as a source for all of these tasks becomes a necessity. You will notice that Node 7, "Analysis Interface - Master Data Dictionary Extraction", is connected to Node 2 by Arc(2,7), for exactly this purpose.

Node 3 is the first node in the graph which represents an actual production process. It represents creation of a "Logical Graphic Model" of the desired system. The exact set of activities represented by this step would depend upon the methodology in use, but the instantiation represented in Figure 4-6 clearly assumes that some structured methodology is enforced. Thus, in the performance of this task, analysts would be interviewing users, discussing preliminary context diagrams with them, as well as creating data flow diagrams or the equivalent.

The weight of Node 3 (the cost of graphic logical modelling) may be expressed in terms of either the cost of production as a whole (total

development), or in terms of the cost of the life cycle phase it is in. We follow the latter approach, as cost estimation is usually done by phase. This method was explained in the methodology section of the previous chapter. Such an approach allows more flexibility in simulating dollar figures under alternate costing assumptions. All production node weights in this model are expressed in the same terms. The production costs are next to each node identifying number in the graph.

The output of Node 3 has two major uses. First, it is necessary for the production of the information model in Node 4. Since Node 3 also needs input from Node 4, a cycle would be created. This is unacceptable for reasons discussed in the previous chapter. Therefore, "dummy" nodes 8 and 9, which have no production cost or content of their own, are created. These dummy nodes are added to maintain the acyclic nature of the graph. Conceptually, each dummy node may be considered as a single together with its parent node. They have no independent production costs as they do not correspond to any actual activity apart from that of their parent. Node 8 exists solely to incorporate the cost of the inputs Node 3 needs from Node 4. Similarly, Node 9 absorbs the costs from Node 3 needed by Node 4. Such costs are added to that portion of the Node 3 or Node 4 output respectively, which then go on to other production processes which are not part of the cycle which would have been created. In this case, the second output of Node 3 is passed on to Node 10, "Assemble and Evaluate Analysis Output", which will be discussed shortly.

Node 4 is the production of the "Information Model". Depending on the methodology in place, this might include entity-relationship diagramming, Jackson structure diagrams, etc. As was just pointed out, some of the logical models

produced in Node 3 are helpful in this production step. As well, some partial information models created in Node 4 may be helpful in the creation of the logical graphic model. The directed acyclic graph theoretic representation of this situation was explained previously.

Most analysts must create a concise "Statement of Purpose" for review by the users, modelled in Node 5. The intent of this statement is both to clarify the overall goals of the project for the analysts, as well as to ensure that technical personnel and the user have the same overall understanding of the nature and purpose of the system. Thus, this statement can be sent on to Node 10 and is not of major importance to the rest of the production process at this stage.

Current theories advocate creation of prototypes early in the development process. While the extent to which this is actually practiced varies, it is common practice to begin report definition very early, frequently accompanied by screen layouts. The advantages to this were discussed in the earlier chapter on software development models. While costly, it is hoped that early user review will ensure a better system product. Items for review are inputs to Node 10.

Node 7 represents the reading from the master organizational data dictionary carried out during analysis. As discussed a few pages ago, it represents the major outside interfacing production process carried out so far, and is necessary for the completion of the logical graphic model (Node 3) and the information model (Node 4). Thus, the costs of this data dictionary extraction must be added to the production costs of these nodes in the creation of their respective outputs.

There are in fact other activities performed at this stage which could be

considered "outside interfacing". The major one which comes to mind is the large amount of time spent in user interviews in order to create the logical graphic model (Node 3), and in creating the prototypes, reports, and screens (Node 6). For the purposes of this graph, these interviews are considered to be part of the cost of each production process. It would add needless complexity to separate graphic modelling, for example, from interviewing. Such separation would be useful if we wanted to show exactly how the cost of interviews related to the cost of time the analyst spends alone creating the graphic model. It might also be helpful if we wanted to see costs of initial interviews compared to the costs of review interviews for discussing the graphic model. However, this level of detail would obfuscate the overall structure this graph attempts to reveal. Perhaps a greater level of detail will be appropriate in the future if this model is accepted as valid in conceptual mission and is applied by a software producer.

The output from Nodes 8, 9, 5, and 6, can be assembled and reviewed. In a rigid project structure, this would have to be done as an entire group only upon completion of the entire output of all these production nodes. And this is what actually occurs in some environments. However, more common in commercial data processing, and certainly among the participating organizations, is the case where this is done in stages as output is created. This review, which does involve users along with technical staff and technical management is costly. It usually includes somewhat formalized procedures, such as Structured Walkthroughs, for the review of the outputs. Different outcomes from this review are possible. The simplest, and "best", is that everything is perfect and can be sent on to the next production node. However, this is usually not the

case, and some of the outputs must be reworked.

Rework At this point a digression must be made, so that we can explain how this graph portrays rework. The problems involved with modelling rework in the graph theoretic representation were discussed in the previous chapter. Primarily we have the problem of avoiding a cycle, and thus introduce another dummy node. A method for modelling rework within this constraint was introduced in the section on methodology in the previous chapter. Now we must clarify exactly what decisions were made in the construction of the Group 2 cost graph.

The first solution devised but not included in Figure 4-6, was to explicitly show those portions of the product which needed rework after the quality control task of each phase. This approach, for System Analysis, is shown in Figure 4-9 as an example. This graph portion shows that the output from Quality Assurance in node 10 is output to several types of rework tasks, each with an explicit production cost. As well, some output needs no rework and can be shown going directly to node 15, which is the Quality Assurance sink. This sink absorbs all the rework output, plus the portion without rework, and sends it on to other production tasks (without any additional production cost).

However, during interviews it became clear that this conception does not represent the actual production process in the interviewed organizations. Some rework is done before the Quality Assurance task, and for the portion done after this task, no separable data is available on the cost of such rework. This is also the case for rework of earlier phases, if the company allows such rework at all. Such rework takes place, for example, when during program analysis it becomes apparent that portions of the system design must be redone. Many of the

interviewed organizations do not permit this kind of rework. The others consider such rework part of the effort of the original production task.

For these reasons, it was decided not to include nodes in the graph which explicitly model rework. Instead, the costs of rework are made part of the production cost of each sub-task shown in the graph. This means that the output from node 10 (Quality Assurance V&V) can go directly to other tasks, ending the phase of System Analysis. This approach to rework is followed throughout the graph.

In the Group 2 graph (Figure 4-6), we have come to node 10, the end of analysis. While much of the output of this phase is needed as input to further production sub-tasks, a portion of the costs incurred at this point are for writing new entries into the data dictionary, or is used to commence user documentation. Both these components of interfacing to these outside entities are shown in flows from node 10 to nodes 92 and 93, comprising together 14% of the analysis costs in our example.

"System Design" receives most of the output created in System Analysis, along with additional input from "outside interfacing" tasks. This is the stage where the logical analysis of the desired system is transformed into an implementable physical design. This will involve more graphic process modelling, reworking of the information model, an analysis of security, audit and control requirements, hardware and software selection, establishment of response and schedule baselines, more report definition, and continued prototype construction.

The sub-tasks of system design are shown as nodes 21 through 27 on the Group 2 example graph (Figure 4-6 continued). The most costly production

processes here are the development of the logical graphic process model into a physical model, and the refining of the information model. The quality assurance step is a costly production task here, as it is in every phase. The prototypes, screen definitions, and report requirements are enhanced and broadened substantially during this phase, accounting for an estimated 15% of the costs of system design. Finally, minor production costs are incurred in outside interfacing, consisting mainly of reading from the master data dictionary for information modelling.

There are cost relationships between the tasks represented by nodes 21,22, and 23. This is due to the fact that portions of the output from each of these tasks are needed for production by other tasks. Thus the "dummy" nodes described earlier are necessary to avoid cycles in the graph. These dummy nodes are 24, 25, and 26 respectively. Again, while they decrease the intuitiveness of the graph's appearance, they give it the necessary mathematical characteristics. Conceptually, each production node and its dummy are treated as one. Thus node pairs 21 & 24, 22 & 25, and 23 & 26 respectively are viewed as single production sub-tasks. As with the previous phase, costs of rework are included in the production costs of each sub-task and are thus not modelled as separate nodes.

In the example graph, fully 30% of the costs of production of system design do not contribute directly to production during program analysis, the following phase. Ten percent is shown as the weight of the cost flow from node 27 to node 81, integration testing. Ten percent is spent in writing more, and more detailed, items into the organization's master data dictionary. And the other ten percent is the cost of providing material for further writing of user

documentation by outside technical writers. This leaves the 70 percent which represents the cost of providing the necessary inputs so that program analysis production sub-tasks can do their work.

Program Analysis In the program analysis phase, the final preparations for coding are begun. Structure Charts or hierarchy charts of modules are created, physical file and database design is done, program control parameters are defined, module specifications are written, and the program module arrangement is reviewed for coupling and cohesion. At this point extensive use will be made of data dictionary and/or code library facilities.

Viewed simply in terms of structure, the program analysis set of sub-tasks appears remarkably similar to the system design task structure. There are the same number of task nodes and cost relationships. However the activities which make up each sub-task node are quite different, as are the weights attached to the various cost flows. We again have reading from the master data dictionary as an activity (node 34), now with a higher cost, reflecting the necessities of sub-task node 35. Node 35 represents the designing of physical files and the verification of the project data dictionary. If a database management system is used for the software project, design of the database will be represented by this task, which is given a production cost weight of 25% of the phase cost in the example.

Structure charts are created during sub-task node 36, which is the first absorber of the seventy percent cost flow coming from node 27. Some of the cost of production of node 36 represents the creation of inputs needed by node 35 (file design) and by node 37, the writing of module specifications as the structure chart is developed. As in the last phase, all the major sub-tasks of this

phase need input from each other in order to complete their respective production. Finally the Quality Assurance task reviews the outputs before further production work. This recurring task represents the strict control points found in the majority of the interviewed organizations.

Only a minor portion of the production during program analysis is necessary for writing to the master data dictionary (10%). The balance of the output from this phase is directly necessary for the actual coding of the software product.

Coding When source code is written, the use of resources such as the project data dictionary and a master reusable code library are more extensive than during any other production phase. Algorithm design of the detailed program logic takes place during this production phase. Unit test data is created and a unit test plan is implemented. Of course, actual programming and program documentation takes place in relation to these other activities. Individual unit testing by programmer and team is accomplished. Quality assurance is a crucial task, the accomplishment of which has been the subject of an extensive literature on program structure, testing, and proofs of correctness (some of which is also very relevant to system integration).

"At last you're ready to write programs, and things begin to happen. ...your hands will be full tending to daily problems that no amount of planning can avert." [128] In this context, it is interesting to note the change in the structure of the graph at this point, covering nodes 50 through 65. There are more nodes than previous phases of the graph, and more cost flow connections. This means that more sub-tasks are being modelled, and the cost relations between them are complex. There are several possible explanations for this.

The first could be that the tasks involved in programming are more widely understood than the tasks which make up other production tasks during the development of software. Second, this might be the case at least for our interviewed base of companies. Third, it also might be that programming is itself more complex than earlier production phases. This research cannot provide a definitive solution to the question of why programming appears to be more complex, although intuition may indicate that programming has traditionally been a topic of much research and discussion. The direction of attention towards other parts of the software production process has occurred more recently.

Even so, programming tends to rely on individual work, often isolated, and has often conflicting goals of correctness, readability, testability, re-usability, and consistent productivity.[129][130][131] This is the area which has been given a great deal of attention by the developers of CASE tools, the impact of which we will analyze in a subsequent section.

Since the content of the programming phase sub-task activities should be apparent from the node descriptions, at least for the readership of this paper, space will not be taken with a detailed description of the tasks. Several dummy or sink nodes are again introduced to preserve the acyclic character of the graph. The assignment of production cost weights is presumed to be more accurate in this section of the graph. This is due to the presumed greater familiarity of the managers interviewed with these sub-tasks, although in all cases no quantitative data was available. Remember that the cost of rework is included in the production cost weight of each node. This makes sense, noting that the costliest tasks are coding (30%), algorithm design (20%), and quality

assurance (20%). These are the very tasks which would be most intimately involved with coding rework.

System Integration The final phase before leaving new software development is System Integration, Verification and Validation (see nodes 77 to 84). Program units are validated and verified, an integration test plan and integration test data are finalized and used for final validation. In addition to the expected inputs from earlier software development production steps, copies of the finalized user documentation are brought in from the documentation group. It is at this point that final entries to the master data dictionary are confirmed. And the master code library will receive the output of thoroughly tested and verified code which will benefit the rest of the organization. This especially is a benefit to the organization, the cost of which can be a significant portion of the development effort.

It has been reported that both managers and programmers exhibit resistance to extensive system testing.[132] This was not evident in the interviews conducted for this research; in fact, promotion of these tasks was evident. However, we did not interview lower organizational levels where such resistance might be pronounced. Theoretically, testing should be both planned and carried out by a separate group not involved with the design or programming of the system.[133][134] Such separate units were not evident in the companies surveyed. Even if implemented by separate persons, such activities and their costs would still be part of software production and be charged to particular projects, in accordance with our example graph.

While the literature has extensive discussions of unit verification and validation, and of system integration, we were unable to clearly define sub-tasks

that make up these activities with our company base. Therefore these activities are not modelled in great detail. They do have cost relationships, and it is clear that system integration testing often results in the need for further unit V&V before system testing can be completed. Therefore nodes 82 and 83 are added as in other parts of the graph to avoid cycles. This test phase both reads from and writes to the master data dictionary and the re-usable code library. In all interviews, it was pointed out that code cannot be placed in the master code library, where one exists, until the task of unit V&V has been passed.

With the completion of system integration or unit V&V at node 84, we consider production complete. The model now collects together the output of the software product from this node, along with costs of outside interfacing to the code library, to the master data dictionary, and to the user documentation group. All these production centers provide input to the graph's sink node 95. As proven in the last chapter, the sum of these inputs to the sink must and do equal the total cost of production. A study of the costs of production at sub-task points, along with an analysis of the costs of necessary inputs to the sub-tasks, will provide some insight into which tasks might be candidates for automation via the introduction of information commodities to production. The insertion of such commodities into the production process will change to cost relations among the sub-tasks, thus changing the old production cost model. An appraisal of such change is the subject of the next section.

CASE Tools: Model Change Analysis

Before analyzing the impact of CASE tools upon the detailed software production cost graph we examined in the previous section, we will return to the

simplified example of the "Dura-Bar" window grate company discussed in Chapter 3. The production cost graph for this example, annotated with dollar figures for a \$ 10,000 project, was shown in Figure 3-3. Let us assume that an information commodity is introduced into the window grate production process in the form of a computer-controlled machine which will automate the bending and cutting sub-task shown as node 6. It is expected that the introduction of this machine will decrease the production costs of sub-task 6 by 20%.

In the first example of the utility of the cost graph approach, let us assume that no other change in the model is introduced by the introduction of the computer controlled bending and cutting machine (although this assumption is faulty, as we will see later). If task 6, which accounts for 25% of the total project cost, is reduced by 20%, this means that the entire project cost is reduced by $(.25 * .20)$ or 5%. Thus, the window grates which formerly had a total production cost of \$ 10,000 now are produced for \$ 9,500.

At the same time, when we analyze the model for sub-task costs as a percentage of the total, we find that we have reduced the percentage of sub-task 6, in our example by 20%, so that the production cost of this node is now 20% $(.25 * .8)$. This means that the production costs of other sub-tasks, which do not make use of the new machine, are increased as a proportion of the total project cost. The percentage increase will be proportional to the original proportion of the total project cost which each sub-task accounted for. For example, node 7 is welding. While it originally accounted for 25% of the project cost, it will now account for some 26.5% of the project cost. This, the relative costs of production among sub-tasks changes as a result of the introduction of the information commodity.

The production cost graph can reflect more change than this. The introduction of a computer controlled bending and cutting machine will certainly create structural change. In other words, new sub-tasks will be created which did not exist previously, and new cost connections will arise. In the "Dura-Bar" setting, this would likely be reflected in a new task of "machine programming" and "machine loading". These tasks would have associated production costs. Thus, a new cost graph would be created which is different from the originally not only in terms of cost allocations but also structurally, in terms of node counts and cost inter-connections.

This such changes in software production are recognized to occur when CASE tools are introduced, it is time to leave the simplified "Dura-Bar" example and return to the detailed cost graph we reviewed in the previous section. Such an evaluation will be made in two ways. First, we present a comparison of example software production cost graphs. Then, we look at just the pre-automation graph, Group 2. We use a hypothetical \$ 500,000 project with assumptions described in Boehm of desirable costs by phase (as shown in Figure 4-11). We will see that the new graph can give insight as to which sub-task nodes are candidates for automation.

Graph Comparison As a first means of comparing the example cost graphs, in Figure 4-10 we have taken the graph representing CASE usage (Group 3), and highlighted in red areas of decreased cost as compared with Figure 4-6, will illustrated the pre-CASE situation. Red is also used to indicate new production nodes which were not present before the introduction of CASE. The reader should note that several old production nodes present in the Group 2 example (Figure 4-6) no longer exist in the portion of the graph representing

coding. This means that the structure of the graph has changed.

Before going further, we must remind ourselves what these graphs leave out. Costs have decreased but we do not measure quality, which may have increased. Information commodity vendors claim quality benefits, although no objective studies prove this. Our graph assumes quality is held constant. Similarly, the volume of software and documentation represented by the project may have increased. For example, although access to re-usable code may be less costly after the introduction of CASE, more code may be re-used than was the situation previously. And a larger or more robust product may be produced at the end of the entire project. Again, this graph assumes the product size is the same.

But back to the analysis of the post CASE situation. The highlighted areas of Figure 4-10 indicate the task nodes which have decreased production costs from the pre CASE situation of Figure 4-6. These tasks are:

- graphic process modelling
- information modelling, including data dictionary access
- prototyping, screens and reports
- coding

The only area of structural change, which results in a decrease of sub-task nodes and a decrease in cost connections, is in that portion of the graph corresponding to the coding phase.

These observations have strong predictive power for production management which is considering the introduction of CASE tools. They are able to see which areas of the production process will be most affected, in a cost factor, by the introduction of the tools. Naturally, those managers with more

reliable and finely grained production cost data will be in the best position to quantify the benefits. But even on a broader analytical scale one can already see the impact.

These findings correlate well with existing research. While little research exists which collects actual cost data, in Chapter 2 we reported the excellent work by Norman and Norman & Nunamaker which documents system analysts' self perceptions of productivity gains from the use of CASE tools.[135][136] Norman reports on a survey of analysts, in which multi-dimensional scaling on a series of questions resulted in an evaluation of the CASE product component parts that contribute the most to system analysts' productivity. These parts, in descending order from the most productive, were: [137]

1. data flow diagram
2. data dictionary
3. project standardization
4. screen and report design

Notice how closely items 1, 2, and 4 on Norman's list correlate with our comparison of the changes from Group 2 (Figure 4-6) to Group 3 (Figure 4-10). These are exactly the CASE components which are used in our graph sub-tasks of producing graphic process models, information modelling, and prototyping.

Item 3 in the Norman study, project standardization, refers to the effect of adherence to analysis and design standards on system production productivity in general. The finding says that analysts feel that CASE tools force greater project standards adherence, and this increases productivity and presumably lowers costs. This finding is interesting when coupled with the analysis in the cost graph examples, which shows a very weak structural correlation between

Group 1 and Group 3 graphs. This was explained earlier in this chapter by stating that organizations with weak use of structured methodologies and weak life cycle standards are apparently poor candidates for the introduction of CASE tools. We had said that a company had to have a "Group 2" cost graph before the introduction of CASE tools should be considered. The results of the Norman study support this analysis.

The structural impact of CASE tools is pronounced during the coding phase, which Norman's work does not address. However, the tasks engaged in during coding are well documented.[138] Unfortunately no research has been published which demonstrably quantifies the impact of CASE upon these tasks. However the literature does suggest an increase in productivity and decrease in production costs when CASE tools are introduced during these tasks. Some of the vendor literature claims productivity increases of 30% to 1,000%. These claims may seem to be disputed by Gilb in his acclaimed book *Principles of Software Engineering Management*:

Most all of the highly-touted productivity tools...have failed to deliver substantial net user-productivity in a well-documented way. This has not prevented them from claiming impressive productivity increases, forgetting that the real end-product is user productivity. My experience in years of trying to substantiate such claims is that:

- they are based on isolated cases and may well be due to uncontrolled factors (the super-programmer on one project, for example);...

- almost none of them meets the conditions of scientific verification via controlled experiments, and statistically valid assertions;...

I do not deny that some of these productivity tools have a beneficial effect.[139]

Gilb focuses on "user productivity" gains, as contrasted with programmer or analyst productivity gains. In other words, he says that increased programmer productivity will not necessarily result in a software product that gives users a

more productive work environment. While this is certainly true, Gilb does agree that such tools may "have a beneficial effect" upon programmer productivity, as indicated by our cost graph example.

Watts Humphrey cautions against putting too much faith in productivity gains through tool usage. Citing data collected in a study by McGarry[140], Humphrey takes the view that tool usage will match higher quality software products, but does not result in greater productivity or decreased faults.[141] Humphrey states "Buyer Beware! The first environment you buy will not work, and the second won't do what you want or need! It is essential to move toward automation, but at least until this business matures, blind leaps will be expensive". He puts more faith in the people in the production process, preferring training, good management, and a clear process model to tools. Humphrey further argues for allowing reduced tool complexity. In other words, tools should focus on small tasks, instead of attempting to automate the entire life cycle. In this sense, he might be seen as arguing for CASE tools which assist in small sets of nodes on our cost graphs. And this is in fact what the market seems to reflect at this time.

For example, the popular Excelerator CASE tool automates data flow diagramming, data dictionaries, structure charts, and screen and report design. This obviously leaves out the programming phase. However, one can buy several excellent additions to this package which attempt to automate programming. They will work stand-alone, as well as with Excelerator, and indeed are frequently purchased separately. Thus, while we might concur with Humphrey in the importance of talented and trained staff, good management practices, and a defined process model, the market seems to agree with him that

less complex tools seen as aids to sub-tasks ("micro tasks" to Humphrey), are what is wanted and can lower production costs.

Sample Costs and the Transition to CASE Another means of using the cost graph to evaluate the possible introduction of an information commodity into the production process, is to examine the graph for nodes which might benefit from the use of automated tools. Several criteria may be used so that high potential nodes will be selected:

1. nodes with high production costs
2. nodes whose cost inputs are high
3. nodes which interface with outside tasks or with already automated tasks

The selection of nodes to examine is made easier when dollar figures are used. In Figures 4-11 and 4-12 we see a spreadsheet printout of the Group 2 example cost graph (Figure 4-6), with \$ 500,000 used as the overall project cost.

Assumptions abstracted from Boehm's *Software Engineering Economics* are shown in Figure 4-11 regarding the allocation of project costs to development phases. The printout of Figure 4-12 shows in dollars, for each node, the incoming costs, the production cost, and each outgoing cost flow. Remember that this graph is only a sample abstracted from the experience of several organizations, and the particular graph for any single organization will no doubt differ from that presented.

If we look over the printout and the graph with the criteria just mentioned in mind, several production nodes will be selected for further analysis regarding potential automation. These nodes are:

1. nodes with high production costs (equal or over \$ 30,000): 3, 4, 10, 22,

23, 27, 80, 81.

2. nodes whose cost inputs are high: 10, 22, 23, 25, 26, 27, 36, 41, 53, 58, 60, 61, 62, 64, 65, 80, 83, 84, 92, 93.

3. nodes which interface with outside tasks or with already automated tasks: 7, 10, 20, 27, 34, 41, 51, 52, 78, 79, 82, 91, 92, 93.

It is useful to re-sort the selected nodes by general task groupings:

Graphic process modelling: 3, 22, 25.

Data Dictionary Usage: 7, 20, 22, 34, 51, 78, 82, 92.

Structure Charts: 36, 53, 57.

Programming: 57, 58.

Re-usable Code Usage: 52, 53, 79, 82.

Information modelling: 4, 23, 26.

Testing, Quality Assurance: 10, 27, 41, 61, 62, 64, 65, 80, 84.

When examined in this manner, we have a picture of tasks which might yield substantial cost reductions if automated. This is more than conjecture, in fact it again is in agreement with analysts' self-perceptions as noted in the preceding section. The first two items on the above list match analysts' top two items for improved productivity through tool usage. The next three items directly relate to the automation of the programming task. And the areas of testing and quality assurance have for years been areas of study and the development of commercial products which have been very successful. IEEE standards have been formulated and well accepted in the military and large commercial firms [142].

Conclusion

Finally, a substantial benefit to the introduction of CASE tools should be

mentioned, even though it is not directly shown in the example cost graphs. This is the ability of CASE tools to output data on the software production process itself. The collection of process measurements from manual processes is very difficult, since it requires additional tasks for data collection. When automated tools are used, data collection is a side-effect of the use of the tool. This is one of the benefits users seek in the introduction of information commodities, and was discussed in Chapter 1. The measurements derivable from CASE tool usage includes complexity data, cost data, productivity data, and documentation data.[143]

In summary, the cost graph comparison before and after adoption of CASE tools provides results which are in general accordance with the established literature on the benefits of CASE usage. In particular, the cost graph analysis agrees with programmer self-perceptions of CASE tool benefits and applications. Through the review in the previous two sections, we see that the cost graph analysis is a unique and useful tool in planning and evaluation for the introduction of CASE tools into the software production process.

Chapter 5: Conclusion

The purpose of this work was to establish an analytic framework for the modelling of cost relationships between sub-tasks within production processes. The theoretical framework of information commoditization was established in Chapter 1. Chapter 2 defined software development, the software production process, and reviewed existing literature on Software Development Life Cycles. Chapter 3 described the application of graph theory to the modelling of sub-task cost relationships, and introduced a methodology for the construction of software production "cost graphs". The specifics of applying the cost graph model to software development were demonstrated in Chapter 4 through the results of interviews with software production managers.

This conclusion begins by confirming that cost graphs are useful under assumptions of varying development phase cost allocations. The utility of cost graphs as applied to software development is then summarized. We then discuss potential generalizations of the cost graph model to other production environments. In the course of these sections the implications of the model, and possible directions for future research, are mentioned.

Cost Graphs and Alternative Phase Costs

In the last chapter, it was demonstrated that cost graphs produced results which were generally consistent with many programmer perceptions of cost and tool utility. That discussion used commonly accepted allocations of project costs to development phases, taken from Boehm [144], namely:

65% for system analysis and design

15% for implementation

20% for integration, verification and validation

However, these assumptions may not correspond to actual costs in a particular organization. As a demonstration of the utility of the cost graph analytic technique, let us examine the cost graph under two different cost allocation assumptions. The first is based on research at AT&T, and the second was suggested during our company interviews.

Fleckenstein Based on research at AT&T, Fleckenstein[145] offers the following allocations:

40% for system analysis and design

10% for implementation

50% for integration, verification and validation

These allocations, summarized in Figure 5-1, are substantially different from those given by Boehm, and were used as input to a spreadsheet containing the cost graph model, shown as Figure 5-2. An analysis of the graph can be made identical to the discussion in the Chapter 4 section on "Sample Costs and the Transition to CASE". Again, we analyze the graph for nodes with either high production costs, or with especially high costs on the inputs needed for sub-task production to proceed.

Clearly, even with Fleckenstein's markedly different allocations, we can still pick out nodes which correspond to the above criteria. To what extent do these nodes correspond to those selected under Boehm's allocations? Regarding Point 1 in Chapter 4, nodes with high production costs, we find that the early nodes of 22, 23 and 27 drop off the list, which is understandable due to the lower portion of costs allocated to these early phase sub-tasks. In fact, the only

new node on the high production cost list is node 84, which represents the final sub-task of quality assurance and system integration. Given Fleckenstein's assumption of 50% of project costs in system integration and test, this should be expected.

Regarding nodes whose input costs are high, some new nodes appear on the list, particularly including nodes 38, 39, 40, and 82. This is perhaps the most surprising result, and it is here that the cost graph analysis can produce insight for production management. If indeed Fleckenstein's assumptions hold for the manager's organization, it would now be important to look at the productivity of these nodes, as well as looking for means of lowering the cost of the inputs to these nodes. Thus, while we may examine different sub-tasks, the cost graph analytical tool is still clearly helpful.

Cost graph analysis under Fleckenstein's allocations indicates that the cost of outside interface reading is up .7%, while the cost of outside interface writes is down 9%. This intimates that proportionally more money spent in testing results in decreased costs as regards contributions to the organization by this project. Again, one would intuitively expect that better testing will result in such a situation.

Interviewee Assumptions Based on interview data, the following phase allocations were suggested:

40% to system analysis and design

40% to implementation

20% to integration, verification and validation

These allocations are summarized in Figure 5-3, with the resultant spreadsheet

printout given in Figure 5-4. Let us again contrast the cost graph with the original analysis in Chapter 4. Regarding point 1, nodes with high production costs, some nodes drop off the list, all in the early analysis and design phase (these are nodes 22, 23, and 27). Three new nodes appear on the list, nodes 54, 56, and 65, all of which are in the phases of programming: algorithm design, program documentation, and code quality assurance. Remember that Boehm allocated only 15% of the project cost to implementation, and our interviewees suggest a figure of 40%! Thus, again, these results conform to intuitive expectations. Under interviewee assumptions, perhaps introducing CASE tools into the coding process is all the more of potential importance.

Regarding the overall list of nodes whose cost inputs are high, the list is exactly the same under our interviewee allocation, as is the list of nodes interfacing with outside tasks. However, the cost graph indicates that the costs of outside interface reading is up 2.3%, while the value of the writes is down by 6%. This result may be surprising, until one realizes that much less of the project budget has been spent in analysis and design under these last assumptions. One could thus expect from the literature and from praxis that the resultant product will likely be less robust than one which spends a greater proportion of the budget on analysis and design. Data collection on actual projects would be a fruitful area for further research and analysis.

Utility of Cost Graphs for Modelling Software Production

Cost graphs have been shown to be useful analytical tools for modelling the cost relationships between software production sub-tasks. We have seen that this analytical framework can be used by organizations which have adopted

different SDLC models, and which have standardized on different analysis and design methodologies. Cost graphs are useful no matter what the job hierarchy or programming language. We saw in the previous chapters that when sample cost data is used, with our cost weight assumptions, the cost graph representation of the introduction of CASE tools closely corresponds to existing studies of analyst self-perceived productivity gains.

Limitations do exist with the cost graph technique as it has been applied to software production in this study. The primary limitation was the lack of quantitative data to support the assumptions made when production cost weights were assigned to sub-task nodes. These weights should not be viewed as fixed, nor indeed as accurate. Any organization seeking to apply this technique should introduce a metrics collection program which will be able to quantify sub-task costs as closely as possible.

Second, a strictly dollar cost limited to a budgeted project cost may be misleading. We said in this study that the interviewed organizations exclude software platform costs, hardware platform costs, and tool training costs (for example) from individual project budgets. However this may not be appropriate in all situations. It may be of interest to allocate to particular production projects these costs among others:

- Training (ongoing versus one time?)
- Skill Components (by job category classifications)
- Managerial Control necessary
- Documentation
- User participation
- Flexibility (hard to define)
- Technology Costs (tools, hardware)
- Interaction Costs (versus self-sufficiency of task)

This is to say that "cost" as we defined it may be too limited, and a useful cost

graph would have to use many, possibly complex, weights in combination as "costs". This would naturally add to the difficulty in creating the graph for a particular environment. However, it in no way contradicts the use of cost graphs as a useful analytic framework.

While the above discussion may be of general interest to managers and financial controllers, there is a more urgent practical reason why cost graph analysis is useful. The software industry is rushing headlong towards automation. This is occurring through the introduction of information commodities known as CASE software tools. All the major computer manufacturers have announced CASE strategies, marking the beginning of a new era in software production. The influence of these computer vendors is such that a great number of software development units are beginning to evaluate, or use, CASE products. Development cost increases and project backlogs are pushing organizations towards CASE. Cost graphs offer the analytical basis for comparing these tools. Furthermore, the development of the CASE tools themselves is in flux. The developing concept is the integration of tools, data, and methods into a continuous process driven by developer needs. But the developers will have an influence on the development of CASE tools, recommending areas for future tool enhancement and helping set enhancement priorities. Cost graph analysis can assist in creating intelligent strategies for CASE advances.

Cost graph techniques as presented in this study can be used in the evaluation of already marketed CASE tools. Through an analysis of productivity gains (cost reductions) expected through the introduction of a CASE tool to the software production process, an analysis could be made which would indicate:

- which sub-tasks in the cost graph would be eliminated

- what new sub-task nodes would appear
- changes in cost flow weights
- creation or elimination of cost relationships

The "cost impact algorithm" would contain cost and structural data which, when applied to an input cost graph, would transform that graph into a cost graph reflecting the effects of introducing that CASE tool into the organization's production environment. Such an extension would be a useful simulation tool in the analysis of CASE tools by software development managers.

Research Directions Using Production Cost Graphs

The cost graph model may be applied to any manufacturing or production environment.[146] The cost graph analytic approach utilized a conceptually simple, directed acyclic graph as the mathematical basis for its construction. Just as network flow graphs have proven applicable to a wide range of environments, there is no reason cost graphs cannot be used to model any organization which exists to create a commodity product.

The production cost graph representation can be used to study questions relating to the organization of production. The complexity of the graph, and hence of the cost connections among production sub-tasks, may be correlated with organizational structure. The structure and complexity of the graph can be compared to the hierarchy of the organization being modelled. This can be extended to a comparison between different companies, each with a distinct organizational hierarchy, and a distinct production cost graph.

The structure of the graph may illuminate the resultant quality potential of the end product. The complexity of the graph could have a relation to the

cost of production, an attribute which could be substantiated by inter-company comparisons as noted in the last paragraph. And the changes introduced in the graph, both structural and cost-wise, may be used to illuminate and validate theories of production management and control.

As they are used in the present study, bearing solely upon cost relations, production cost graphs can be readily applied to any production environment, not exclusively that of software production. All that is required is the ability to define the sub-tasks involved in production, and the cost relationships between them. Some means of defining a total project cost, either from budget or spending analysis, is necessary. And finally there must be enough familiarity with the production process to allocate costs to each sub-task, and weights to the cost connections between sub-tasks.

In this manner, cost graphs are uniquely applicable to a wide range of commodity production environments. The environments which are characterized by familiarity with the components listed in the previous paragraph are commodity production environments. These are situations where large teams of workers must come together under managerial control to create an end product. The applicability of cost graph analysis to other environments, for example to the work of individual craft persons, or to hobby work in the home, remains to be proven and would be a suitable area for further study.

We have seen that the production cost graph technique provides useful insight for managers of software development. It would be useful to implement cost graphs as a simulation tool. Such a tool would portray the cost graph on screen, and allow the user to change assumptions regarding allocations to phases, sub-task production costs, cost flow weights, and cost inter-connections. The

simulation would then dynamically change the flow of costs through the production cost graph and portray the results. In this manner, a manager would use the tool to investigate the effects of varying cost assumptions. Even better, the manager would also be able to make assumptions about how a CASE tool would affect the graph, and obtain a simulation of the resulting cost graph. Such a software tool was in fact requested by two of the interview subjects!

There remains a need for further study which could lead to a refinement in production cost allocations to sub-tasks. This could be accomplished by the collection of data after the application of the cost graph technique to a particular production environment. The present study validated cost graphs as a conceptual technique, but does not engage in quantitative data collection in an organization. It is the intention of the author to carry the study forward into such a sphere as soon as the opportunity arises.

Appendix: Figures

Figure 2-1: IEEE Waterfall SDLC Standard

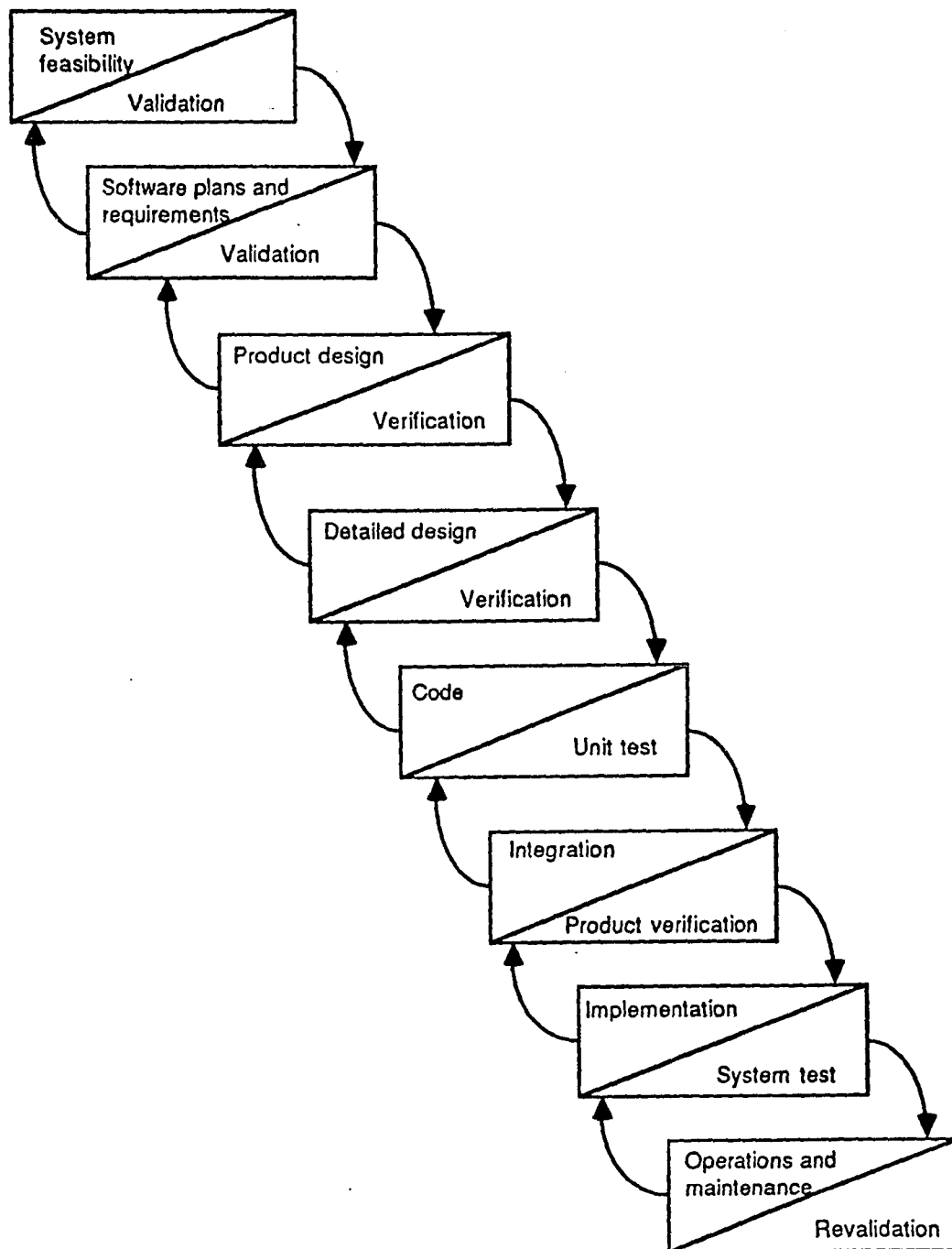


Figure 2-2: Pressman's Prototype SDLC Model

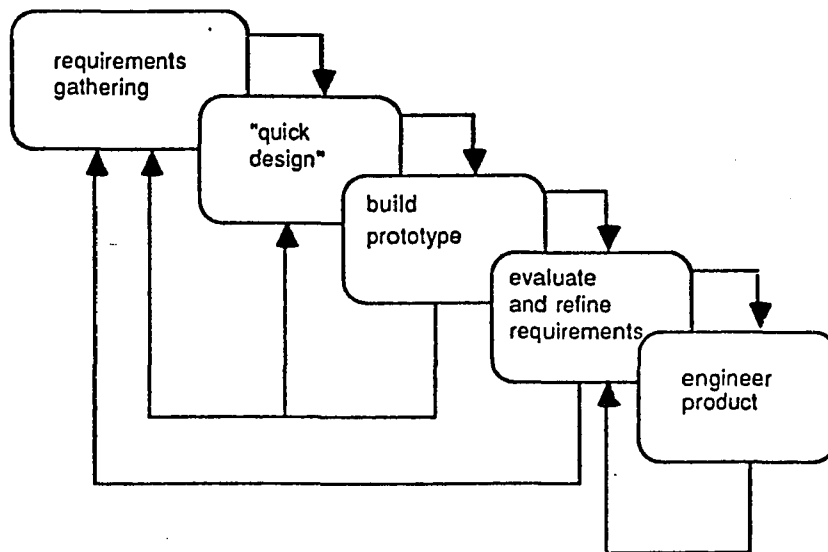


Figure 2-3: Pressman's "Combined Prototype" SDLC Model

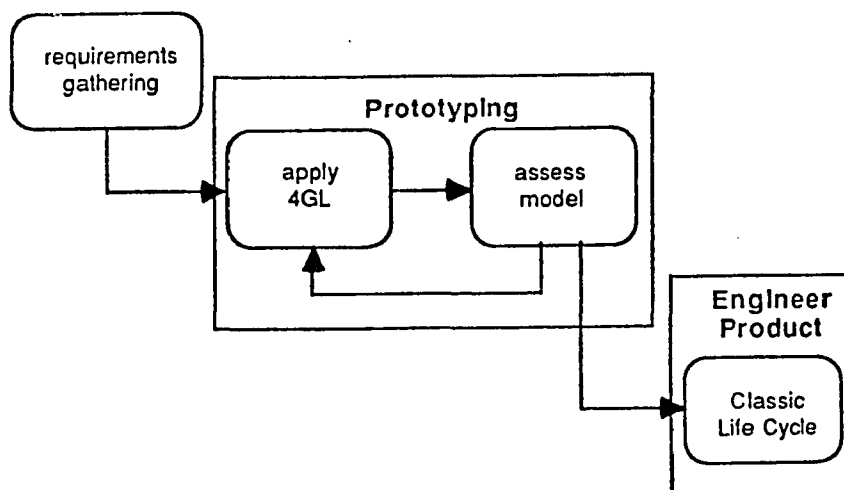
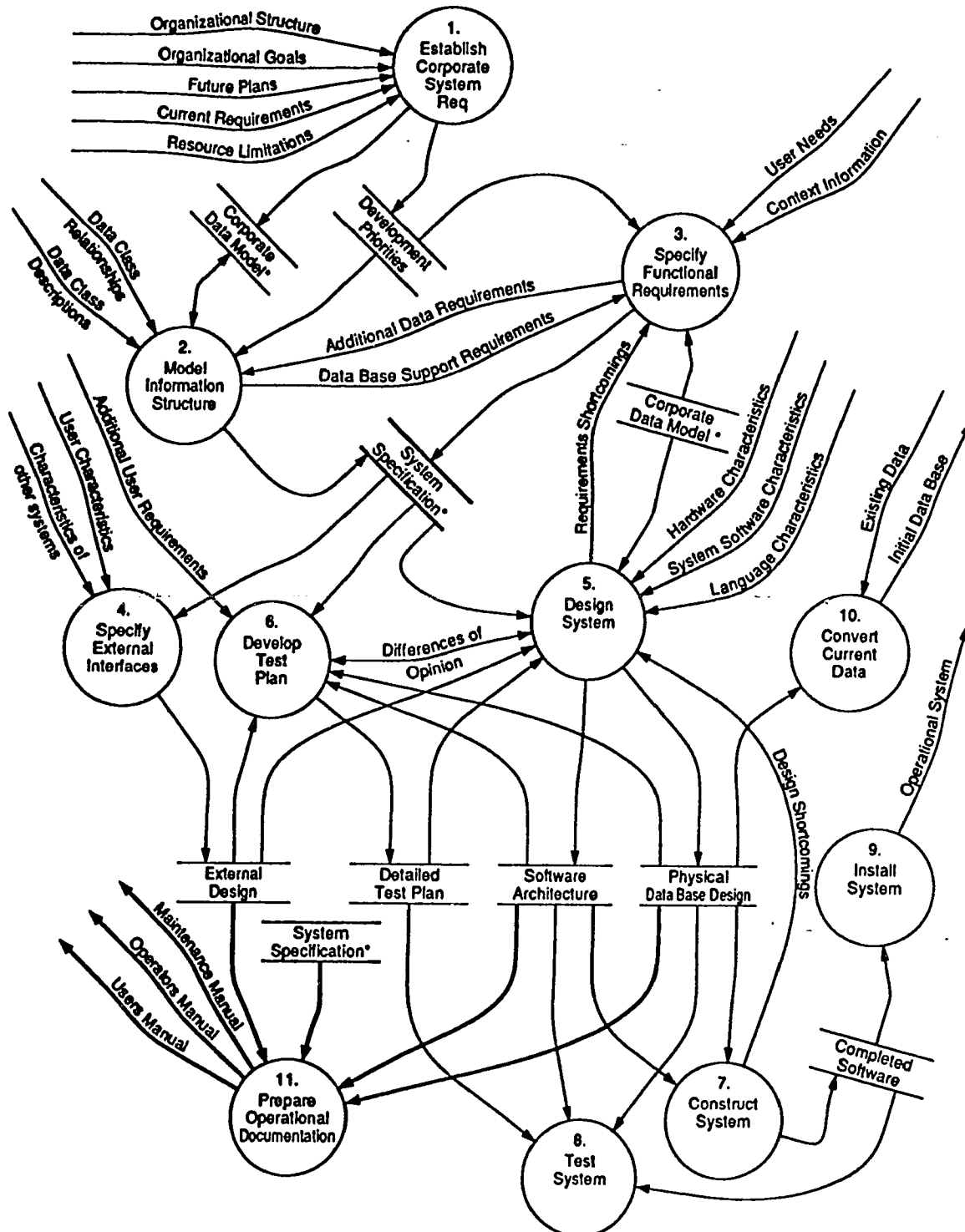


Figure 2-4: StarSys Inc. MacBubbles Suggested SDLC



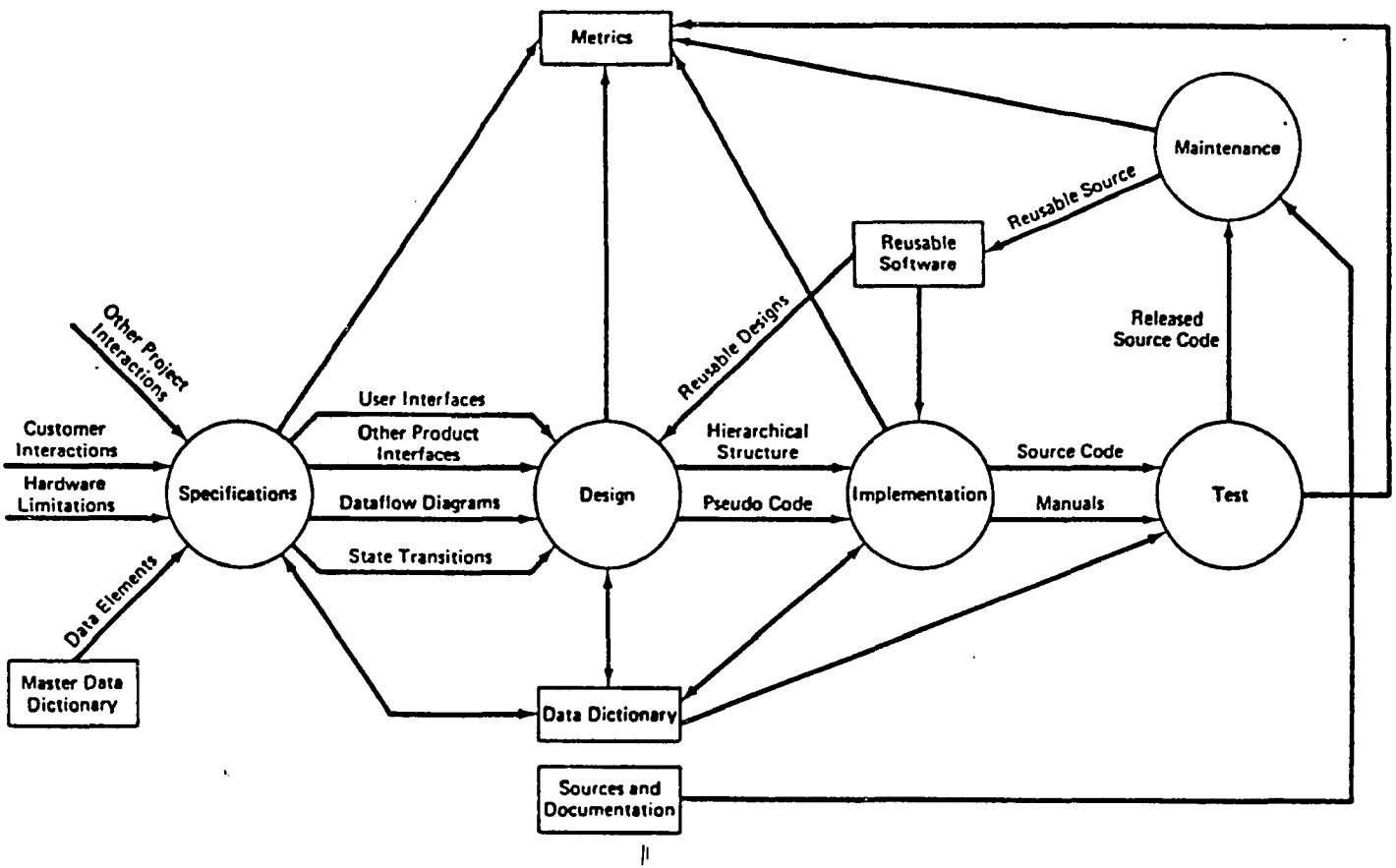


Figure 2-5: Hewlett-Packard Metrics Oriented SDLC

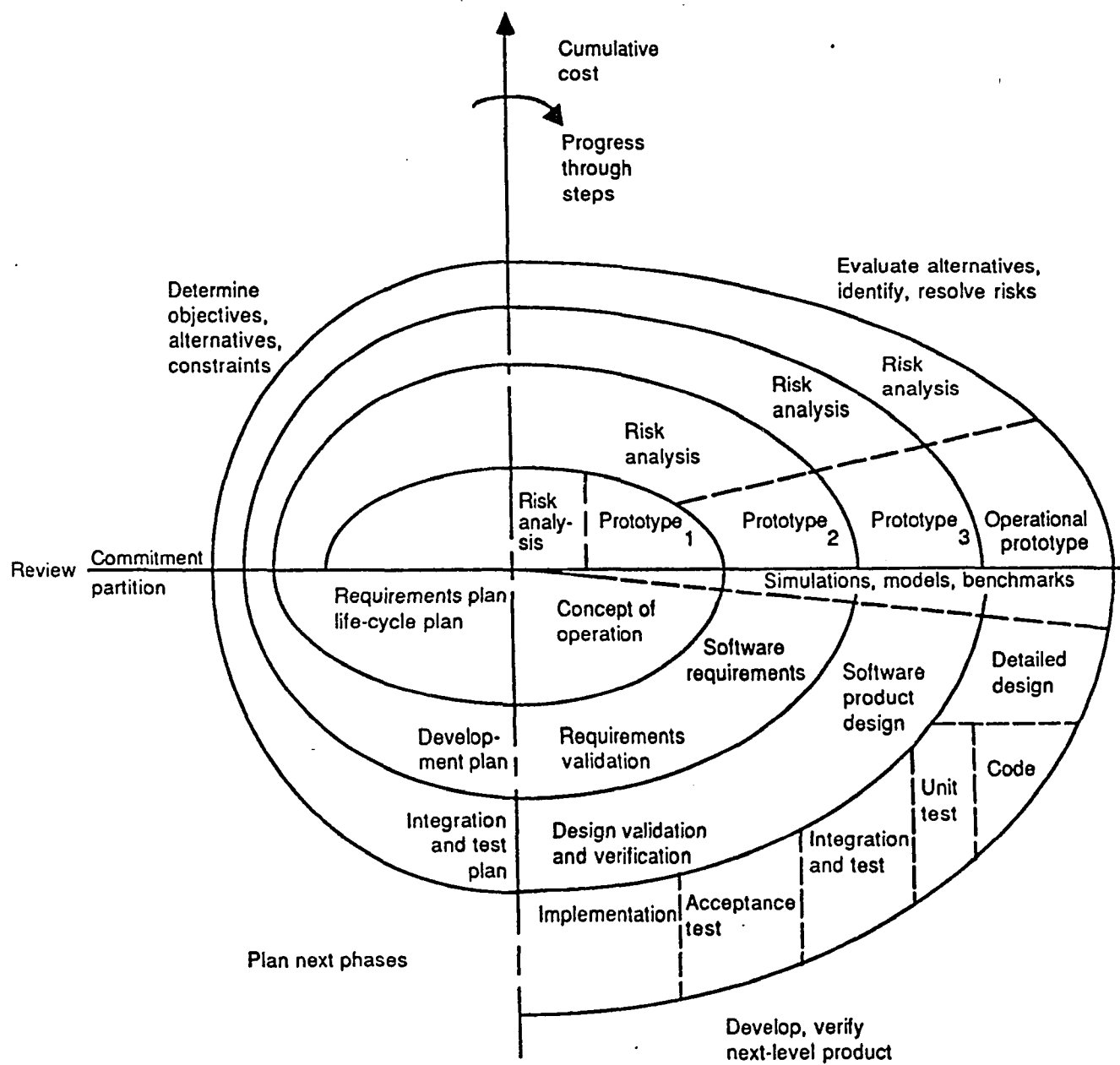


Figure 2-6: Boehm's Spiral SDLC Model

Figure 2-7: Mills' Spiral SDLC Model

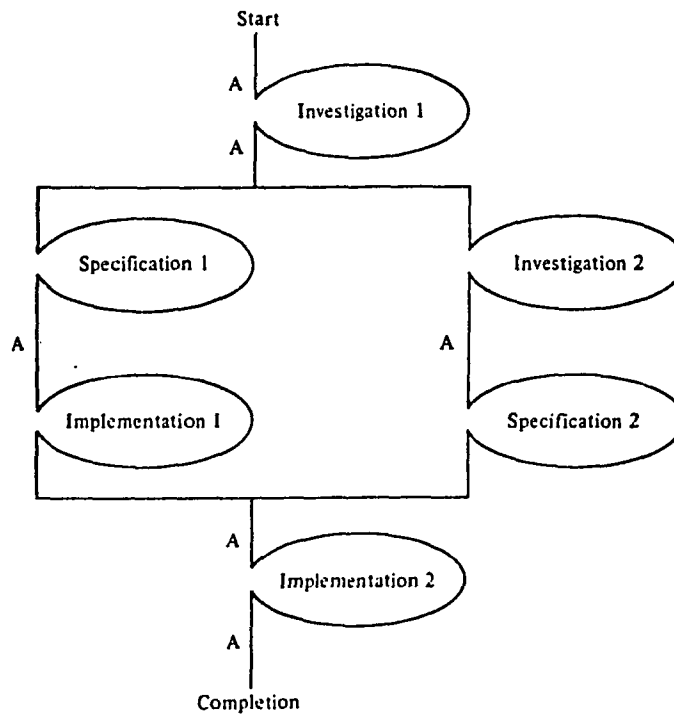


Figure 2-8: Box Diagram of SDLC and Business Relationship

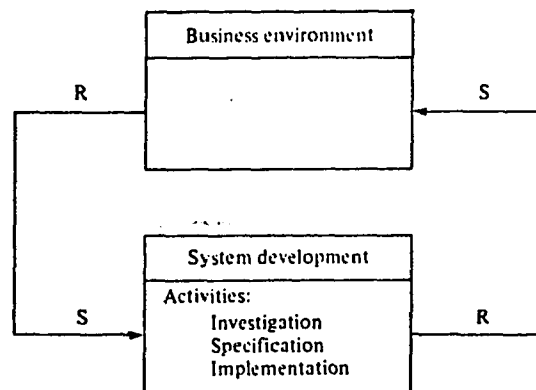


Figure 2-9: Humphrey's Universal Level of the SDLC

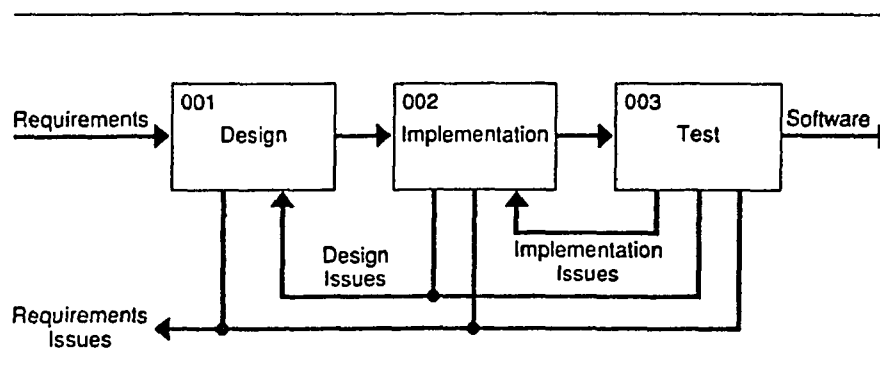


Figure 2-10: Humphrey's Universal Level SDLC Cell Specification

U-LEVEL DEVELOPMENT PROCESS CELL SPECIFICATIONS

Cell:	001	002	003
Entry	Approved requirements, changes, and development plan	Inspected and approved design and changes	Inspected and approved code and changes
Exit	Inspected and approved design and changes	Inspected and approved code and changes	Inspected, tested, and approved software
Feedback In	Design issues	Implementation issues	
Feedback Out	Requirements issues	Requirements and design issues	Requirements, design and implementation issues
Task	Design	Implementation, inspection, and unit test	Testing: Integration, function, system, and acceptance
Measures	Resources, Product: Changes Errors Design Document pages	Resources, Product: Changes Errors Code Document pages	Resources, Product: Changes Errors Code Document pages Test suite

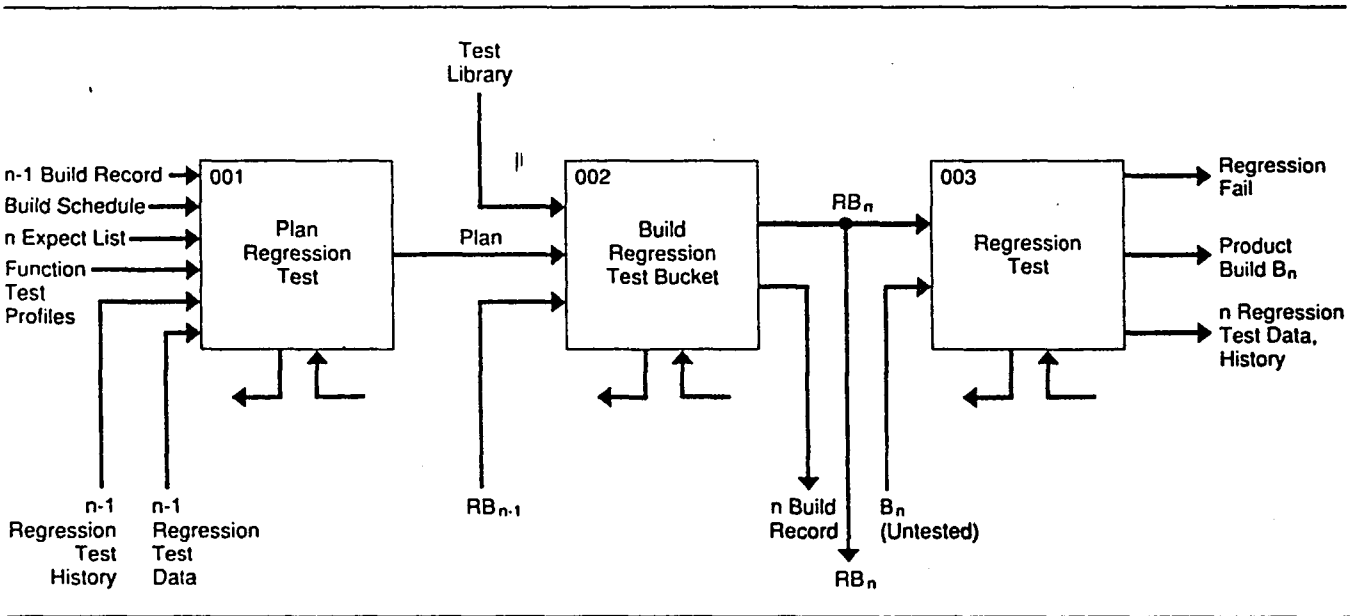


Figure 2-11: Humphrey's Atomic Level (example)

Figure 2-12: Intermediate COCOMO Cost Driver Attributes

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY Required software reliability	.75	.88	1.00	1.15	1.40	
DATA Data base size		.94	1.00	1.08	1.16	
CPLX Product complexity	.70	.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME Execution time constraint			1.00	1.11	1.30	1.66
STOR Main storage constraint			1.00	1.06	1.21	1.56
VIRT Virtual machine volatility*		.87	1.00	1.15	1.30	
TURN Computer turnaround time		.87	1.00	1.07	1.15	
Personnel Attributes						
ACAP Analyst capability	1.46	1.19	1.00	.86	.71	
AEXP Applications experience	1.29	1.13	1.00	.91	.82	
PCAP Programmer capability	1.42	1.17	1.00	.86	.70	
VEXP Virtual machine experience*	1.21	1.10	1.00	.90		
LEXP Programming language experience	1.14	1.07	1.00	.95		
Project Attributes						
MODP Use of modern programming practices	1.24	1.10	1.00	.91	.82	
TOOL Use of software tools	1.24	1.10	1.00	.91	.83	
SCED Required development schedule	1.23	1.08	1.00	1.04	1.10	

* For a given software product, the underlying virtual machine is the complex of hardware and software (OS, DBMS, etc.) it calls on to accomplish its tasks.

Figure 2-13: Analysts' Rankings of 17 CASE Functions

CASE Functions Ranking ($n = 91$)			
Stimulus item	Number of times selected	Pct. selec- ted	Relative choice
Data Flow Diagram	1155	0.79	1.00
Data Dictionary	1128	0.77	0.98
Project Standardization	862	0.59	0.75
Screen/Report Design	857	0.59	0.74
Presentation Graphics	854	0.59	0.74
Analysis → Report Writer	827	0.57	0.72
Analysis → Entity List	728	0.50	0.63
Entity/Relationship data model	726	0.50	0.63
Structure Charts	721	0.50	0.62
Logical Data Model	712	0.49	0.62
Analysis → Graph Analysis	683	0.47	0.59
Project member's communication via [CASE product]	616	0.42	0.53
Structure Diagrams	602	0.41	0.52
Record Layout Generation	598	0.41	0.52
Import and/or Export Facility	577	0.40	0.50
[CASE product] works on both PC and mainframe	453	0.31	0.39
LAN support for the CASE product	277	0.19	0.24
Total Choices	12,376		
Total Choices Per Item	1,456		

Figure 3-1: "Dura-Bar" Example: Preliminary Graph

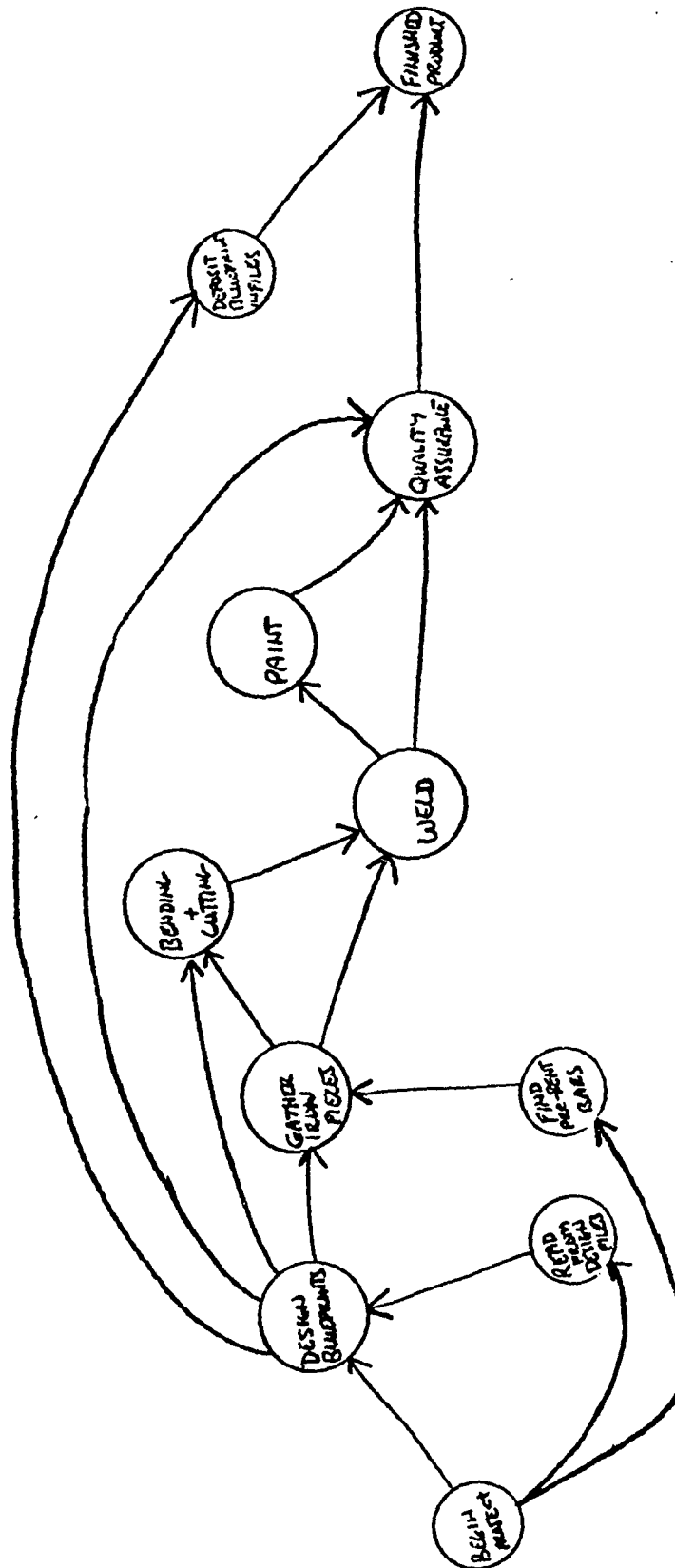


Figure 3-2: "Dura-Bar" Example: Production Cost Graph

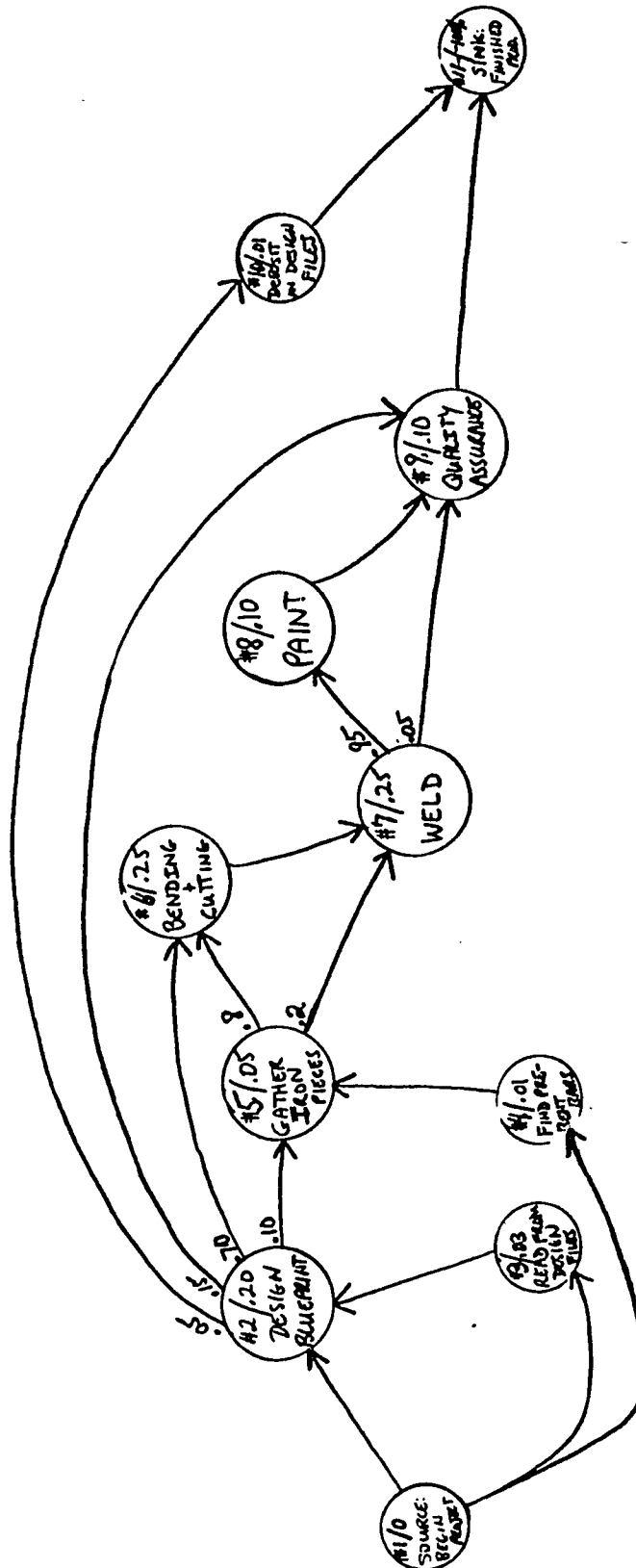


Figure 3-3: "Dura-Bar" Example: Annotated With Dollar Amounts

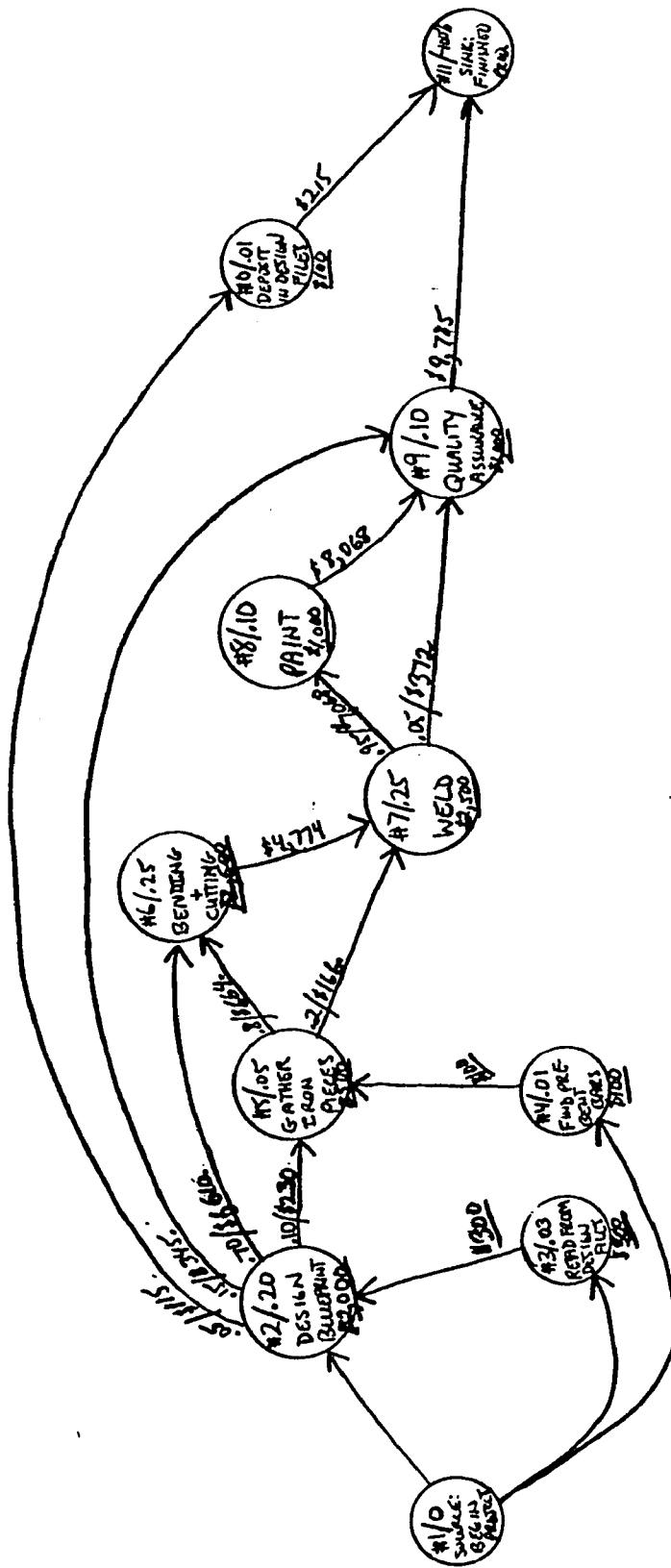


Figure 3-4: A Simple Cycle

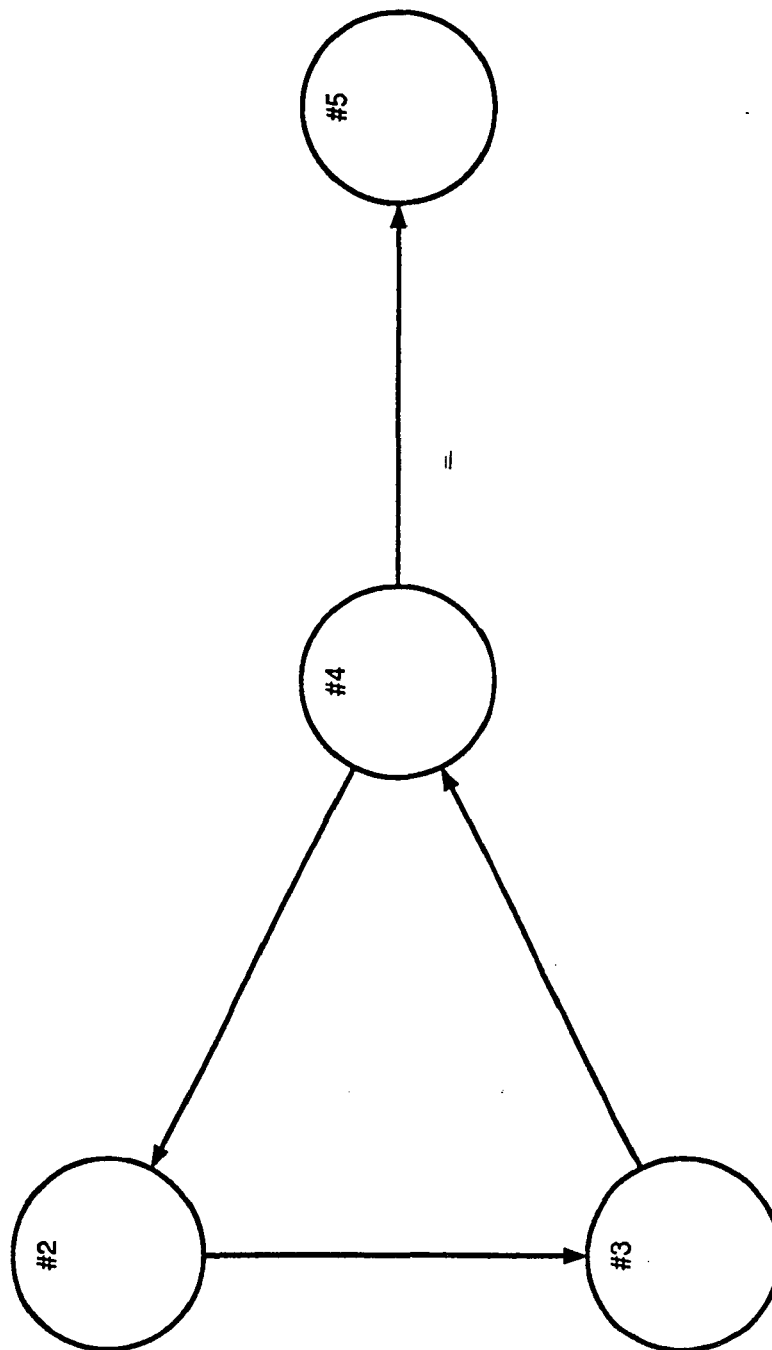


Figure 3-5: Dummy Node to Remove Looping

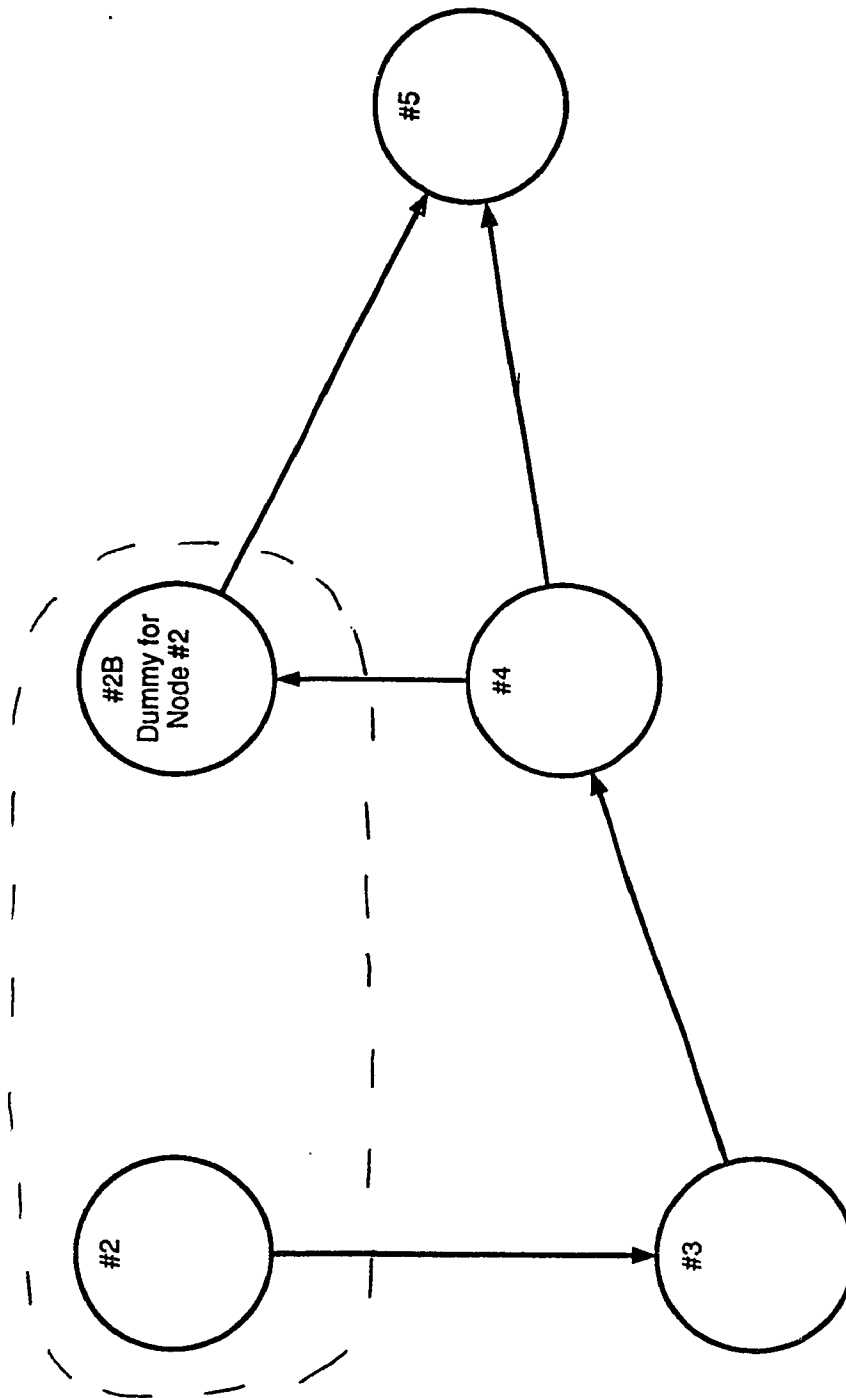


Figure 3-6: A Cycle

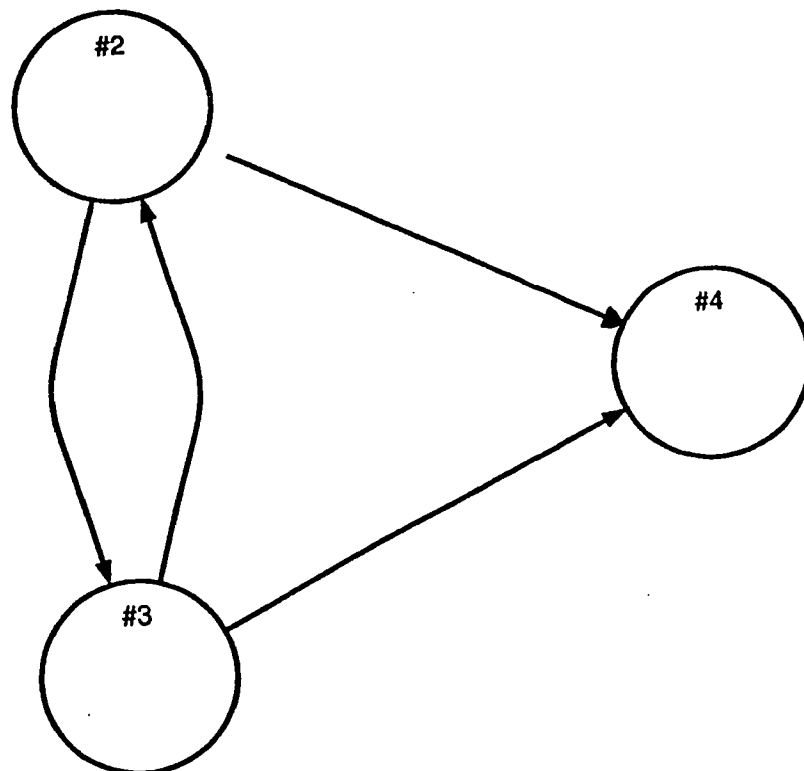
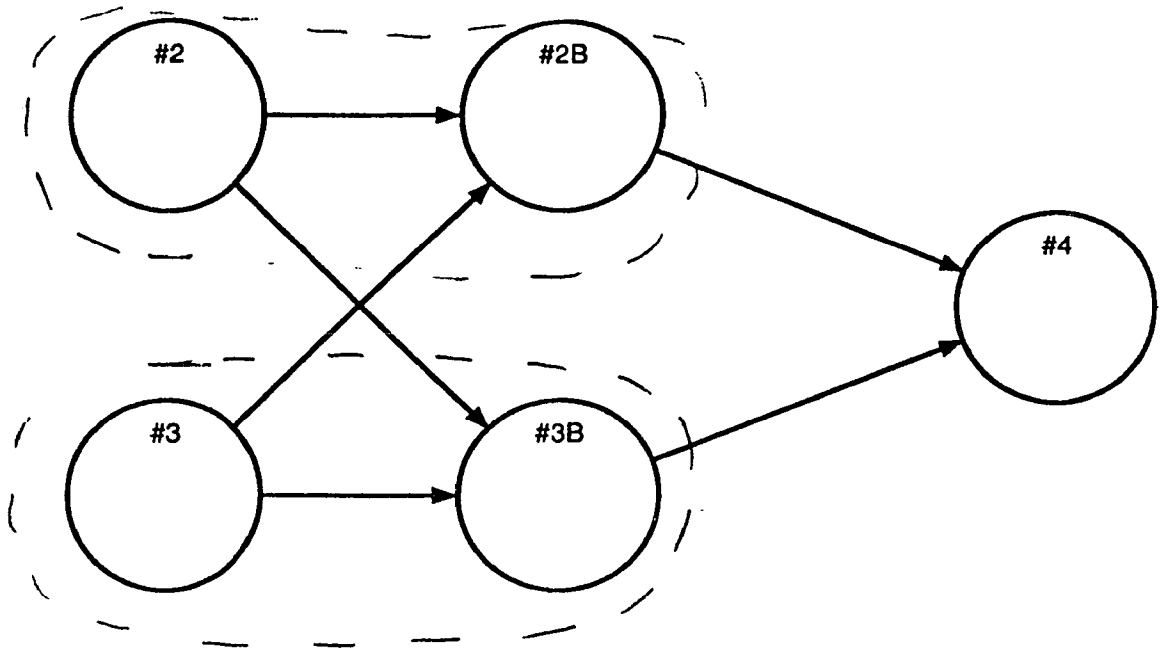
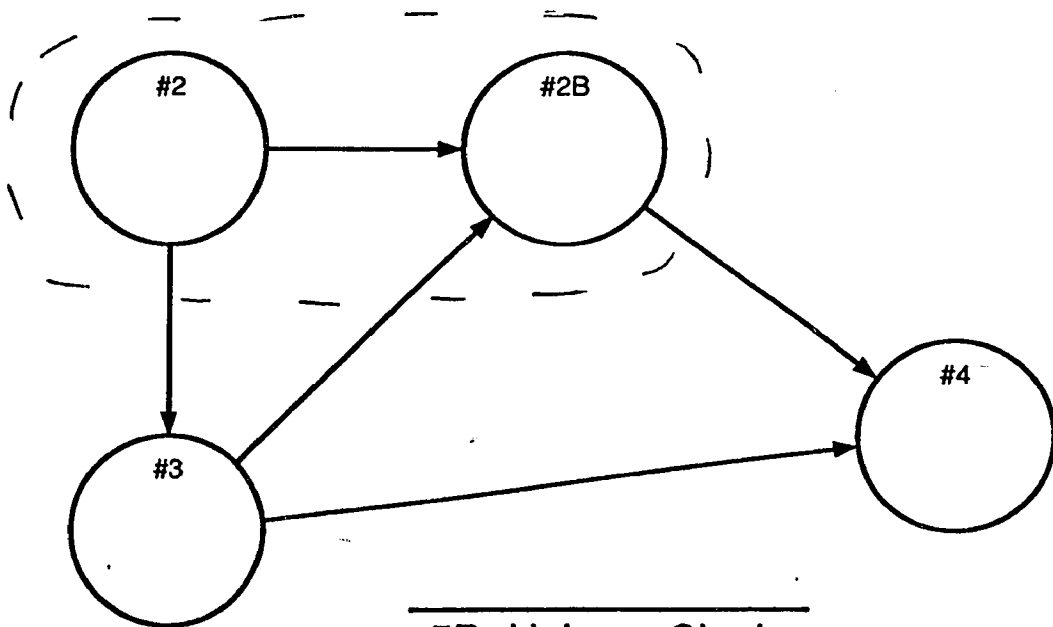


Figure 3-7: Cycle Removed



5A: Using Two
Dummy Nodes



5B: Using a Single
Dummy Node

Figure 3-8: SDLC Overview Based on Waterfall Standard

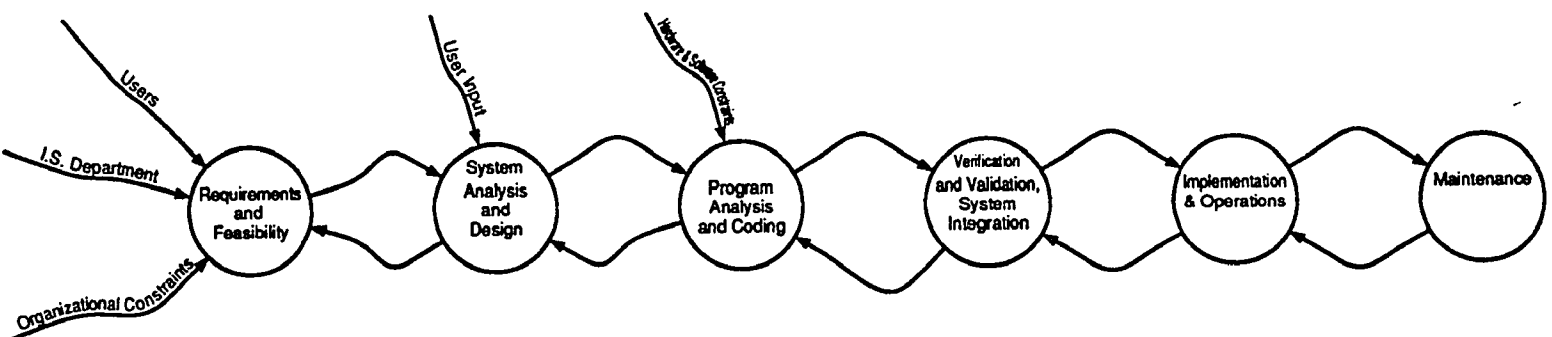


Figure 3-9: Participant Firm SDLC Standard

SDLC Desk Reference Manual
Major Development

VI. Major Development Project
Overview

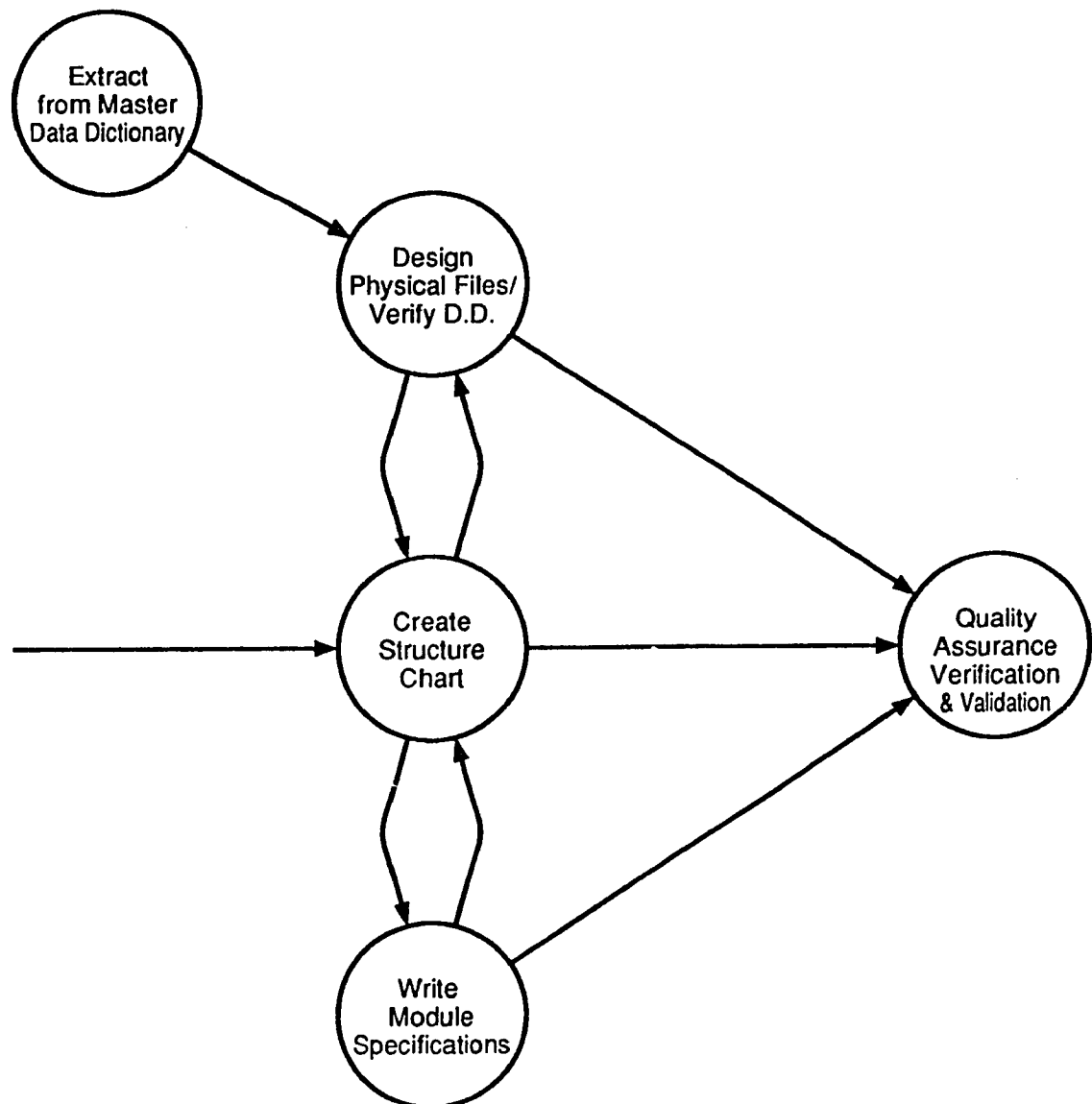
Project Category VI Overview

Phase	Objective	Deliverable
1.0 Determine Requirements	• Describe each of the requirements in terms of inputs, outputs, and processes. • Select or re-affirm the development approach.	Requirements Specification Document
2.0 Conceptual System Design	• Identify the major subsystems (groupings of functional requirements) and their interrelationships. • Re-affirm or revise the development approach.	Conceptual Design Report
3.0 System Architecture	• Develop a logical database design. • Define input and output specifications for each subsystem. • Define telecommunications, security, common functions, and standards.	System Architecture Report
4.0 Detail Design	• Define physical interface (reports, screens, and other interfaces). • Identify and describe programs. • Develop a physical data base design. • Design security and control features. • Re-affirm or revise the project plan.	Detailed Design Specification

Figure 3-9 (continued)

Phase	Objective	Deliverable
5.0 Implementation Planning	• Identify the impact of this project on the use of hardware and system software resource capacity. • Identify conversion requirements. • Plan the systems implementation. • Plan test strategies. • Data Base Administration & Telecommunications Planning	Implementation Plan Report
6.0 Program Design	• Define program structures and describe program modules.	Program Design Specification
7.0 Program Coding	• Create program components (modules) to fulfill requirements.	Program code
8.0 Unit Testing	• Ensure that each program is error-free.	Unit-tested programs in a controlled system test library
9.0 Procedures and User Documentation	• Prepare procedures and user documentation that are needed to support the new functions.	Procedures and User Documentation
10.0 Training	• Develop and provide training for the users and operators.	Trained users and operators
11.0 System (Regression) Testing	• Ensure that the system, as a whole, works according to expectations. • Ensure that the system meets user requirements.	System-tested programs in controlled libraries Open Issues
12.0 Implementation and System Acceptance	• Put a system, acceptable to users, into production.	Installed System

Figure 3-10: Draft Cost Graph of Program Analysis Sub-tasks



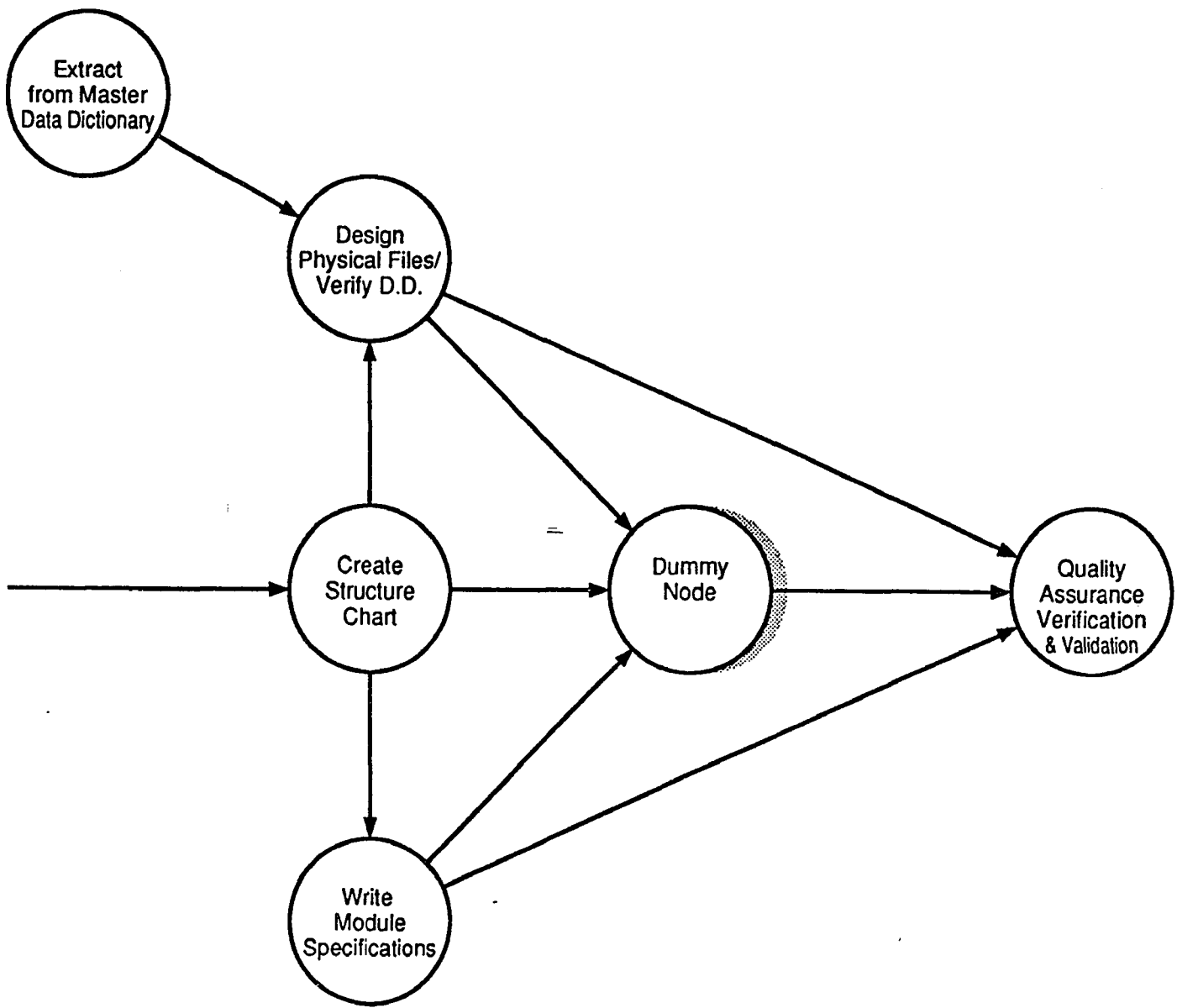


Figure 3-11: Draft Cost Graph of Program Analysis
Eliminating Cycles

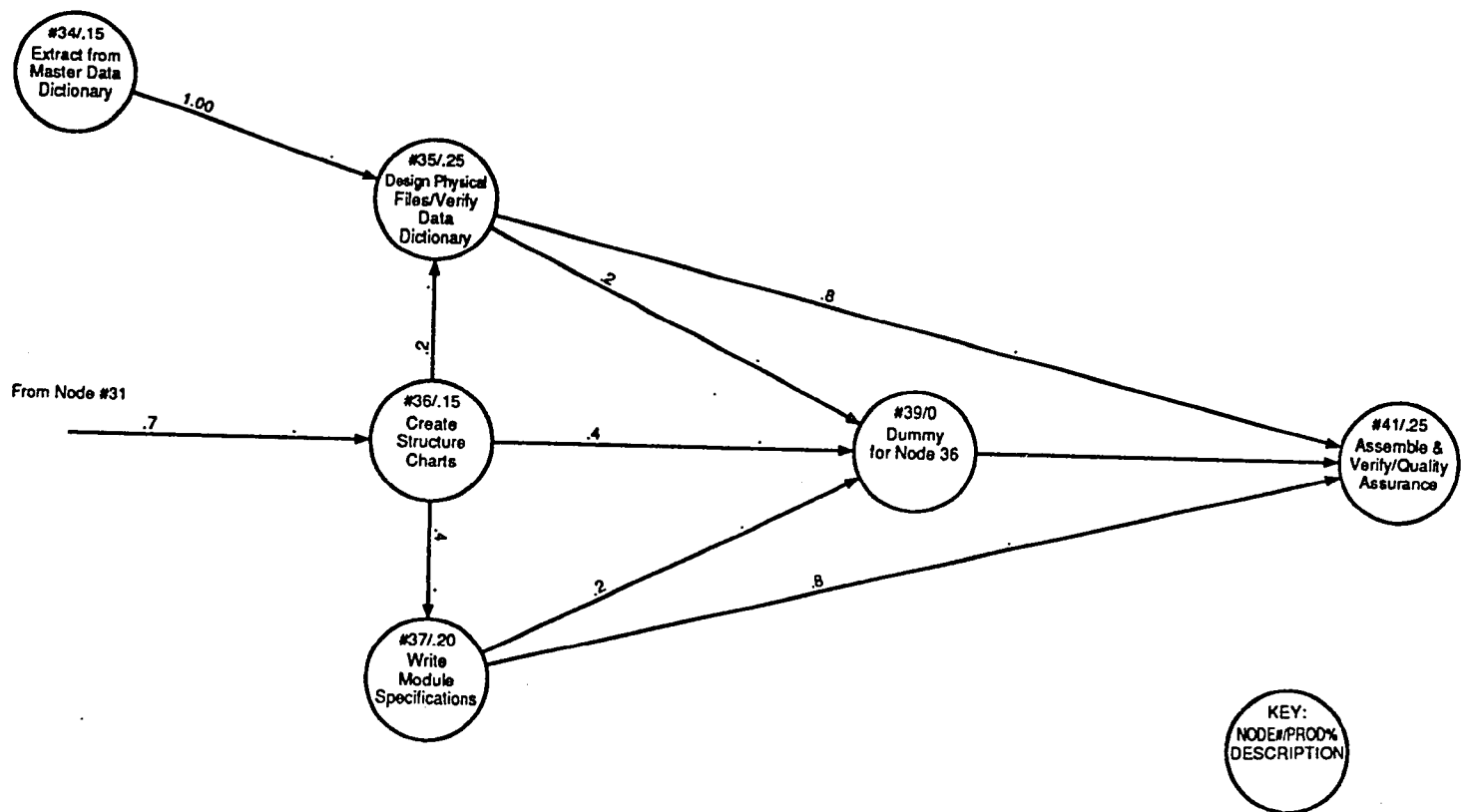


Figure 3-12: Cost Graph for Program Analysis

Figure 4-1: Interview Background Questions

What is the current budgeted staff size?

Number of analysts, programmers.

What is the annual budget of the software development group?

How many projects are commenced (monthly/annually) ?

What is the cost and size of an average (typical) project?

Are the following functions performed as part of the software development effort or by separate units:

user documentation

quality assurance

operations (implementation)

ongoing maintenance (post implementation)

Current SDLC model used, if any?

Current development methodology used, if any?

Current CASE tool usage, if any?

Current development software, hardware used (for development and for target system).

The following questions were asked after preliminary explanation of the cost flow model:

Cost statistics (if any):

collection points in life cycle

granularity of data

objectivity of data

measurement unit (lines of code, hours, dollars)?

Rework:

% of rework, if known

separate cost of rework, if known

measurement points and units, if known

length and measurement of feedback loop: eg, if during coding you have to rework design, how is this costed? As part of coding cost, or added to design cost, or kept as a separate rework cost (etc) ?

Figure 4-2 (continued)

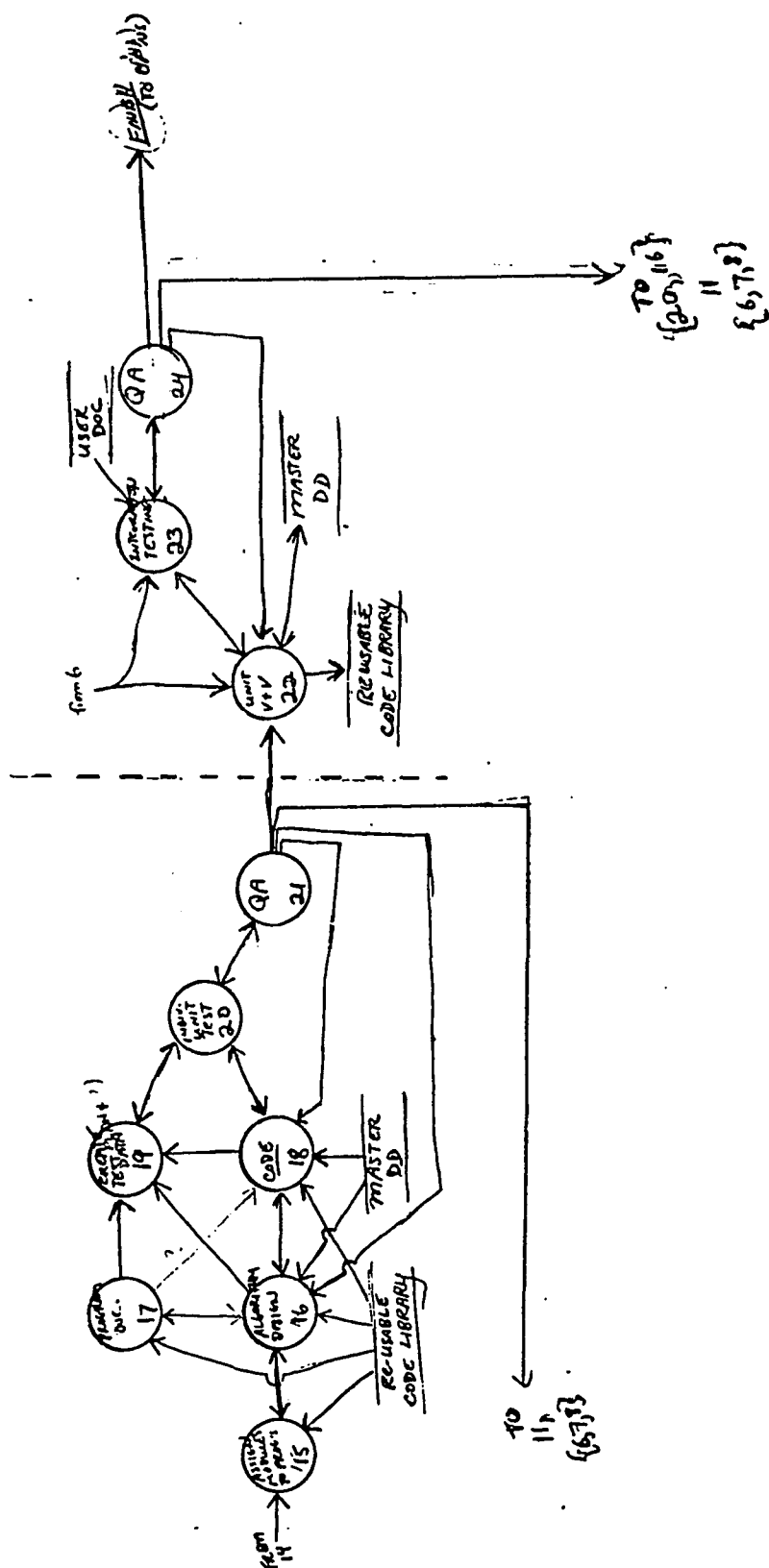


Figure 4-3: Company Data Summary

Company	Govt ?	Average Project Budget	Staff Size	Str.Mth?	CASE?
A	Yes	\$ 750,000 - \$ 1,200,000	220	(No)	No
B	Yes	\$ 750,000 - \$ 2,000,000	1,000	Yes	No
C	No	\$ 200,000 - \$ 400,000	150	Yes	Yes
D	No	\$ 100,000 - \$ 250,000	20	Yes	No
E	No	\$ 250,000 +	60	Yes	No

Figure 4-4: Company by Cost Graph Group

Group #	Company	Description
1	A	No structured methodologies, No CASE use
2	B, D, E	Use structured methods, No CASE use
3	C	Use structured methods, use CASE tools

Figure 4-5: Cost Graph for Group 3
Structured Methodologies, CASE Use

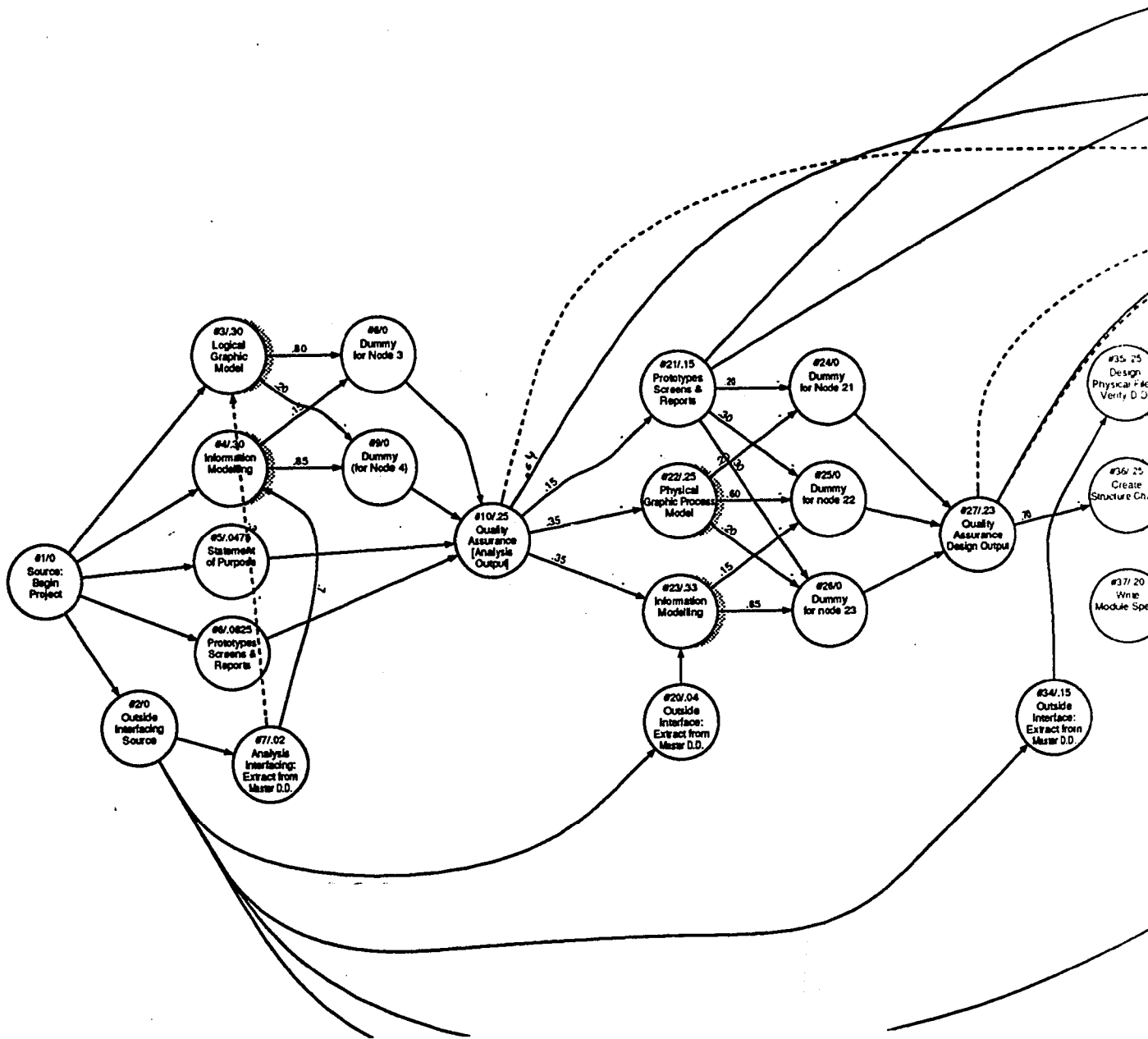
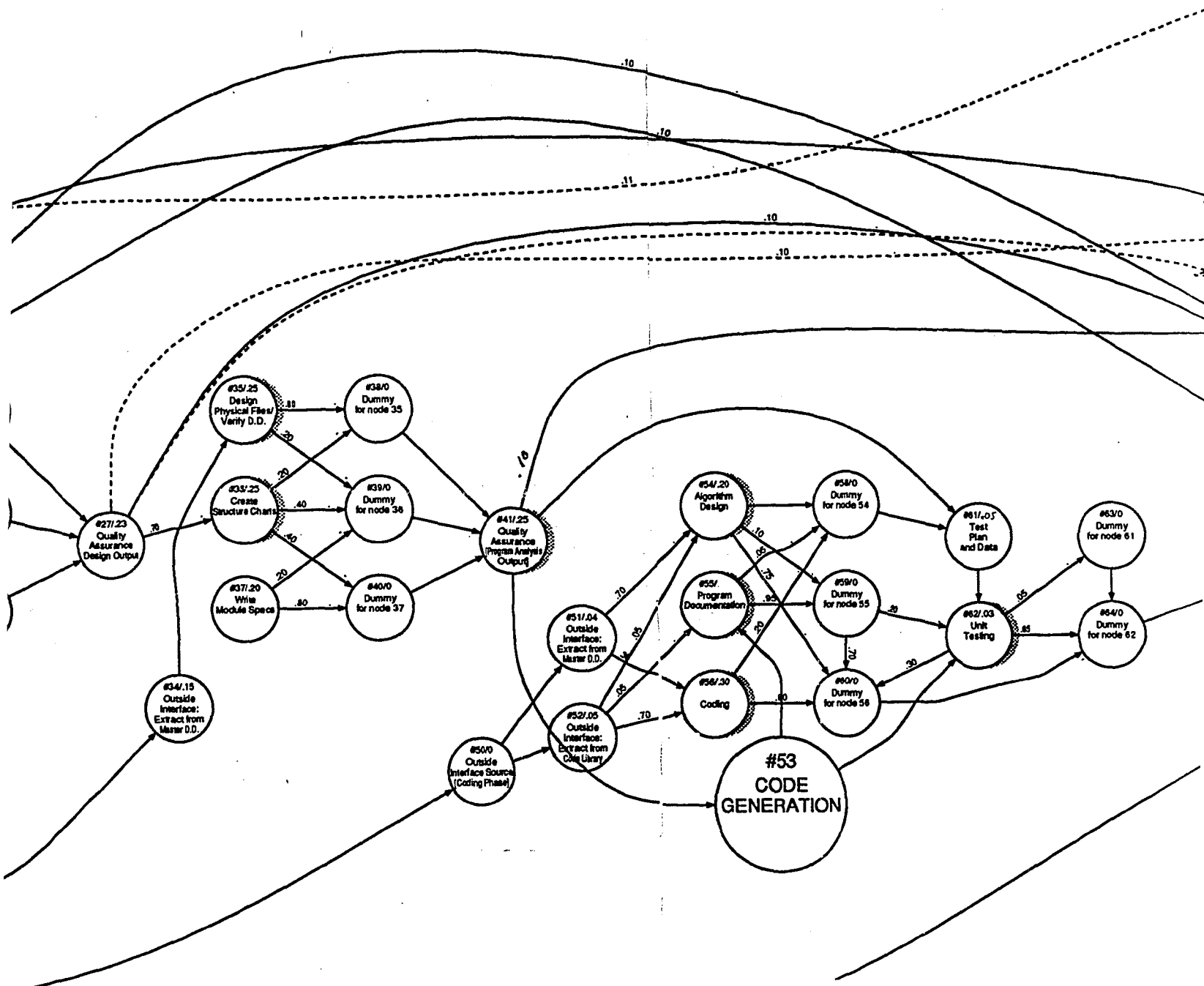


**Figure 4-5: Group 3
Structured Methodologies, CASE Use**



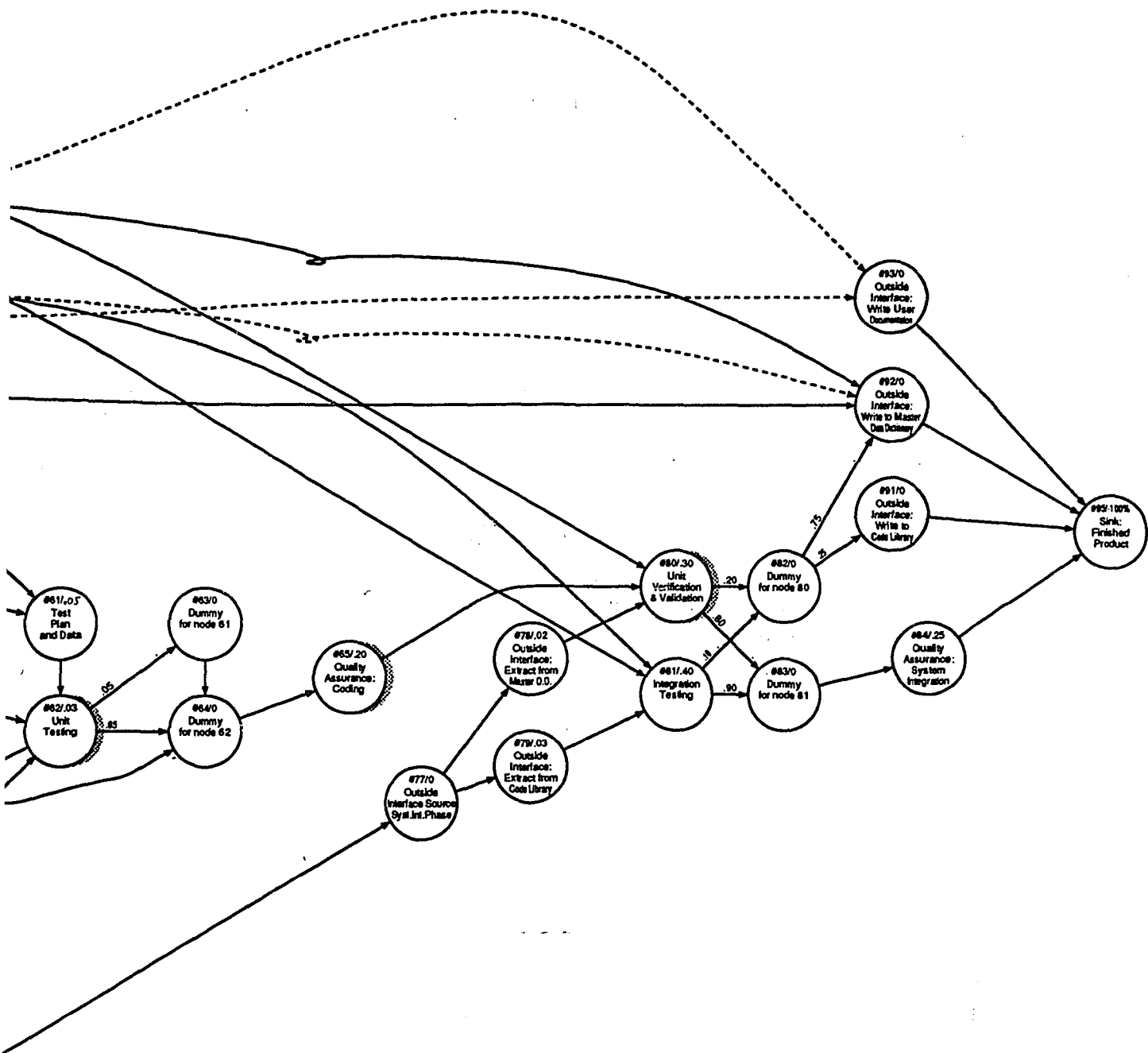


Figure 4-6: Cost Graph for Group 2
Structured Methodologies, No CASE Use

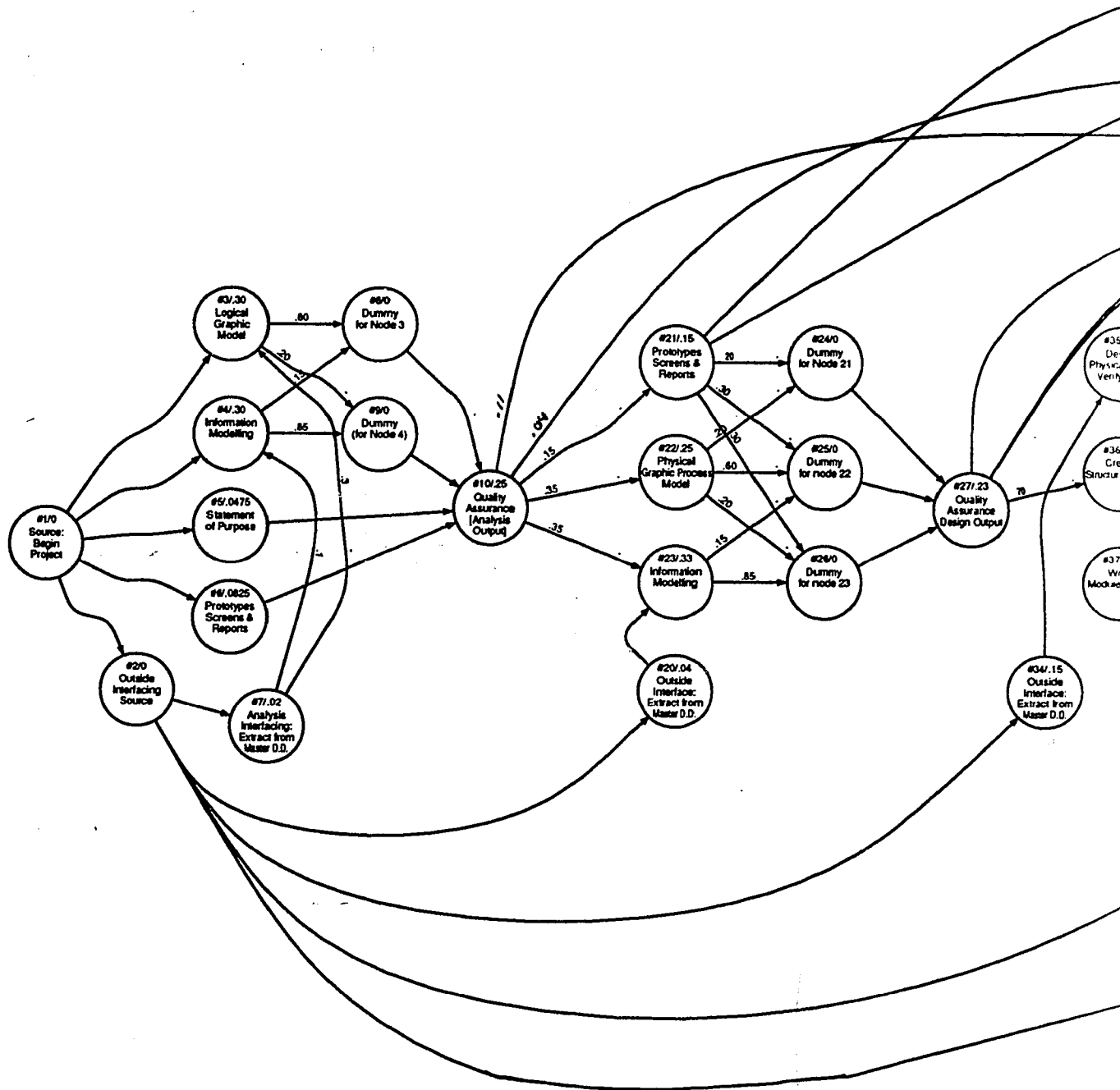
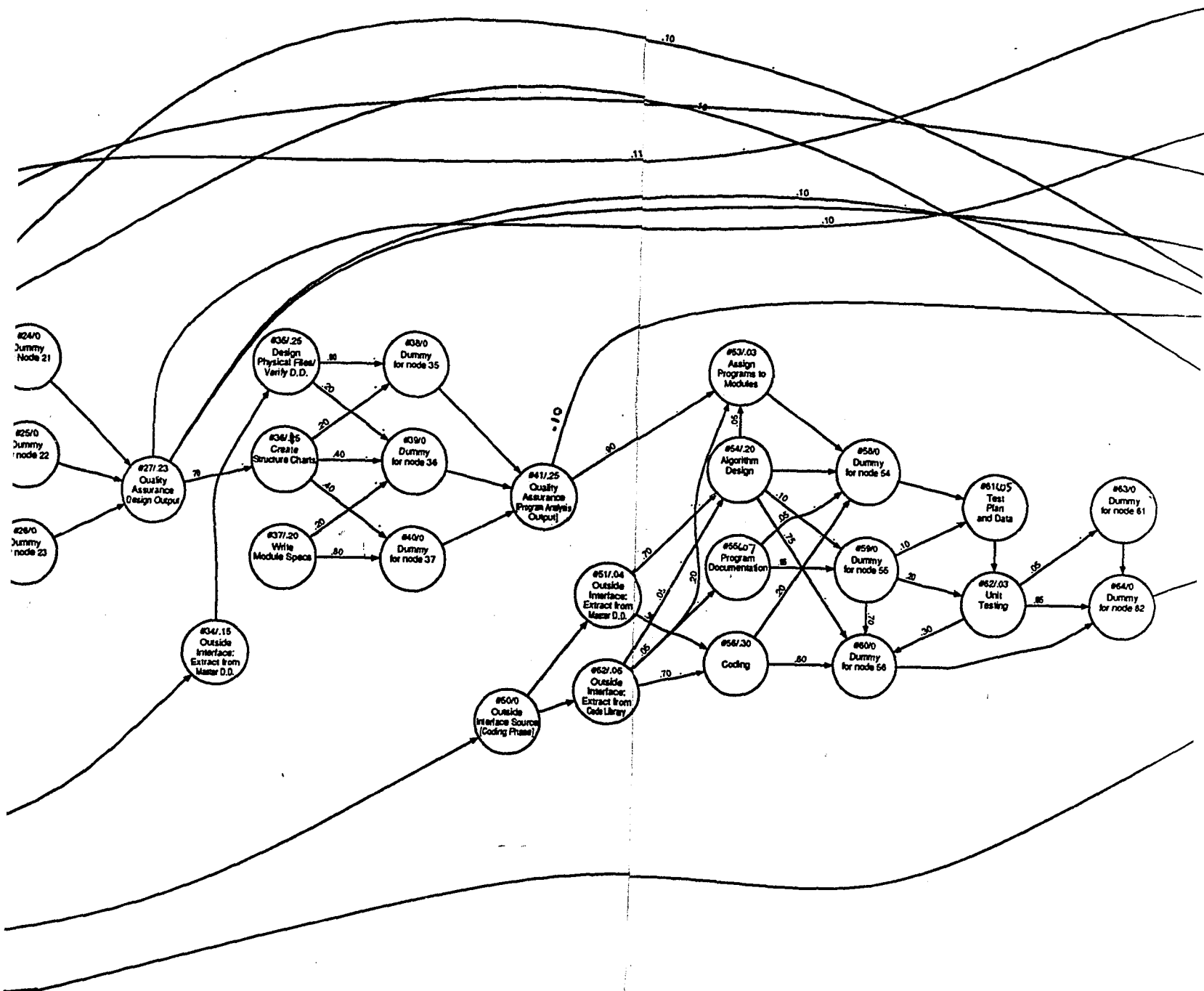


Figure 4-6: Group 2
Structured Methodologies, No CASE Use



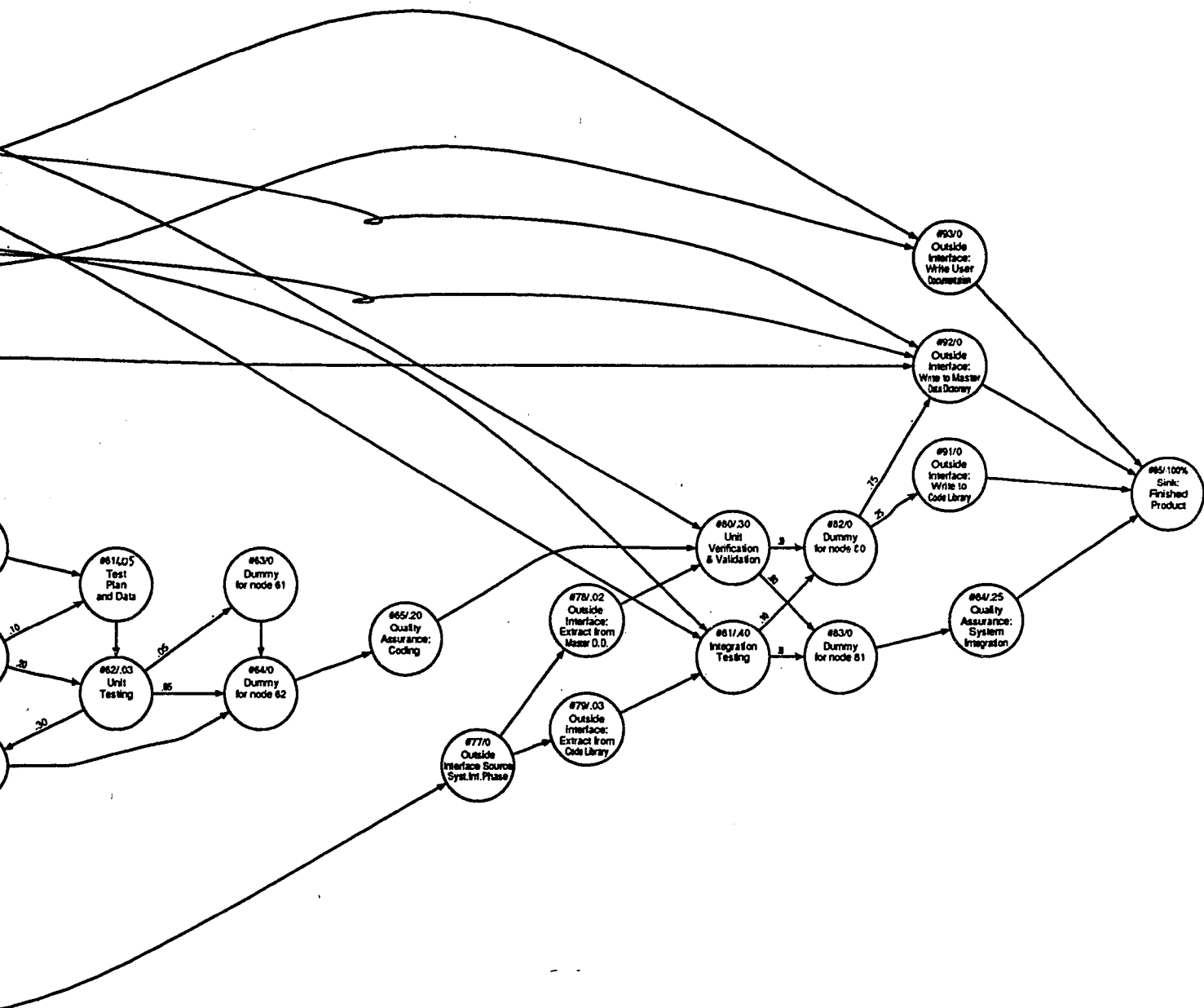


Figure 4-7: Cost Graph for Group 1
No Structured Methodologies, No CASE Use

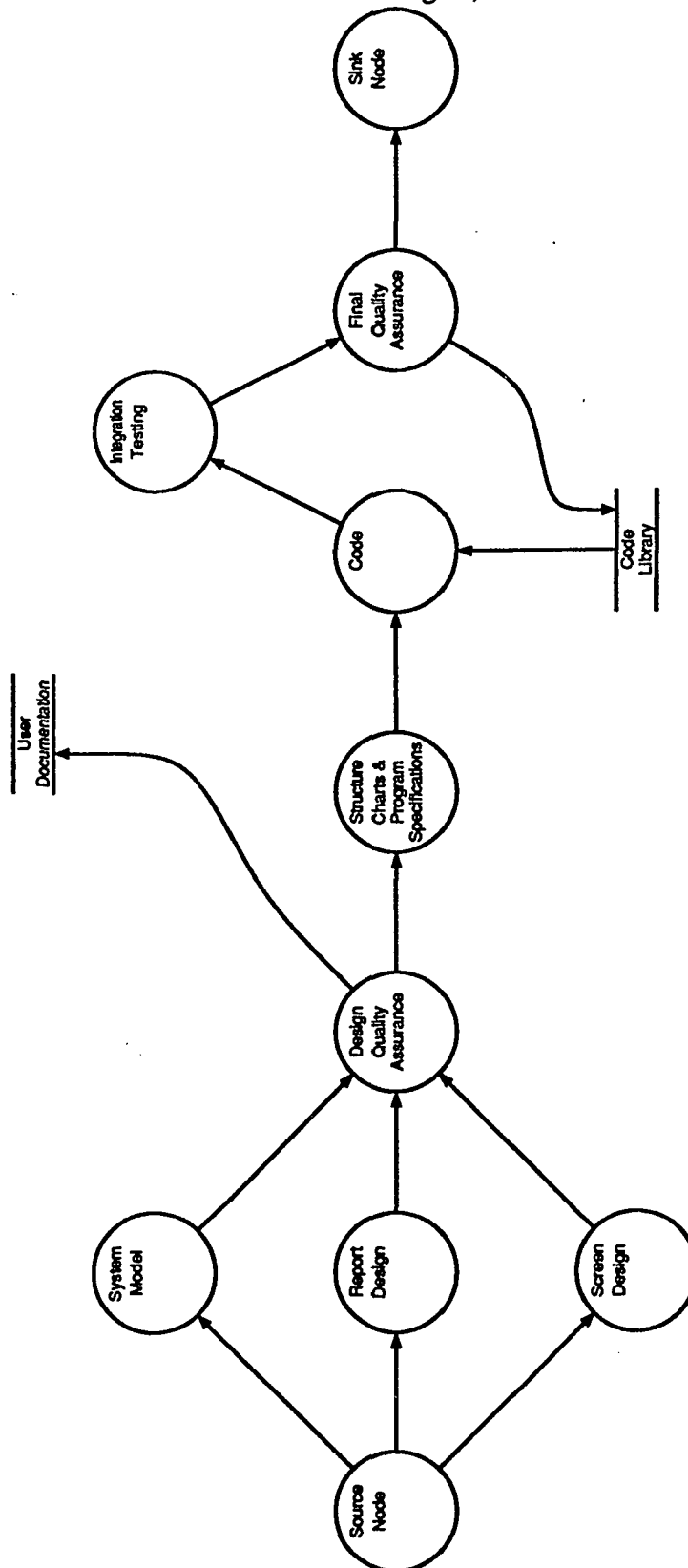


Figure 4-8: SDLC Overview

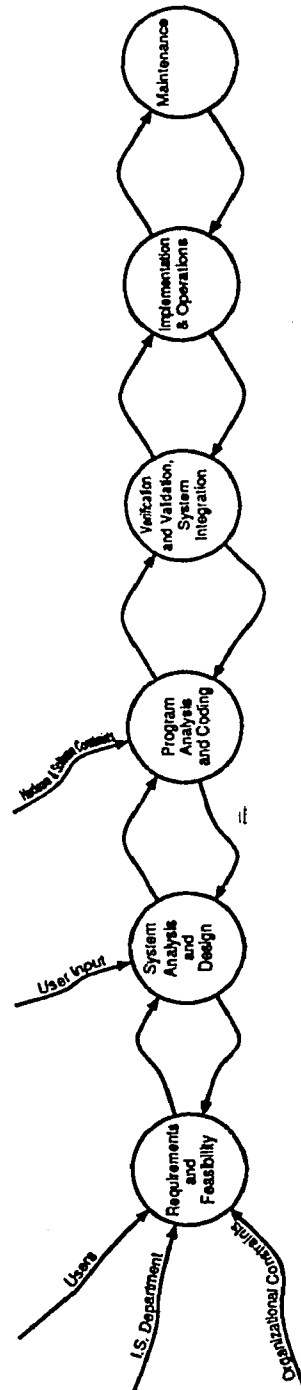


Figure 4-9: One Way to Graph Rework Without Cycles

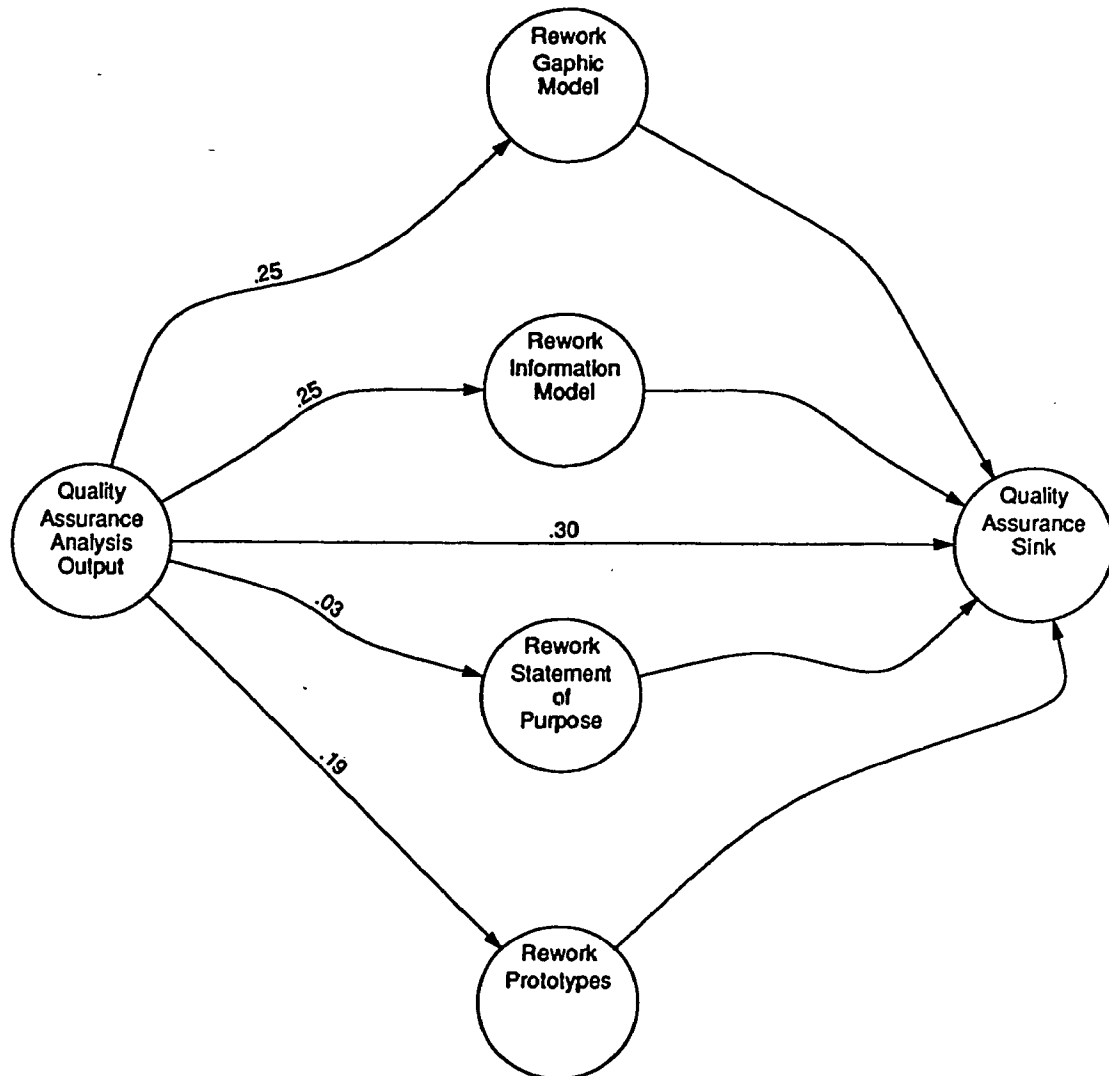
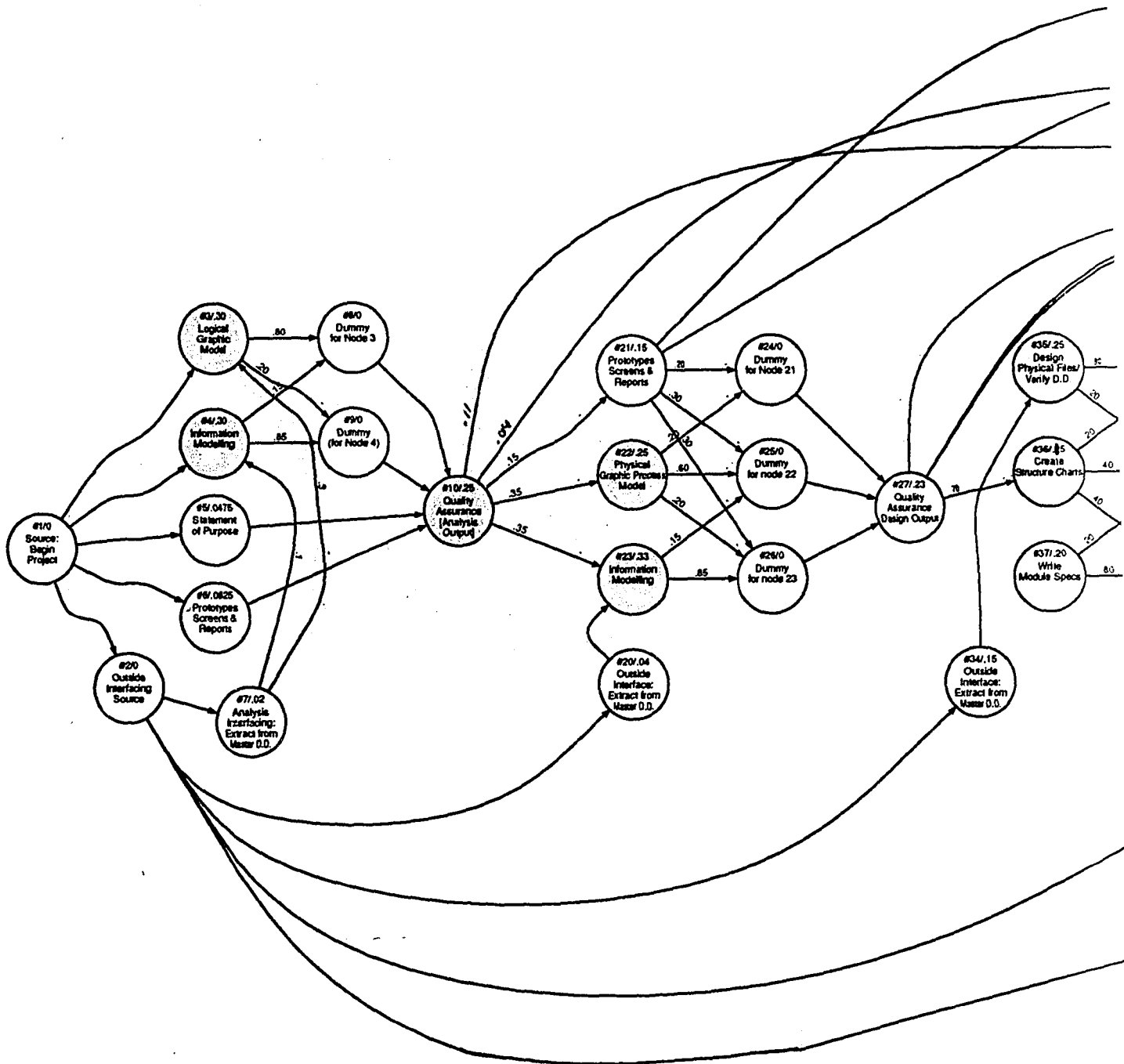
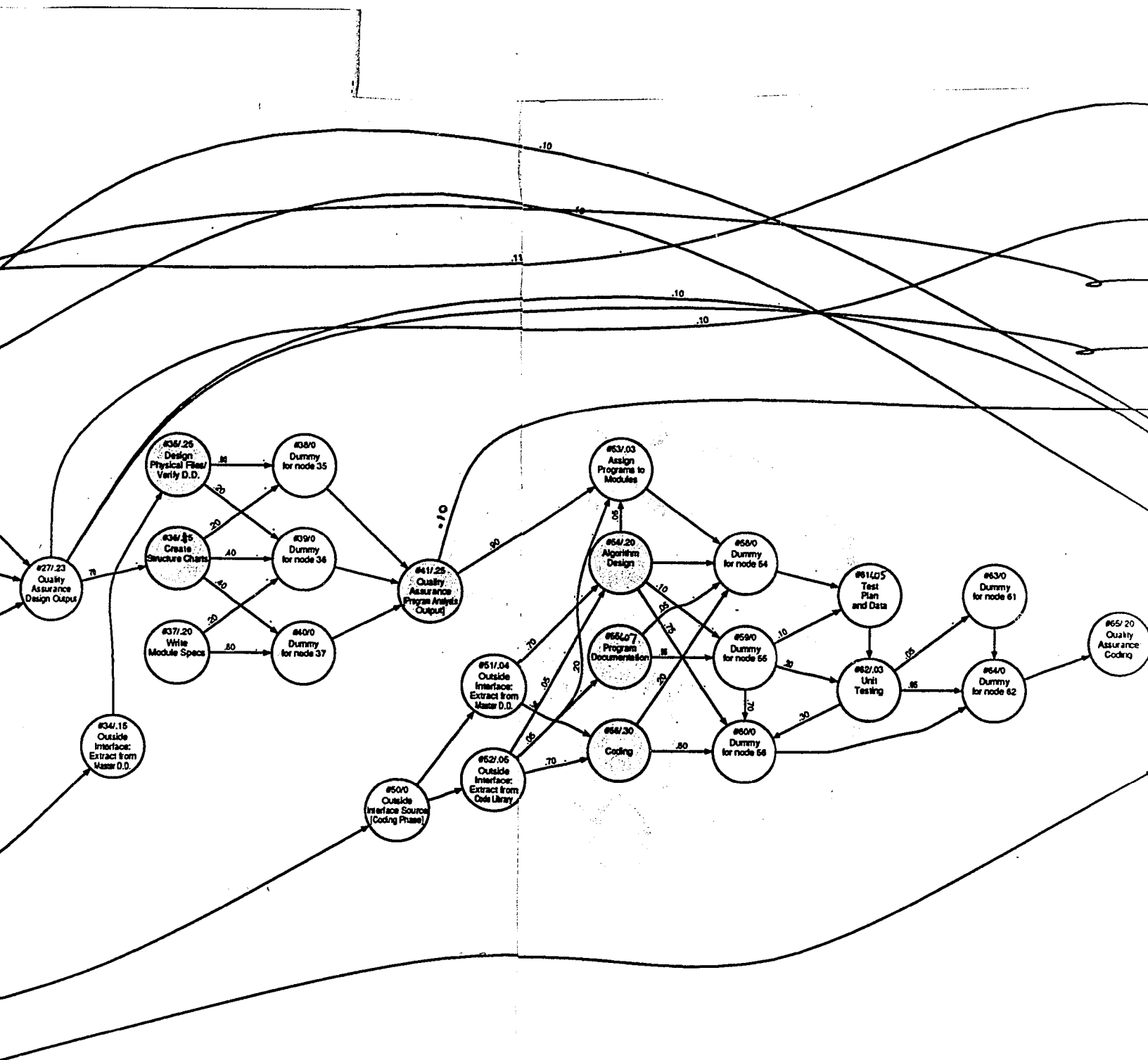


Figure 4-10: Group 3 with Changes from Group 2 Highlighted





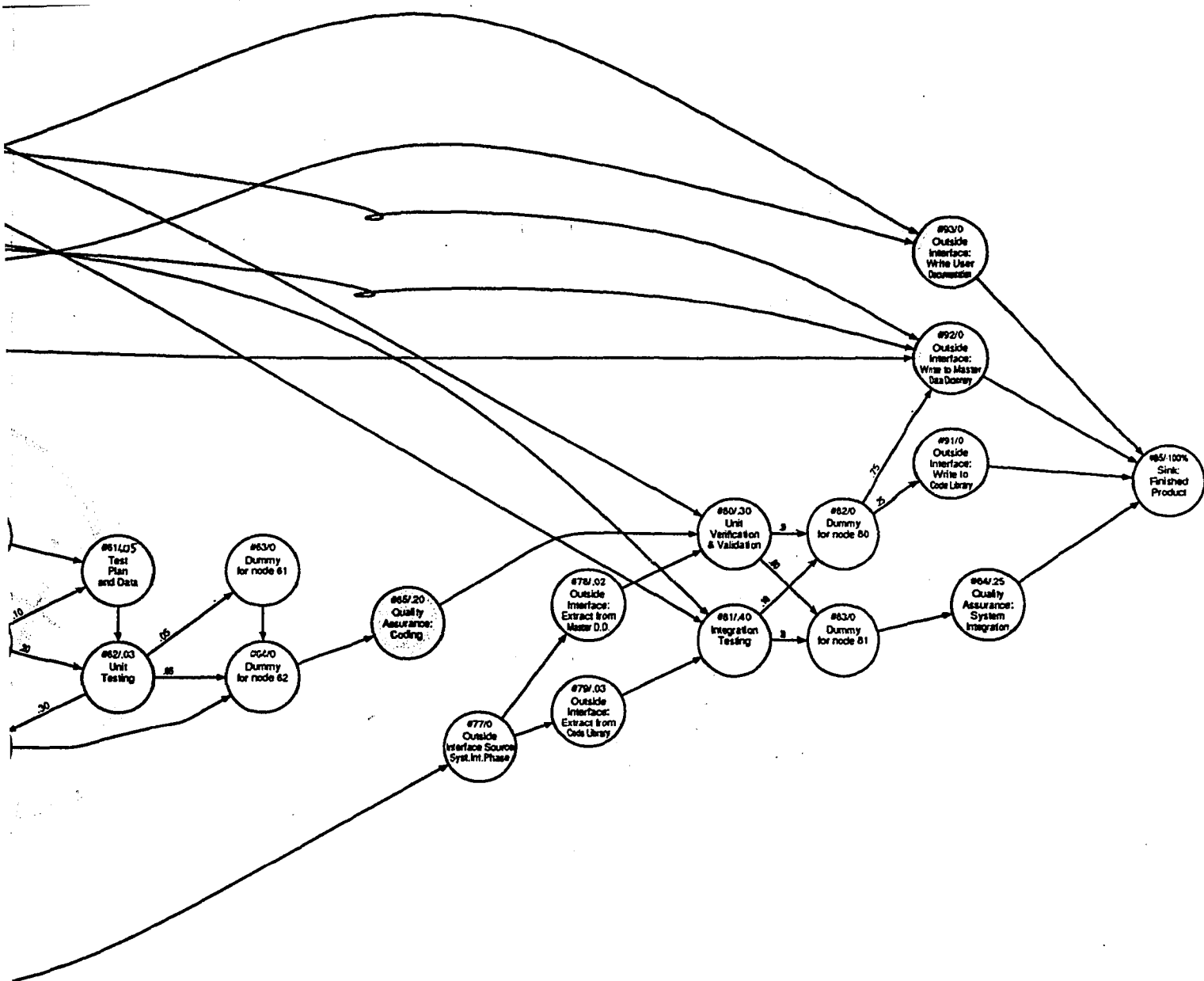


Figure 4-11: Spreadsheet Dollar Assumptions

System Development Process Model		
Type 2: Structured Methodology, Pre-CASE		
Standard % assumptions (Ref: Boehm 1981)		
	Percent	\$ Cost
TOTAL PROJECT COST:	100%	\$500, 000.00
SYSTEM ANALYSIS AND DESIGN	65%	\$325, 000.00
Logical Modelling	60%	\$195, 000.00
Physical Modelling	40%	\$130, 000.00
IMPLEMENTATION	15%	\$75, 000.00
Program Analysis	25%	\$18, 750.00
Coding	75%	\$56, 250.00
INTEGRATION, V&V	20%	\$100, 000.00
Below are added from graph, across all phases		
Outside Interfacing		
ProdCost of Reading	4.4%	\$21, 975.00
Value of writing	35%	\$174, 028.50

Figure 5-1: Fleckenstein Phase Cost Assumptions

System Development Process Model		
Type 2: Structured Methodology, Pre-CASE		
PHASE % FROM FLECKENSTEIN (1983)		
	Percent	\$ Cost
TOTAL PROJECT COST:	100%	\$500, 000.00
SYSTEM ANALYSIS AND DESIGN	40%	\$200, 000.00
Logical Modelling	60%	\$120, 000.00
Physical Modelling	40%	\$80, 000.00
IMPLEMENTATION	10%	\$50, 000.00
Program Analysis	25%	\$12, 500.00
Coding	75%	\$37, 500.00
INTEGRATION, V&V	50%	\$250, 000.00
Below are added from graph, across all phases		
Outside Interfacing		
ProdCost of Reading	4.7%	\$23, 350.00
Value of writing	26%	\$128, 106.00

Figure 5-3: Interviewee Cost Assumptions

System Development Process Model		
Type 2: Structured Methodology, Pre-CASE		
Interviewee suggested % assumptions		
	Percent	\$ Cost
TOTAL PROJECT COST:	100%	\$500,000.00
SYSTEM ANALYSIS AND DESIGN	40%	\$200,000.00
Logical Modelling	60%	\$120,000.00
Physical Modelling	40%	\$80,000.00
IMPLEMENTATION	40%	\$200,000.00
Program Analysis	25%	\$50,000.00
Coding	75%	\$150,000.00
INTEGRATION, V&V	20%	\$100,000.00
Below are added from graph, across all phases		
Outside Interfacing		
ProdCost of Reading	6.3%	\$31,600.00
Value of writing	29%	\$145,056.00

N#	Pcost%	Prodcost	in1	in2	in3	in4	in5	Incst	Outcost	out1	out1%	out1\$	out2	out2%	out2\$	out3	out3%	out3\$	out4	out4%	out4\$	out5	out5%	out5\$
1																								
2																								
3	30.0%	\$36,000	1	7				\$720	\$36,720	8	80.0%	\$29,376	9	20.0%	\$7,344									
4	30.0%	\$36,000	1	7				\$1,680	\$37,680	8	15.0%	\$5,652	9	85.0%	\$32,028									
5	4.8%	\$5,700	1					\$0	\$5,700	10	100.0%	\$5,700												
6	8.3%	\$9,900	1					\$0	\$9,900	10	100.0%	\$9,900												
7	2.0%	\$2,400	1					\$0	\$2,400	3	30.0%	\$720	4	70.0%	\$1,680									
8			3	4				\$35,028	\$35,028	10	100.0%	\$35,028												
9			3	4				\$39,372	\$39,372	10	100.0%	\$39,372												
10	25.0%	\$30,000	8	9	5	6		\$90,000	\$120,000	92	4.0%	\$4,800	93	11.0%	\$13,200	21	15.0%	\$18,000	22	35.0%	\$42,000	23	35.0%	\$42,000
11																								
12																								
13																								
14																								
15																								
16																								
17																								
18																								
19																								
20	4.0%	\$3,200	2					\$0	\$3,200	23	100.0%	\$3,200												
21	15.0%	\$12,000	10					\$18,000	\$30,000	80	10.0%	\$3,000	81	10.0%	\$3,000	24	20.0%	\$6,000	25	30.0%	\$9,000	26	30.0%	\$9,000
22	25.0%	\$20,000	10					\$42,000	\$62,000	24	20.0%	\$12,400	25	60.0%	\$37,200	26	20.0%	\$12,400						
23	33.0%	\$26,400	10	20				\$45,200	\$71,600	25	15.0%	\$10,740	26	85.0%	\$60,860									
24			21	22				\$18,400	\$18,400	27	100.0%	\$18,400												
25			21	22	23			\$56,940	\$56,940	27	100.0%	\$56,940												
26			21	22	23			\$82,260	\$82,260	27	100.0%	\$82,260												
27	23.0%	\$18,400	24	25	26			\$157,600	\$176,000	92	10.0%	\$17,600	93	10.0%	\$17,600	81	10.0%	\$17,600	36	70.0%	\$123,200			
28																								
29																								
30																								
31																								
32																								
33																								
34	15.0%	\$7,500	2					\$0	\$7,500	35	100.0%	\$7,500												
35	25.0%	\$12,500	34					\$7,500	\$20,000	38	80.0%	\$16,000	39	20.0%	\$4,000									
36	15.0%	\$7,500	27					\$123,200	\$130,700	38	20.0%	\$26,140	39	40.0%	\$52,280	40	40.0%	\$52,280						
37	20.0%	\$10,000						\$0	\$10,000	39	20.0%	\$2,000	40	80.0%	\$8,000									
38			35	36				\$42,140	\$42,140	41	100.0%	\$42,140												
39			35	36	37			\$58,280	\$58,280	41	100.0%	\$58,280												
40			36	37				\$60,280	\$60,280	41	100.0%	\$60,280												
41	25.0%	\$12,500	38	39	40			\$160,700	\$173,200	92	10.0%	\$17,320	53	90.0%	\$155,880									
42																								
43																								
44																								
45																								
46																								
47																								
48																								

Figure 5-4: Graph Printout Using Interviewee Assumptions

References

- [1]Thomson, G. *Capitalism and After: The Rise and Fall of Commodity Production* (London: China Policy Study Group, 1973).
- [2]Marx, K. & Engels, F. *The Origin of the Family, Private Property, and the State* (New York: International Publishers, 1972).
- [3]Shaiken, H.: *Work Transformed*, (New York: Holt Rinehart & Winston, 1986).
- [4]Mowshowitz, A. *Information-Commodities and Organizational Flexibility*, unpublished proposal to the National Science Foundation, January 1989.
- [5]Bell, D. "The Social Framework of the Information Society" in Dertouzos, M. & Moses, J. *The Computer Age: A Twenty-Year View* (Cambridge, MA: MIT Press, 1983).
- [6]Cherry, C. in: *The Age of Access: Information Technology and Social Revolution*, W. Edmonson, editor. (London: Longwood Publishing, 1985).
- [7]Cleveland, H. "Information as a Resource", *The Futurist*, 1982, pp.34-39.
- [8]Mowshowitz, A. *The Information Marketplace*, Report to the Office of Technology Assessment, August 1987.
- [9]Fronzizi, R. *What is Value? An Introduction to Axiology* 2nd. ed. (LaSalle, IL: Open Court, 1971).
- [10]Black, S. H. & Marchand, D. A. "Assessing the Value of Information in Organizations: A Challenge for the 1980s", *The Information Society Journal*, 1(3), pp. 191-225.
- [11]Griffiths, J. "The Value of Information and Related Systems, Products, and Services", in Williams, M. W. (ed.) *Annual Review of Information Science and Technology Volume 17* (White Plains, NY: Knowledge Industry Publications, 1982), pp. 269-284.
- [12]Tate, P. "A Badly Needed Sense of the Value of Information", *Datamation*, March 15, 1989, page 78.
- [13]Black, S. H. & Marchand, D. A.: "Assessing the Value of Information in Organizations: A Challenge for the 1980s", *The Information Society Journal*, 1:3 (1982), pp. 191-225.
- [14]Hershleifer, J.: "Where Are We in the Theory of Information?", *American Economic Review*, 63:2 (1973), pp. 31-39.
- [15]Burch, J. "In a Competitive World, The Strongest Weapon is Information", *Data Management*, February 1986, pp. 22-28. This article is just one example of hundreds devoted to this point of view.

[16]Yovits, M. & Foulk, C. "Experiments and Analysis of Information Use and Value in a Decision-Making Context", *Journal of the American Society for Information Science*, 36(2) 1985, pp. 63-81.

[17]Er, M. "A Critical Review of the Literature on the Organizational Impact of Information Technology", *IEEE Technology and Society Magazine*, June 1989, pp. 17-23.

[18]Robey, D. "Computer Information Systems and Organization Structure", *Communications of the ACM*, 24(10), 1981, pp. 679-687.

[19]Haney, W.V. "Serial Communication of Information in Organizations", *Concepts and Issues in Administrative Behavior*, Marlick, S. & VanNess, E. ed. (Englewood Cliffs, NJ: Prentice Hall, 1962), pp.150-164.

[20]Koehler, J., Anatol, K., & Applebaum, R. *Organizational Communication: Behavioral Perspectives*, (New York: Holt Rinehart & Winston, 1976).

[21]McClure, C. "The Information Rich Employee and Information for Decision Making: Review and Comments", *Information Processing & Management*, V. 14 (1978), pp. 381-394.

[22]Taylor, R. "Value-Added Processes in the Information Life Cycle", *Journal of the American Society for Information Science*, September 1982, pp. 341-346.

[23]Taylor, R. *Value-Added Processes in Information Systems*, (Norwood, NJ: Ablex, 1986).

[24]Gotlieb, C. *The Economics of Computers: Costs, Benefits, Policies, and Strategies* (Englewood Cliffs, NJ: Prentice Hall, 1985).

[25]Machlup, F. *The Production and Distribution of Knowledge in the United States* (Princeton, NJ: Princeton University Press, 1962).

[26]Porat, M. *The Information Economy: Definition and Measurement*, US Department of Commerce, Office of Telecommunications Publication 77-12(1), May 1977.

[27]OECD Organization for Economic Cooperation and Development: *Information Activities, Electronics and Telecommunications Technologies: Impact on Employment, Growth and Trade*, 1981.

[28]Information Industry Association: "The Outlook for the Information Industry", *The Business of Information 1983: Volume 2 - The Outlook for the Information Industry* (Washington, DC: Information Industry Association, 1984).

[29]de Sola Pool, I. "Tracking the Flow of Information", *Science*, 221(4611) 1983, pp. 609-613.

[30]Boehm, B. *Software Engineering Economics*, (1981: Prentice Hall). (Englewood Cliffs, NJ: Prentice Hall, 1981).

- [31]Pressman, R. *Making Software Engineering Happen: A Guide for Instituting the Technology* (Englewood Cliffs, NJ: Prentice Hall, 1988), page 4.
- [32]Page-Jones, M. *Practical Project Management: Restoring Quality to DP Projects and Systems* (New York: Dorset House, 1985).
- [33]Exponents of this view can be found in numerous articles in the computer trade press as well as in periodicals such as *Time* or *Newsweek*. The effects of this view are combatted forcefully in Deming, W. *Out of the Crisis* (Cambridge, MA: MIT Center for Advanced Engineering Study, 1986).
- [34]Zuboff, S. *In the Age of the Smart Machine* (New York: Basic Books, 1988).
- [35]Shaiken, H. *op. cit.*
- [36]Garson, B. *Electronic Sweatshop* (New York: Penguin, 1990).
- [37]Howard, R. *Brave New Workplace* (1985: Viking).
- [38]Cooley, M. *Architect or Bee?* (Boston, MA: South End Press, 1980).
- [39]Hirschhorn, L. *Beyond Mechanization* (Cambridge, MA: MIT Press, 1984).
- [40]See the works cited above by Garson and Howard. More recent study on electronic monitoring has been performed by trade unions, notably by the 9-to-5 National Organization of Working Women, for example in *Hidden Victims: Clerical Workers, Automation, and the Changing Economy* (1985), *Computer Monitoring and Other Dirty Tricks* (1986), and *Stories of Mistrust and Manipulation* (1990).
- [41]Boehm, B. *op. cit.*
- [42]Gotlieb, C. *The Economics of Computers: Costs, Benefits, Policies, and Strategies* (Englewood Cliffs, NJ: Prentice Hall, 1985).
- [43]Phister, M. *Data Processing Technology and Economics, 2nd ed.* (Boston, MA: Digital Press, 1979).
- [44]Gotlieb, C. *op. cit.*, page 132.
- [45]Gotlieb, C. *op. cit.*, page 139.
- [46]Greenbaum, J. *In the Name of Efficiency: Management Theory and Shopfloor Practice in Data-Processing Work* (Philadelphia: Temple University Press, 1979).
- [47]Greenbaum, J. *op. cit.*, page 54.
- [48] Whitten, J., Bentley, L. and Barlow, V. *Systems Analysis & Design Methods*, 2nd. ed. (Homewood, IL: Irwin, 1989), page 59.

- [49] Pressman, R. *Making Software Engineering Happen: A Guide for Instituting the Technology* (Englewood Cliffs, NJ: Prentice Hall, 1988).
- [50] *Ibid*, pp. 4-5.
- [51] Mills, H., Linger, R. & Hevner, A. *Principles of Information Systems Analysis and Design* (New York: Academic Press, 1986), page 35.
- [52] Humphrey, W. & Kellner, M. "Software process modeling: principles of entity process models," *Proceedings of the 11th International Conference on Software Engineering*, IEEE, 1989.
- [53] Dolatta, T. *Data Processing in the 1980's* (New York: Wiley, 1976).
- [54] Smith, A. Jr. "Making A Case For CASE", *Business and Technology Consultant*, V. 2, N. 6, June 1990, pp. 27-27.
- [55] Humphrey, W. *Managing the Software Process* (Reading, MA: Addison Wesley, 1989), page 254.
- [56] McCracken, D. & Jackson, M. "Life Cycle Concept Considered Harmful", *ACM SIGSOFT Software Engineering Notes*, V. 7, N. 2, 1982.
- [57] Pressman, R. *op. cit.*, page 7.
- [58] Pressman, R. *op. cit.*, page 10.
- [59] StarSys, Inc. *MacBubbles*, Support Disk Version 1.8.
- [60] Grady, R. & Caswell, D. *Software Metrics: Establishing A Company-Wide Program* (Englewood Cliffs, NJ: Prentice Hall, 1987), page 195.
- [61] Gilb, T. *Principles of Software Engineering Management* (Reading, MA: Addison Wesley, 1988), page 84.
- [62] *Ibid*, page 94.
- [63] Boehm, B. "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, May 1988, pp. 61-72.
- [64] Mills, H. et. al., *op. cit.*, page 32.
- [65] *Ibid*, page 33.
- [66] Humphrey, W. *op. cit.*
- [67] *Ibid*, page 259.
- [68] *Ibid*, page 260.

- [69] *Ibid*, page 277.
- [70] *Ibid*, page 14.
- [71] Boehm, B. *Software Engineering Economics* (Englewood Cliffs, NJ: Prentice Hall, 1981).
- [72] McCabe, T. "A Complexity Measure", *IEEE Transactions on Software Engineering*, December 1976, pp. 308-320. Focuses on complexity as a reliability and cost measure.
- [73] Halstead, M. *Elements of Software Science* (New York: Elsevier, 1976).
- [74] Putnam, L. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", *IEEE Transactions on Software Engineering*, July 1978, pp. 345-361.
- [75] Boehm, B. *op. cit.*, p. 118.
- [76] Boehm, B. *Op. cit.*, page 84.
- [77] Boehm, B. *Op. cit.*, page 138.
- [78] Boehm, B. *Op. cit.*, page 495.
- [79] Details of the COCOMO data base are discussed in detail in Chapter 29 of Boehm's book.
- [80] Chmura, L, Norcio, A. and Wicinski, T. "Evaluating Software Design Processes by Analyzing Change Data Over Time", *IEEE Transactions on Software Engineering*, V. 16, N. 7, July 1990, pp. 729-739, citing V. Church, D. Card, F. McGarry and V. Basili, "Guide to data collection," NASA/Goddard SEL, Greenbelt, MD, Tech. Rep. SEL-81-001, 1981.
- [81] Gilb, T. *op. cit.*, Chapter 16.
- [82] Bryce, M. & Associates Inc., "What's In A Word?", *Management Visions*, V. 6, N.2, April 1990, pp. 1-3.
- [83] Williamson, M. "Editorial", *CASE Strategies*, V. 1, N. 1, June 1989, page 2.
- [84] Datamation interview, "The Approach is Realistic", *Datamation*, March 15, 1990, pp. 55-58.
- [85] Gibson, M. "The CASE Philosophy", *Byte*, April 1988, pp.209-218. This article appears to be the first which divided CASE tools into the upper, middle, and lower classifications.
- [86] Babcock, C. "Applications Backlogs", *Computerworld*, December 14, 1987.
- [87] Snyders, J. "A CASE of Unknown Identity", *InfoSystems*, October 1987, pp. 54-59.

[88] Grochow, J., Vice President of American Management Systems, as quoted in *ibid.*

[89] *Software Magazine*, December 1989.

[90] Hanson, S. and Rosinski, R. "Programmer Perceptions of Productivity and Programming Tools", *CACM*, V. 28, N. 2, February 1985, pp. 180-189.

[91] Jones, T. "Measuring Programming Quality and Productivity", *IBM Systems Journal*, V. 17, N. 1, 1978, pp. 39-63.

[92] Thadhani, A. "Factors Affecting Programmer Productivity During Application Development", *IBM Systems Journal*, V. 23, N. 1, 1984, pp. 19-35.

[93] Jones, C. *Programmer Productivity* (New York: McGraw Hill, 1986).

[94] For example, Perrone, G. "Low-cost CASE: tomorrow's promise emerging today", *IEEE Computer*, November 1987, pp. 104-110, and Martin, J. series in *PC Week*, Winter through Spring 1990.

[95] Norman, R. *Integrated Development Environments in Support of Information Systems Design Methodologies and Systems Analysts' Productivity*, PhD Thesis, University of Arizona, 1987.

[96] Norman, R. & Nunamaker, J. "CASE Productivity Perceptions of Software Engineering Professionals", *CACM*, V. 32, N. 9, 1989.

[97] Statland, N. "Payoffs Down the Pike: A CASE Study", *Datamation*, April 1, 1989, pp. 32-56.

[98] Smith, D. "CASE in Context", *SEI Bridge*, March 1990, pp. 4-5.

[99] Norman, R. and Nunamaker, J. *op. cit.*, page 1102, emphasis added.

[100] Statland, N. *op. cit.*

[101] Gotlieb, C. *The Economics of Computers: Costs, Benefits, Policies, and Strategies* (Englewood Cliffs, NJ: Prentice Hall, 1985). See especially pages 169-180.

[102] Boehm, B. *op. cit.*

[103] Mowshowitz, A. *Inside Information: Computers in Fiction* (Reading, MA: Addison-Wesley, 1977).

[104] Lawler, E. *Combinatorial Optimization: Networks and Matroids* (New York: Holt Rhinehart & Winston, 1976).

[105] Mowshowitz, A. *The Conquest of Will: Information Processing in Human Affairs* (Reading, MA: Addison Wesley, 1976), page 211.

[106] Lawler, E. *op. cit.*

- [107] Elmaghraby, S. *Activity Networks: Project Planning and Control by Network Models* (New York: Wiley, 1977).
- [108] Elmaghraby, S. *ibid.*
- [109] Burrill, C. & Ellsworth, L. *Modern Project Management: Foundations for Quality and Productivity* (Tenafly, NJ: BEA Inc., 1980), Chapter 15.
- [110] Fulkerson, D. R. "A Network Flow Computation for Project Cost Curve", *Management Science* (7)2 1961, pp. 167-178.
- [111] Elmaghraby, S. *op. cit.*
- [112] Mowshowitz, A. "On the Market Value of Information Commodities", report to the National Science Foundation, 1990.
- [113] Alreck, P. & Settle, R. *The Survey Research Handbook* (Homewood, IL: Irwin, 1985).
- [114] Alreck, P. & Settle, R. *op. cit.*, p. 42.
- [115] Humphrey, W. *Managing the Software Process* (Reading, MA: Addison Wesley, 1989).
- [116] Humphrey, W. *op. cit.*, page 252.
- [117] Humphrey, W. *op. cit.*
- [118] Bellin, D. & Suchman, S. *op. cit.*
- [119] Davis, W. *Systems Analysis and Design: A Structured Approach* (Reading, MA: Addison Wesley 1983).
- [120] DeMarco, T. *Structured Analysis and System Specification* (Englewood Cliffs, NJ: Yourdon-Prentice Hall, 1979).
- [121] McMenamin, S. & Palmer, J. *Essential Systems Analysis* (Englewood Cliffs, NJ: Yourdon-Prentice Hall, 1984).
- [122] Page-Jones, M. *The Practical Guide to Structured Systems Design, 2nd. ed.* (Englewood Cliffs, NJ: Yourdon-Prentice Hall, 1988).
- [123] Ward, P. *System Development Without Pain* (Englewood Cliffs, NJ: Yourdon-Prentice Hall, 1984).
- [124] Whitten, J., Bentley, L. & Barlow, V. *Systems Analysis and Design Methods, 2nd. ed.* (Homewood, IL: Irwin, 1989).
- [125] Lawler, *op. cit.*, page 110.

- [126] Elmaghraby, *op. cit.*, p. 8.
- [127] Elmaghraby, *op. cit.*
- [128] Metzger, P. *Managing a Programming Project 2nd ed.*, (Englewood Cliffs, NJ: Prentice Hall, 1981), p. 86.
- [129] Metzger, P. *op. cit.*
- [130] Boehm, B. *op. cit.*
- [131] Gilb, T. *op. cit.*
- [132] Metzger, P. *op. cit.*, p. 146.
- [133] Pressman, R. *Making Software Engineering Happen: A Guide for Instituting the Technology* (Englewood Cliffs, NJ: Prentice Hall, 1988), page 119.
- [134] Humphrey, W. S. *Managing the Software Process* (Reading, MA: Addison Wesley, 1989), page 222.
- [135] Norman, R. *Integrated Development Environments in Support of Information Systems Design Methodologies and Systems Analysts' Productivity*, PhD Dissertation, University of Arizona, 1987.
- [136] Norman, R. *CASE Productivity Perceptions of Software Engineering Professionals*, CACM 32(9), 1989.
- [137] Norman, R. *op. cit.*, page 188.
- [138] Many of the bibliographic citations discuss the tasks of the coding stage. The most notable are Metzger, Pressman, Humphrey, and Freeman.
- [139] Gilb, T. *Principles of Software Engineering Management* (Reading, MA: Addison Wesley, 1988), page 258-259.
- [140] McGarry, F., Voltz, S. & Valett, J. "Determining Software Productivity Factors in the SEL", *Proceedings of the 11th Annual Software Engineering Workshop*, NASA Goddard Space Flight Center, December 3, 1986.
- [141] Humphrey, W. *op. cit.*, page 404-405.
- [142] Poston, R. "Software Testing", coursebook, PEI Inc., 1984.
- [143] Grady, R. and Caswell, D. *op. cit.*
- [144] Boehm, B. *op. cit.*
- [145] Fleckenstein, W. "Challenges in Software Development", *IEEE Computer*, March 1983, pp. 60-64.

[146] Mowshowitz, A. "On the Market Value of Information Commodities", report to the National Science Foundation, 1990.

Bibliography

Abbott, *An Integrated Approach to Software Development* (New York: Wiley, 1989).

Abdel-Hamid, T. K. "The Dynamics of Software Development Project Management: An Integrative Systems Dynamics Perspective", PhD Thesis, Sloan School of Management, Massachusetts Institute of Technology, 1984.

Abdel-Hamid, T. K. & Madnick, S. E. "Lessons Learned from Modeling the Dynamics of Software Development", *CACM* (32)12 1989, pp. 1426-1455.

Ackerman, A. F., Fowler, P. J. & Ebenau, R. G. "Software Inspections and the Industrial Production of Software", Unpublished paper, 1983.

Alchian, A. A. & Demsetz, H. "Production, Information Costs, and Economic Organization", *American Economic Review* (62)5 1972, pp. 124-144.

Alreck, P. & Settle, R. *The Survey Research Handbook* (Homewood, IL: Irwin, 1985).

Aranow, E. "Is CASE Too Immature for Real Integration?", *Software Magazine* May 1990, pp. 89-96.

Aronowitz, S. "The Myth of Full Employment", *The Nation* n.d. 1986, pp. 135-137.

Arrow, K. J. "The Economics of Information", in Dertouzos, M. & Moses, J. (eds.) *The Computer Age: A Twenty Year View* (Cambridge, MA: MIT Press, 1983), pp. 306-317.

Aukerman, R., Schooley, R., Nord, D. & Nord, J. "The Importance of Selected Systems Analysis and Design Tools and Techniques as Determined by Industry Systems Analysts", *ACM SIGCSE Bulletin* (21)3 1989, pp. 30-34.

Austin, S. "New On-line Choices for Business", *Business Computer Systems* n.d. 1985, pp. 50-57.

Babcock, C. "Applications Backlogs", *Computerworld* December 14, 1987.

Baram, F. & Steinberg, G. "Selection Criteria for Analysis and Design CASE Tools", *ACM SIGSOFT Software Engineering Notes* (14)6 1989, pp. 73-80.

Batt, G. T. "Case Technology and the Systems Development Life Cycle: A Proposed Integration of Case Tools with DoD (Department of Defense) STD-2167A", Master's thesis, Naval Postgraduate School, 1989.

Baumol, W. *Economic Theory and Operations Analysis 2nd ed.* (Englewood Cliffs, NJ: Prentice Hall, 1965).

Beer, G. "The Cobb-Douglas Production Function", *Mathematics Magazine* (53)1 1980, pp. 44-48.

Bell, D. "The Social Framework of the Information Society", in Dertouzos, M. & Moses, J. (eds.) *The Computer Age: A Twenty Year View* (Cambridge, MA: MIT Press, 1983), pp. 163-211.

Bellin, D. "Statistics from Survey of Special Librarians", unpublished paper, 1987.

Bellin, D. "The Online Information Industry: Free Trade or Stranglehold?", *Government Data Systems* June 1988, pp. 8-9.

Bellin, D. & Suchman, S. *The Structured Systems Development Manual* (Englewood Cliffs, NJ: Yourdon-Prentice Hall, 1990).

Black, S. H. & Marchand, D. A. "Assessing the value of information in organizations: A challenge for the Eighties", University of South Carolina, Bureau of Government Research and Service, 1980. Also published in *The Information Society* (1)3 1982, pp. 191-225.

Boehm, B. W. "A Spiral Model of Software Development and Enhancement", *IEEE Computer* May 1988, pp. 61-72.

Boehm, B. *Software Engineering Economics* (Englewood Cliffs, NJ: Prentice Hall, 1981).

Boone, G. "CASE's Next Step: Process", *Computerworld* (23)47 1989, pp. 43-45.

Bourne, C. P. "On-line systems: History, Technology, and Economics", *JASIS* (31)0 1980, pp. 155-160.

Braunstein, Y. M. "Costs and benefits of library information: The user point of view", *Library Trends* (28) 1979, pp. 79-87.

Braunstein, Y. M. "The functioning of information markets", *Information Reports and Bibliographies* (11)1 1982, pp. 3-20.

Braunstein, Y. M. "Information as a Factor of Production: Substitutability and Productivity", *The Information Society* (3)3 1985, pp. 261-273.

Brooks, F. P. *The Mythical Man-Month: Essays on Software Engineering* (Reading, MA: Addison Wesley, 1982).

Bryce, M. & Associates Inc. "What's In A Word?", *Management Visions* (6)2 April 1990, pp. 1-3.

Buckley, F. J. "Standard set of useful software metrics is urgently needed", *IEEE Computer* July 1989, pp. 88-89.

Budgen, D. & Sinovec, R. "Introduction to Software Design", Carnegie Mellon University SEI Curriculum Module SEI-CM-2-1.2, 1987.

Burch, J. G. "In a Competitive World, the Strongest Weapon is Information", Data Management n.d. 1986, pp. 22-28.

Burrill, C. W. & Ellsworth, L. W. *Modern Project Management: Foundations for Quality and Productivity* (Tenafly, NJ: BEA Inc., 1980).

Cave, W. C. & Salisbury, A. B. "Controlling The Software Life Cycle - The Project Management Task", IEEE Transactions on Software Engineering (4)4 1978, pp. 326-334.

Cherry, C. & Edmorgon, W. (Eds.) *The Age of Access* (London: Longwood Publishing Group, 1985).

Chmura, L., Norcio, A. & Wicinski, T. "Evaluating Software Design Processes by Analyzing Change Data Over Time", IEEE Transactions on Software Engineering (16)7 1990, pp. 729-740.

Cleveland, H. "Information as a Resource", The Futurist 1982, pp. 34-39.

Collofello, J. S. "The Software Technical Review Process", Carnegie Mellon University SEI Curriculum Module SEI-CM-3-1.2, 1987.

Cooley, M. *Architect or Bee?* (Boston: South End Press, 1980).

Cooper, R. B. "Review of Management Information Systems Research: A Management Support Emphasis", Information Processing & Management (24)1 1988, pp. 73-102.

Crosby, P. B. *Quality Without Tears: The Art of Hassle-Free Management* (New York: McGraw Hill, 1984).

Cruise, R. *Information, Economics, and Power* (London: Hodder & Stoughton, 1983).

Curtis, B., Krasner, H. & Iscoe, N. "A Field Study of the Software Design Process for Large Systems", CACM (31)11 1988, pp. 1268-1287.

Davis, W. *Systems Analysis and Design: A Structured Approach* (Reading, MA: Addison Wesley, 1983).

DeMarco, T. *Structured Analysis and System Specification* (Englewood Cliffs, NJ: Yourdon-Prentice Hall, 1979).

Deming, W. *Out of the Crisis* (Cambridge, MA: MIT Center for Advanced Engineering Study, 1986).

- Dolatta, T. *Data Processing in the 1980's* (New York: Wiley, 1976).
- Dunn, R. & Ullman, R. *Quality Assurance for Computer Software* (New York: McGraw Hill, n.d.).
- Dyer, M. "The Management of Software Engineering Part IV: Software Development Practices", IBM Systems Journal (19)4 1980, pp. 451-465.
- Eisenhart, C. "Expression of the uncertainties of final results", Science (160) 1968, pp. 1201-1204.
- Elias, A. W. "The economics of online data base creation", Paper presented at the Annual Meeting of the American Society for Information Science, 1979.
- Ellison, R. "Software Development Environments", Carnegie Mellon University SEI Technical Memorandum SEI-86-TM-10, 1986.
- Ellison, R., Huff, C., Morris, E., Smith, D., & Zarrella, P. "Tools Within Software Engineering Practice", Carnegie Mellon University SEI Technical Review 1989, pp. 11-20.
- Elmaghraby, S. *Activity Networks: Project Planning and Control by Network Models* (New York: Wiley, 1977).
- Er, M. C. "A Critical Review of the Literature on the Organizational Impact", IEEE Technology and Society (8)2 1989, pp. 17-23.
- Fagan, M. E. "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal (15)3 1976, pp. 182-211.
- Fallon, C. *Value Analysis to Improve Productivity* (New York: Wiley-Interscience, 1971).
- Feltham, G. "The Value of Information", Accounting Review (43)4 1968, pp. 684-696.
- Firth, R., Wood, B., Pethia, R., Roberts, L., Mosley, V. & Dole, "A Classification Scheme for Software Development Methods", Carnegie Mellon University SEI Technical Report CMU/SEI-87-TR-41, 1987.
- Firth, R., Mosley, V., Pethia, R., Roberts, L., & Wood, W. "A Guide to the Classification and Assessment of Software Engineering Tools", Carnegie Mellon University Technical Report CMU/SEI-87-TR-10 ESD-TR-87-111, 1987.
- Fleckenstein, W. "Challenges in Software Development", IEEE Computer March 1983, pp. 60-64.
- Frank, W. *Critical Issues in Software: A Guide to Software Economics, Strategy,*

and Profitability (New York: Wiley, n.d.).

Freeman, P. *Software Perspectives: The System is the Message* (Reading, MA: Addison Wesley, 1987).

Frenkel, K. "Toward Automating the Software-Development Cycle", *CACM* (28)6 1985, pp. 578-589.

Fronidizi, R. *What is Value? An Introduction to Axiology 2nd. ed.* (LaSalle, IL: Open Court, 1971).

Fulkerson, D. R. "A Network Flow Computation for Project Cost Curve", *Management Science* (7)2 1961, pp. 167-178.

Garson, B. *Electronic Sweatshop* (New York: Penguin, 1990).

Gibson, M. "The CASE Philosophy", *Byte* April 1988, pp.209-218.

Gilb, T. *Principles of Software Engineering Management* (Reading, MA: Addison Wesley, 1988).

Glass, R. L. *Modern Programming Practices: A Report from Industry* (Englewood Cliffs, NJ: Prentice Hall, 1982).

Goldberg, R. & Lorin, H. *The Economics of Information Processing Volume 1* (New York: Wiley, 1982).

Goldberg, R. & Lorin, H. *The Economics of Information Processing Volume 2* (New York: Wiley, 1982).

Gotlieb, C. *The Economics of Computers: Costs, Benefits, Policies, and Strategies* (Englewood Cliffs, NJ: Prentice Hall, 1985).

Grady, R. & Caswell, D. *Software Metrics: Establishing A Company-Wide Program* (Englewood Cliffs, NJ: Prentice Hall, 1987).

Greenbaum, J. *In the Name of Efficiency: Management Theory and Shopfloor Practice in Data-Processing Work* (Philadelphia: Temple University Press, 1979).

Griffiths, J. M. "The value of information and related systems, products, and services", in Williams, M. W. (ed.) *Annual Review of Information Science and Technology Volume 17* (White Plains, NY: Knowledge Industry Publications, 1982), pp. 269-284.

Hager, J. A. "Software Cost Reduction Methods in Practice", *IEEE Transactions on Software Engineering* (15)12 1989, pp. 1638-1644.

Hall, K. "The Economic Nature of Information", *The Information Society* (1) 1981, pp. 143-166.

- Halstead, M. *Elements of Software Science* (New York: Elsevier, 1976).
- Haney, W.V. "Serial Communication of Information in Organizations", in Marlick, S. & VanNess, E. (eds.) *Concepts and Issues in Administrative Behavior* (Englewood Cliffs, NJ: Prentice Hall, 1962), pp. 150-164.
- Hanson, S. J. & Rosinski, R. R. "Programmer Perceptions of Productivity and Programming Tools", *CACM* (28)2 1985, pp. 180-189.
- Harary, F., Norman, R. Z. & Cartwright, D. *Structural Models: An Introduction to the Theory of Directed Graphs* (New York: Wiley, 1965).
- Harary, F. *Graph Theory* (Reading, MA: Addison Wesley, 1969).
- Harris, R. B. *Precendence and Arrow Networking Techniques for Construction* (New York: Wiley, 1978).
- Harris, C. L. "Information Power", *Business Week* n.d. 1985, pp. 108-114.
- Hayes, R. & Borko, H. "Mathematic Models of Information System Use", *Information Processing & Management* (19)3 1983, pp. 173-186.
- Hirschheim, R. & Klein, H. K. "Four Paradigms of Information Systems Development", *CACM* 32(10) 1989, pp. 1199-1216.
- Hirschhorn, L. *Beyond Mechanization* (Cambridge, MA: MIT Press, 1984).
- Hirshleifer, J. "Where Are We in the Theory of Information", *American Economic Review* (63)2 1973, pp. 31-39.
- Howard, R. *Brave New Workplace* (New York: Penguin, 1985).
- Hughes, C. T. & Clark, J. D. "The Stages of CASE Usage", *Datamation* (2)n.d. 1990, pp. 41-44.
- Humphrey, W. S. "Characterizing the Software Process: A Maturity Framework", Carnegie Mellon University SEI Technical Report 87-TR-11 ESD-TR-87-112, 1987.
- Humphrey, W. S. *Managing the Software Process* (Reading, MA: Addison Wesley, 1989).
- Humphrey, W. & Kellner, M. "Software process modeling: principles of entity process models," *Proceedings of the 11th International Conference on Software Engineering* (1989: IEEE).
- Information Industry Association: "The Outlook for the Information Industry", *The Business of Information 1983: Volume 2 - The Outlook for the Information*

Industry (Washington, DC: Information Industry Association, 1984).

Jeong, D. "The Nature of the Information Sector in the Information Society: An Economic and Societal Perspective", *Special Libraries* (81)3 1990, pp. 230-235.

Jones, C. "A Survey of Programming and Specification Techniques", *Proceedings, Specifications of Reliable Software* (1979: IEEE), pp. 91,103.

Jones, C. *Programmer Productivity* (New York: McGraw Hill, 1986).

Jones, T. "Measuring Programming Quality and Productivity", *IBM Systems Journal* (17)1 1978, pp. 39-63.

Kass, E. M. "How AD/Cycle Adds Up", *Information Week* n.d. 1990, pp. 29-31.

Kellner, M. "Software Process Modeling: Value and Experience", *Carnegie Mellon University SEI Technical Review* 1989, pp. 23-54.

Kelly, T. J. "The Dynamics of R&D Project Management", *Masters Thesis, Sloan School of Management, Massachusetts Institute of Technology*, 1970.

Kemerer, C. F. "An Empirical Validation of Software Cost Estimation Models", *CACM* (30)5 1987, pp. 416-429.

King, J. L. & Schrems, E.L. "Cost-benefit analysis in information systems development and operation", *Computing Surveys* (10)0 1978, pp. 19-34.

Kochen, M. "Information Science Research: The Search for the Nature of Information", *JASIS* (35)3 1984, pp. 194-199.

Koehler, J., Anatol, K., & Applebaum, R. *Organizational Communication: Behavioral Perspectives* (New York: Holt Rinehart & Winston, 1976).

Kraut, R., Dumais, S. & Koch, S. "Computerization, Productivity, and Quality of Work-life", *CACM* 32(2) 1989, pp. 220-238.

Kumar, K. & Bjorn-Anderson, N. "A Cross-Cultural Comparison of IS Designer Values", *CACM* 33(5) 1990, pp. 528-538.

Lamberton, D. "The Economics of Information and Organization", in Williams, M. E. (ed.) *Annual Review of Information Science and Technology* (19) 1984, pp. 4-30.

Langlois, R. "Systems Theory and the Meaning of Information", *JASIS* November 1982, pp. 395-399.

Laranjeira, L. A. "Software Size Estimation of Object Oriented Systems", *IEEE Transactions on Software Engineering* (16)5 1990, pp. 510-522.

Lawler, E. *Combinatorial Optimization: Networks and Matroids* (New York: Holt Rhinehart & Winston, 1976).

Levy, L. S. *Taming the Tiger: Software Engineering and Software Economics* (New York: Springer-Verlag, 1987).

Lewis, P. M. "Information Systems is an Engineering Discipline", CACM (32)9 1989, pp. 1045-1046.

Lide, D. R. Jr. "Critical data for critical needs", Science (212) 1981, pp. 1343-1349.

Linger, R. C. "The Management of Software Engineering Part III: Software Design Practices", IBM Systems Journal (19)4 1980, pp. 432-450.

Low, G. C., Jeffery, D. R. "Function Points in the Estimation and Evaluation of the Software Process", IEEE Transactions on Software Engineering (16)1 1990, pp. 64-71.

Luqi "A Graph Model for Software Evolution", IEEE Transactions on Software Engineering (16)8 1990, pp. 917-927.

Machlup, F. *The Production and Distribution of Knowledge in the United States* (Princeton, NJ: Princeton University Press, 1962).

Machlup, F. "An Economist's Reflections on an Institute for the Advanced Study of Information Science", JASIS (30)2 1979, pp. 111-113.

Machlup, F. "Uses, value, and benefits of information", Knowledge: Creation, Diffusion, Utilization (1) 1979, pp. 62-81.

Machlup, F. *Knowledge, its creation, Distribution & Economic Significance* (Princeton, NJ: Princeton University Press, 1980).

Machlup, F. & Mansfield, U. (eds.) *The Study of Information: Interdisciplinary Messages* (New York: Wiley, 1983).

Machlup, F. *The Economics of Information and Human Capital* (Princeton, NJ: Princeton University Press, 1984).

Martin, J. "Complex RAD Cycle Can Be Distilled to Manageable Phases", PC Week February 2, 1990, page 52.

Martin, J. "Integrated CASE Tools a Must for High-Speed Development", PC Week January 22, 1990, page 78.

Marx, K. & Engels, F.: *The Origin of the Family, Private Property, and the State* (New York: International Publishers, 1972).

Mason, R. M. & Creps, J. E. Jr. *Information Services: Economics, Management, and Technology* (San Diego, CA: Westview, 1981).

McCabe, T. "A Complexity Measure", *IEEE Transactions on Software Engineering* December 1976, pp. 308-320.

McClure, C. R. "The information rich employee and information for decision making: Review and comments", *Information Processing & Management* (14) 1978, pp. 382-394.

McCracken, D. & Jackson, M. "Life Cycle Concept Considered Harmful", *ACM SIGSOFT Software Engineering Notes* (7)2 1982, pp. 29-32.

McDowell, R. "Changes Present a Challenge for Us All", *The Consultant* January-February 1985, page 21.

McDowell, R. "A Pilot Study Can Help anagers Make Sound Decisions About Office Technology", *The Consultant* January-February 1985, pp. 13-15.

McGarry, F., Voltz, S. & Valett, J. "Determining Software Productivity Factors in the SEL", *Proceedings of the 11th Annual Software Engineering Workshop*, NASA Goddard Space Flight Center, December 3, 1986.

McMenamin, S. & Palmer, J. *Essential Systems Analysis* (Englewood Cliffs, NJ: Yourdon-Prentice Hall, 1984).

Mercer, B. D. "Weapon System Software Acquisition and Support: A Theory of System Structure and Behavior", MS Thesis, Air Force Institution of Technology, 1982.

Metzger, P. W. *Managing A Programming Project 2nd. ed.* (Englewood Cliffs, NJ: Prentice Hall, 1981).

Mills, H. D. "The Mangagement of Software Engineering Part I: Principles of Software Engineering", *IBM Systems Journal* (19)4 1980, pp. 415-420.

Mills, H. D. *Software Productivity* (Boston, MA: Little Brown & Co., 1983).

Mills, H., Linger, R. & Hevner, A. *Principles of Information Systems Analysis and Design* (New York: Academic Press, 1986).

Mowshowitz, A. "Entropy and the Complexity of Graphs: I. An Index of the Relative Complexity of a Graph", *Bulletin of Mathematical Biophysics* (30) 1968, pp. 175-204.

Mowshowitz, A. *The Conquest of Will: Information Processing in Human Affairs* (Reading, MA: Addison Wesley, 1976).

Mowshowitz, A. *Inside Information: Computers in Fiction* (Reading, MA: Addison

Wesley, 1977).

Mowshowitz, A. "The Information Marketplace", report to the Office of Technology Assessment, August 1987.

Mowshowitz, A. "The Information Commodity", unpublished presentation overheads, 1988.

Mowshowitz, A. "Information Commodities and Organizational Flexibility", unpublished proposal to the National Science Foundation, January 2, 1989.

Mowshowitz, A. "On the Market Value of Information Commodities", report to the National Science Foundation, 1990.

Mulvaney, J. E. *Analysis Bar Charting: A Simplified Critical Path Analysis Technique* (Cleveland, OH: Chemical Rubber Company Press, 1969).

9to5 National Association of Working Women *Stories of Mistrust and Manipulation* (Cleveland, OH: 9to5 National Organization of Working Women, 1990).

9to5 National Association of Working Women *Computer Monitoring and Other Dirty Tricks* (Cleveland, OH: 9to5 National Organization of Working Women, 1986).

9to5 National Association of Working Women *Hidden Victims: Clerical Workers, Automation, and the Changing Economy* (Cleveland, OH: 9to5 National Organization of Working Women, 1985).

Nolan, R. L. "Advice to an Executive on the way to the 21st Century", The Consultant January-February 1985, page 6.

Nolan, R. L. "Recharting Business and Computing in the Decade Ahead", The Consultant January-February 1985, pp. 1-5.

Norman, R. J. "Integrated Development Environments in Support of Information Systems Design Methodologies and Systems Analysts' Productivity", PhD Thesis, Business Administration, University of Arizona, 1987.

Norman, R. J. & Nunamaker, J. F. Jr. "CASE Productivity Perceptions of Software Engineering Professionals", CACM (32)9 1989, pp. 1102-1108.

Nortan, D. P. "Information Technology - an Investment and an Asset", The Consultant n.d. 1985, pp. 7-8.

Norton, D. P. "Information Technology - The Investment Enigma", The Consultant n.d. 1985, pp. 9-12.

OECD Organization for Economic Cooperation and Development: *Information*

Activities, Electronics and Telecommunications Technologies: Impact on Employment, Growth and Trade, 1981.

O'Neill, D. "The Management of Software Engineering Part II: Software Engineering Program", IBM Systems Journal (19)4 1980, pp. 421-431.

Ostberg, O. "Review of Workplace Aspects of Robot-Based Production", report to the National Institute for Occupational Safety and Health, July 1984.

Page-Jones, M. *Practical Project Management: Restoring Quality to DP Projects and Systems* (New York: Dorset House, 1985).

Page-Jones, M. *The Practical Guide to Structured Systems Design*, 2nd. ed. (Englewood Cliffs, NJ: Yourdon - Prentice Hall, 1988).

Perrone, G. "Low-cost CASE: tomorrow's promise emerging today", IEEE Computer November 1987, pp. 104-110

Phister, M. *Data Processing Technology and Economics* 2nd. ed. (Boston, MA: Digital Press, 1979).

Polack, A. J. "Practical Applications of CASE Tools on DoD Projects", ACM SIGSOFT Software Engineering Notes (15)1 1990, pp. 73-78.

Porat, M. *The Information Economy: Definition and Measurement*, US Department of Commerce, Office of Telecommunications Publication 77-12(1), May 1977.

Porter, M. E. & Millar, V. E. "How Information Gives You Competitive Advantage", Harvard Business Review July-August 1985, pp. 149-160.

Poston, R. "Software Economics", course notes, PEI Inc. June 1984.

Poston, R. "Software Testing", coursebook, PEI Inc., 1984.

Pressman, R. *Making Software Engineering Happen: A Guide for Instituting the Technology* (Englewood Cliffs, NJ: Prentice Hall, 1988).

Putnam, L. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", IEEE Transactions on Software Engineering July 1978, pp. 345-361.

Quinnan, R. E. "The Management of Software Engineering Part IV: Software Engineering Management Practices", IBM Systems Journal (19)4 1980, pp. 466-477.

Radice, R. A., Harding, J. T., & Phillips, R. W. "A Programming Process Study", IBM Systems Journal (24)2 1985, pp. 91-101.

Randers, J. (ed.) *Elements of the Systems Dynamics Method* (Cambridge, MA:

MIT Press, 1980).

Repo, A. J. "The Value of Information: Approaches in Economics, Accounting, and Management Science", JASIS (40)2 1989, pp. 68-85.

Richan, P. H. "Telecommunications, Matter, and the Formation of Value", IEEE Technology and Society March 1988, pp. 19-22.

Richardson, G. P. & Pugh, G. L. *Introduction to System Dynamics Modeling with Dynamo* (Cambridge, MA: MIT Press, 1981).

Roberts, N., et. al. *Introduction to Computer Simulation: The System Dynamics Approach* (Reading, MA: Addison Wesley, 1981).

Roberts, E. B. *Managerial Aspects of System Dynamics* (Cambridge, MA: MIT Press, 1981).

Robey, D. "Computer Information Systems and Organization Structure", CACM 24(10) 1981, pp. 679-687.

Rockart, J. F. & Flannery, L. S. "The Management of End User Computing", CACM(26)10 1985, pp. 776-784.

Rosson, M. B., Maass, S., & Kellogg, W. A. "The Designer as User: Building Requirements for Design Tools from Design Practice", CACM 31(11) 1988, pp. 1288-1298.

Rubin, H. A. & Verdugo, G. "Modeling the Benefits of Your Applications Productivity Strategy", Nolan, Norton & Co., 1986.

Rubin, H. A. "How to Become a Software Engineering 'Big Foot'", American Programmer January 1989, pp. 22-33.

Rubin, H. A. "What You Need To Know About Software Engineering to Survive in the 1990's", unpublished paper, November 1989.

Scacchi, W. "Models of Software Evolution: Life Cycle and Process", Carnegie Mellon University SEI Curriculum Module SEI-CM-10-1.0, 1987.

Schiller, H. I. "Information-A Shrinking Resource", The Nation January 4, 1986, pp. 708-710.

Schiller, A. R. & Schiller, H. I. "Commercializing Information", The Nation October 4, 1986, pp. 306-309.

Schneiderman, B. & Carroll, J. M. "Ecological Studies of Professional Programmers", CACM 31(11) 1988, pp. 1256-1258.

Schonberger, R. J. *Operations Management: Productivity and Quality 2nd. ed.*

(Texas: Business Publications Inc., 1985).

Shaiken, H.: *Work Transformed*, (New York: Holt Rinehart & Winston, 1986).

Shallis, M. *The Silicon Idol* (New York: Schocken, 1984).

Shannon, C. & Weaver, W. *The Mathematical Theory of Communication* (Chicago: University of Illinois Press, 1949).

Shaw, M., Giuse, D. & Reddy, R. "What A Software Engineer Needs to Know: I. Program Vocabulary", Carnegie Mellon University Technical Report CMU-CS-89-180, 1989.

Shere, K. D. *Software Engineering and Management* (Englewood Cliffs, NJ: Prentice Hall, 1988).

Simpson, J. W. "Information Megatrends", Online March 1985, pp. 6-7.

Smith, A. Jr. "Making A Case For CASE", Business & Technology Consultant (2)6 1990, pp. 27-29.

Smith, D. "CASE in Context", SEI Bridge March 1990, pp. 4-5.

Snyders, J. "A CASE of Unknown Identity", InfoSystems October 1987, pp. 54-59.

Sola Pool, I. de, "Tracking the Flow of Information", Science (221)4611 1983, pp. 609-613.

Soloway, E., Pinto, J., Letovsky, S., Littman, D. & Lampert, R. "Designing Documentation to Compensate for Delocalized Plans", CACM 31(11) 1988, pp. 1259-1267.

Spence, A. M. "An economist's view of information", in C. Cuadra (ed.) *Annual Review of Information Science and Technology* (9) 1974, pp. 57-78.

Statland, N. "Payoffs Down the Pike: A CASE Study", Datamation April 1, 1989, pp. 32-56.

Strassmann, P. A. "Managing the Costs of Information", Harvard Business Review (54)5 1976, pp. 123-142.

Swamy, M. N. S. & Thulasiraman, K. *Graphs, Networks, and Algorithms* (New York: Wiley, 1981).

Swanson, E. & Beath, C. "Departmentalization in Software Development and Maintenance", CACM 33(6) 1990, pp. 658-667.

Tahvanianen, V. & Smolander, K. "An Annotated CASE Bibliography", ACM

SIGSOFT Software Engineering Notes (15)1 1990, pp. 79-92.

Tate, P. "A Badly Needed Sense of the Value of Information", *Datamation* March 15, 1989, page 78.

Tausworthe, R. "The Work Breakdown Structure in Software Project Management", *Journal of Systems and Software* (1)1 1980, pp. 181-186.

Taylor, R. S. "Value-added processes in the information life-cycle", *JASIS* (33) 1982, pp. 341-346.

Taylor, R. S. "Benefits and costs of information use", in *Proceedings of the 45th ASIS Annual Meeting* (White Plains, NY: Knowledge Industry Publications, 1982), pp. 296-297.

Taylor, R. S. "Value-added processes in the information life cycle", final report to the National Science Foundation, Grant No. IST-81-06080, 1983.

Taylor, R. S. "Value-added processes in document-based systems: Abstracting and indexing services", *Information Services and Use* (4)3 1984, pp. 127-146.

Taylor, R. S. "Value-added processes in information analysis", in *Proceedings of the 46th ASIS Annual Meeting* (White Plains, NY: Knowledge Industry Publications, 1984), pp. 7-9.

Taylor, R. *Value-Added Processes in Information Systems* (Norwood, NJ: Ablex, 1986).

Thadhani, A. "Factors Affecting Programmer Productivity During Application Development", *IBM Systems Journal* (23)1 1984, pp. 19-35.

Thayer, R. H. "Modeling a Software Engineering Project Management System", PhD Thesis, Department of Computer Science, University of California at Santa Barbara, 1979.

Thibodeau, R. & Dodson, E. N. "Life Cycle Phase Inter-relationships", *Journal of Systems and Software* (1)3 1980, pp. 203-211.

Thomson, G.: *Capitalism and After: The Rise and Fall of Commodity Production*, (London: China Policy Study Group, 1973).

Tracz, W. "Where Does Reuse Start?", *ACM SIGSOFT Software Engineering Notes* (15)2 1990, pp. 42-46.

Turoff, M. "Information, Value and the Internal Marketplace", New Jersey Institute of Technology, September 1983.

Verity, J. "Where It All Began", *Datamation* December 1, 1984, pp. 153-154.

Vincent, J., Waters, A., & Sinclair, J. *Software Quality Assurance: Volume II A Program Guide* (Englewood Cliffs, NJ: Prentice Hall, 1988).

Wallace, R., Stockenberg, J. & Charette, R. *A Unified Methodology for Developing Systems* (New York: McGraw Hill, 1987).

Ward, P. *System Development Without Pain* (Englewood Cliffs, NJ: Yourdon-Prentice Hall, 1984).

Whitten, J., Bentley, L. & Barlow, V. *Systems Analysis & Design Methods 2nd. ed.* (Homewood, IL: Irwin, 1989).

Wiest, J. D. & Levy, F. K. *A Management Guide to PERT/CPM 2nd. ed.* (Englewood Cliffs, NJ: Prentice Hall, 1977).

Williamson, M. "Editorial", CASE Strategies (1)1 June 1989, page 2.

Wilson, R.J. *Introduction to Graph Theory* (New York: Academic Press, 1972).

Winkler, C. "Better Project Management in Uncertain Times", Datamation June 1, 1990, pp. 95-98.

Wolverton, R. W. "The Cost of Developing Large-Scale Software", IEEE Transactions on Computers June 1974, pp. 283-303.

Wood, E. G. *Added Value* (London: Business Book, 1978).

Yourdon, E. *Modern Structured Analysis* (Englewood Cliffs, NJ: Prentice Hall, 1989).

Yovits, M. C. & Foulk, C. R. "Experiments and Analysis of Information Use and Value in a Decision-Making Context", JASIS (36)2 1985, pp. 63-81.

Zuboff, S. *In the Age of the Smart Machine* (New York: Basic Books, 1988).