Aalto University
School of Economics

Cost estimation in global software development - Review of estimation techniques

Information Systems Science

Master's thesis

Juha Koskenkylä

2012

Department of Information and Service Economy
Aalto University
School of Economics

Aalto University
School of Economics

Cost estimation in global software development

Review of estimation techniques

Master's Thesis
Juha Koskenkylä
August 1st, 2012
Information and Service Management

Approved in the Department of Information and Service Economics _____
and awarded the grade _____

_____

1st inspector's first name family name 2nd inspector's first name family name

AALTO UNIVERSITY SCHOOL OF ECONOMICS ABSTRACT
Department of Information and Service Economy August 1st, 2012
Master's thesis
Juha Koskenkylä

ABSTRACT

Objectives of the Study

The objective of this research is to review existing software costs estimation techniques and to understand how these techniques can be applied in global software development considering its specific nature. The research problem is formulated with following questions: What factors impact software development cost in global software development? How these factors are considered in the available software cost estimation techniques? How available software estimation techniques can be applied in global software development?

Findings and conclusions

Geographic and temporal distance, differences in language and culture, social factors, problem arising from organization structure, processes and projects, barriers deriving from infrastructure and product architecture have an impact, usually increasing, on the costs and effort in global software development. These cost factors can also have a compounding effect on each other, increasing the overall negative impact on the software development.

These cost factors are not considered in the evaluated software cost estimation techniques except for few techniques and factors. COCOMOII takes into account some impact of geographic distance, cultural differences and social factors. In addition, there are four adaptations of COCOMOII or CoBRA techniques which have been developed to especially address the characteristics of global software development. These techniques contain almost all global software development related cost factors.

Most of the evaluated software cost estimation techniques are still suitable and applicable for estimation of global software development, but they require set-up and calibration work. Only expertise-based techniques and global software development related adaptations of COCOMOII or COBRA can be directly used. Almost all techniques require data from past global software development projects as a precondition for set-up or calibration.

Keywords

Global Software Development, Cost Estimation, Effort Estimation

AALTO-YLIOPISTON KAUPPAKORKEAKOULU
Tieto- ja palvelutalouden laitos
Pro gradu-tutkielma
Juha Koskenkylä

TIIVISTELMÄ
01.08.2012

ABSTRAKTI

Tutkimuksen tavoitteet

Tämän tutkielman tavoitteena on käydä läpi olemassaolevia ohjelmistokehityksen kustannusten arviointimenetelmiä ja ymmärtää kuinka näitä menetelmiä voidaaan soveltaa globaaliin ohjelmistokehitykseen huomioiden sen erityispiirteet. Tutkimusongelma voidaan muotoilla seuraavin kysymyksin: Mitkä tekijät vaikuttavat kustannuksiin globaalissa ohjelmistokehityksessä? Miten nämä tekijät on huomioitu olemassaolevissa ohjelmistokustannusten arviointimenetelmissä? Miten olemassaolevia ohjelmistokustannusten arviointimenetelmiä voidaan soveltaa globaalissa ohjelmistokehityksessä?

Tulokset ja päätelmät

Maantieteellinen ja aikavyöhykkeiden etäisyys, kieli- ja kulttuurierot, sosiaaliset tekijät, ongelmat organisaatiossa, prosesseissa ja projekteissa, rakenteet infrastruktuurissa ja ohjelmistoarkkitehtuurissa vaikuttavat, yleensä kasvattavasti, kustannuksiin ja työmääriin globaalissa ohjelmistokehityksessä. Näillä kustannustekijöillä voi myös olla yhteisvaikutus, joka edelleen lisää negatiivista kokonaisvaikutusta ohjelmistokehitykseen.

Näitä kustannustekijöitä ei ole huomioitu olemassaolevissa ohjelmistokehityksen kustannusten arviointimenetelmissä muutamaa poikkeusta lukuunottamatta. COCOMOII -menetelmä huomioi maantieteellisen etäisyyden, kulttuurierot sekä sosiaaliset tekijät jollain tasolla. Lisäksi on olemassa neljä sovitettua versiota COCOMOII tai CoBRA -menetelmistä, jotka on kehitetty erityisesti huomioimaan globaalin ohjelmistokehityksen piirteet. Nämä menetelmät sisältävät melkein kaikki globaaliin ohjelmistokehityksen liittyvät kustannustekijät.

Valtaosaa olemassaolevista ohjelmistokehityksen kustannusten arviointimenetelmistä voidaan kaikesta huolimatta käyttää globaalin ohjelmistokehityksen kustannusten arviointiin, mutta näiden menetelmien käyttäminen vaatii alustus- ja kalibrointityötä. Vain osaamisperusteisia menetelmiä ja erityisesti globaaliin ohjelmistokehitykseen kehitettyjä COCOMOII ja CoBRA – menetelmien versioita voidaan käyttää suoraan globaalin ohjelmistokehityksen kustannusten arviointiin. Lähes kaikki menetelmät vaativat myös toteutuneista globaaleista ohjelmistoprojekteista menetelmien alustamista ja kalibrointia varten.

Avainsanat

Globaali ohjelmistokehitys, Kustannusten estimointi, Työmäärän estimointi

ACKNOWLEDGEMENTS

To Katri and Meri Helfrid.

Thank you for the love and support during this long research and journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. BACKGROUND

Globalization of the software industry has grown rapidly during the recent decade and it is expected to continue to grow. The advances and changes in communication technology, standardization of software platforms and applications, digitalization of work, reorganization of work processes for outsourcing, new business models, and global political, educational and free trade factors have driven the rapid shift to a global software industry in which globally distributed software development is a reality (ACM, 2006).

Globally distributed software development, or global software development (GSD), refers to software development done by multiple teams in different geographic locations. This is different to the tradition software development, i.e. local software development, which has been done by one or multiple teams located in same country and usually even in the same physical location.

The number of global software development projects has been steadily increasing. In 2006, thirty percent of the World's 1000 largest firms are offshoring work and this percentage was expected to still increase. The amount of work offshored is expected to grow by 20 to 30 percent for the offshoring industries in India and China. (ACM, 2006)

Many western companies choose to development their software in Eastern Europe, South America, and Asia to gain substantial cost savings due to the lower labor rates offered by these regions. The other often stated reasons for global software development also include (Herbsleb and Moitra, 2001, Noll et al., 2010):

- The need to capitalize on the global resource pool to successfully and cost-competitively use scarce resources, wherever located;
- The business advantages of proximity to the market, including knowledge of customers and local conditions, as well as the good will engendered by local investment;
- The quick formation of virtual corporations and virtual teams to exploit market opportunities;

- Severe pressure to improve time-to-market by using time zone differences in "round-the-clock" development;
- The need for flexibility to capitalize on merger and acquisition opportunities wherever they present themselves.[1]

Although all these reasons can be seen as important for increase in the global software development, I will focus on the software development costs in this research as the lower costs of software development are very often used as the main justification for the decision to develop software globally instead of developing it locally.

There seems to be a strong business reason to develop software globally due to the lower labor costs in developing regions. However this can be misleading considering the challenges in the global software development. Studies report that compared to single-site work, cross-site work takes much longer, and requires more people for work of equal size and complexity (Carmel, 1999, Herbsleb and Grinter, 1999, Herbsleb et al., 2001). Recent studies also indicate that the benefits in global software development are neither clear-cut nor can their realization be taken for granted as many people seem to believe (Conchúir et al., 2009).

In software development the physical separation, especially across several time zones and between different cultures and languages, requires additional activities and effort. For example the separation increases the needs for team building, knowledge transfer for asynchronous collaboration, creating an architecture that is easily distributed and that minimizes cross-site communication, and facilitating communication among teams working on parts of the architecture that are interrelated.

Global software development can also increase the requirements regarding development processes, project management practices, architecture, quality, collaboration tools and so forth. All this translates into a substantial overhead in the day-to-day operations of global software development projects which can exceed the benefits from the lower labor rates in the developing

[1] In this research, bullet list is used for unordered list of items

countries. Hence it is crucial to be able to understand and estimate the total costs for global software development in order to evaluate whether it is more beneficial compared to local software development.

There is a very large variety of different software cost estimation techniques. However many of these techniques have been developed before the trend of global software development and hence they might not take into account the additional challenges in global software development to accurately estimate the costs needed to develop software in global manner.

It also remains unclear if there are any new estimation techniques or adaptations of existing techniques created to address these additional challenges in global software development. If such techniques exist, it is uncertain if they consider all the necessary aspects of global software development in the estimation.

Hence this research focuses on studying the suitability of existing software cost estimation techniques to global software development. The research tries to improve the understanding on how costs can be estimated for global software development.

1.2. RESEARCH PROBLEM AND METHOD

This research studies software cost estimation in global software development. The purpose of this research is to review existing software costs estimation techniques and to understand how these techniques can be applied to global software development considering its specific nature.

The research problem is formulated with following questions:

1. What factors impact software development cost in global software development?
2. How these factors are considered in the available software cost estimation techniques?
3. How available software estimation techniques can be applied to global software development?[2]

[2] In this research, numbered list is used for ordered list of items

The research is conducted by reviewing available research on the software cost estimation and global software development. The primary sources of information for this research are the databases of Institute of Electrical and Electronics Engineers (IEEE) and Association of Computing Machinery (ACM).

1.3. RESEARCH STRUCTURE

This research has seven sections. The first section, this section, works as an introduction to the research. It also defines the research problem, method and structure.

The following two sections focus on the software cost estimation. The second section presents an overview on software cost estimation. It describes what software cost estimation is, how and when it is done, and why software cost estimates are usually very inaccurate. In the third section, an example of software cost estimation process is presented.

In the fourth section, different categories of software cost estimation techniques are presented and few examples from each category are described in more details.

The fifth section focuses on global software development and describes the additional barriers in global software development. The section also presents recent software cost estimation techniques which have been created especially for global software development.

In the sixth section, we evaluate how different software cost estimation techniques consider the additional cost factors in global software development. We will also discuss how the techniques can be applied to the global software development.

The seventh and final section concludes the research and its findings.

2. SOFTWARE COST ESTIMATION

2.1. WHAT IS A COST ESTIMATE?

In project management, cost estimation is the process of developing an approximation of the monetary resources needed to complete project activities (PMBOK, 2000). Usually software is developed in projects and hence software cost estimate can be considered as an approximation of

the monetary resources needed to complete software. The inputs, tools, techniques and outputs for a generic cost estimate are shown in the figure below (PMBOK, 2000).

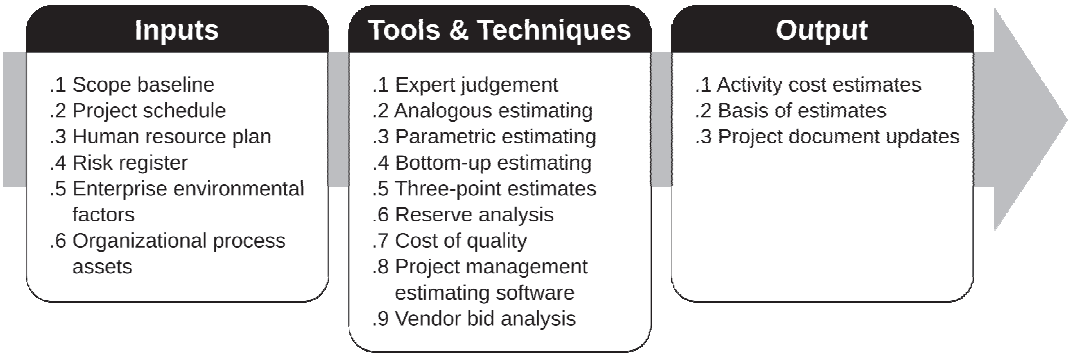

Figure 1: Cost estimation in project management (PMBOK, 2000)

The key outputs are activity cost estimates, basis of estimates and project document updates. Project documents are project management related outputs which include updates to the project documents such as the risk register (PMBOK, 2000).

In the research literature of software effort estimation, there is usually no difference made between effort and costs. This is mainly caused by the fact that in software development, practically all costs are personnel costs, which are directly dependent on effort. In global software development, however, cost rates might differ between different sites, meaning that effort at one site might cause significantly higher costs than effort at another site (Conchúir et al., 2009). Even though, in this research effort and costs are used as synonyms unless otherwise specified.

2.2. ACTIVITY COST ESTIMATES

The activity costs estimates are quantitative assessments of the costs required to complete software related work (PMBOK, 2000). Costs are estimated for all resources that will be needed for development the software. This includes, but is not limited to, labor, materials, equipment, services, and facilities, as well as special categories such as an inflation allowance or risk/contingency costs.

Cost estimates are expressed in units of some currency (i.e., dollars, euro, yen, etc.); although in some instances other units of measure, such as man days or man months, are used to facilitate comparisons by eliminating the effects of currency fluctuations.

NASA's guide on software cost estimation gives the following as the basic equation for software costs (NASA, 2003):

Total SW Costs=SW Development Labor Costs+Other Labor Costs+Non Labor Costs

Below table summarizes key elements which NASA includes in the software cost estimates (NASA, 2003).

Table 1: Cost elements of software development (NASA, 2003)

Cost type	Item	Description
SW Development Costs	Software Systems Engineering	Performed by the software architects, software system engineers, and subsystem engineers for activities like functional design, software requirements, and interface specification.
	Software Engineering	Performed by the cognizant engineer and developers to unit design, develop code, unit test, and integrate software components
	Software Test Engineering	Covers test engineering activities from writing test plans and procedures to performing any level of test above unit testing.
Other labor	Software management and support	Performed by the project element manager, software manager, technical lead, and system administration to plan and direct the software project and software configuration management.
	Test-bed development	
	Development environment support	
	Software system-level test support	including development and simulation software
	Assembly, test and launch operations	
	Administration and support costs	
	Software Quality Assurance	
	Independent Verification & Validation	
	Other review and support charges	
Non labour costs	Support and services	workstations, test-bed boards & simulators, ground support equipment, network and phone charges, etc
	Software Procurement	including development environments, compilers, licences, CM tools, test tools and development tools
	Travel and trips	related to customer reviews and interfaces, vendor and remove team visits, project related conference,
	Training	

2.3. BASIS OF ESTIMATE

The Basis of Estimate should provide a clear and complete understanding of how the cost estimate was derived (PMBOK, 2000). The amount and type of additional details supporting the

cost estimate varies per application area. Supporting detail for activity cost estimates may include:

- Documentation of the basis of the estimate (i.e., how it was developed)
- Documentation on the coverage of the estimate (i.e. what was estimated)
- Documentation of all assumptions made
- Documentation of any known constraints
- Indication of the range of possible estimates (e.g., $10,000 (±10%) to indicate that the item is expected to cost between a range of values)
- Indication of the confidence level of the final estimate.

It is very important that all information related to the estimation is documented clearly in the basis of estimate as this enables more thorough reviews, makes it easier to revise estimates at future dates when assumptions may need to be revised. Without a clear basis of estimate, the activity estimates become useless as there will not be a clear understanding on what those estimates include.

2.4. WHEN SOFTWARE COSTS ARE ESTIMATED?

Software cost estimation is not an independently and individual activity performed once in a software project. Instead it is an iterative process which continues during the different phases and which is heavily linked to the other activities in the software development. The estimates are derived in large part from the requirements of the software, and they will be strongly affected by the tools, process, and other attributes associated with the project. The costs estimation as part of software development activities is illustrated in the figure below (McConnell, 2006).

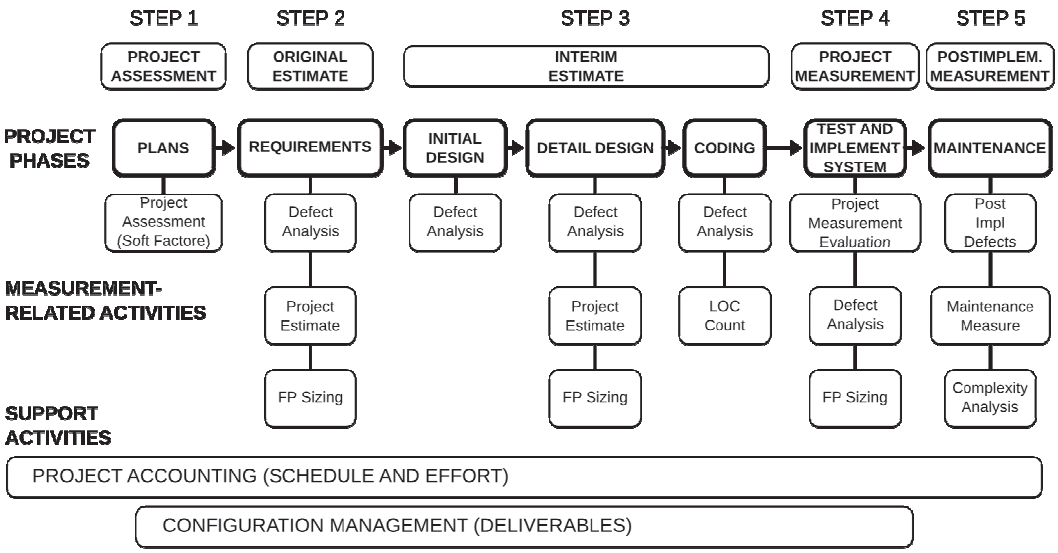

Figure 2: Cost estimation as part of software development activities (McConnell, 2006)

The estimates should be refined during the course of the software development to reflect additional details as those become available. For any project larger than trivial, multiple cost estimates will be prepared during the course of development, including but not limited to the following:

- A rough pre-requirements guesstimate
- An initial formal estimate derived from the project requirements
- One or more midlife estimates, which reflect requirements changes
- A final cost accumulation using project historical data.

In the early life-cycle phases, software cost estimation is closely related to design activities, where the interaction between these activities is iterated many times as part of doing design trade studies and early risk analysis. Later on in the life-cycle, cost estimation supports management activities – primarily detailed planning, scheduling and risk management.

2.5. WHY SOFTWARE COST ESTIMATES ARE INACCURATE?

It seems to be an industry wide problem that the software costs are very inaccurately estimated and these cause huge problems in software development projects. These problems are mentioned

in many of the research literature software cost estimation (Brooks, 1975, Boehm, 1981, Heemstra, 1992, McConnell, 2006). There are even ”industry jokes” on humoristic estimation methods like one in which the real effort in a project can be found by multiplying the work estimate with Phi and adding 30% extra on top. In reality it is not very rare that some work amounts are estimated this much below the actual effort (Haikala and Märijärvi, 1997).

Software development is a process of gradual refinement. A project starts out with a very general vision of the software. The concept of the software is refined based on the product requirement and project goals. The accuracy of a project estimate will increase as the work progresses through the software development life cycle. With respect to estimation inaccuracy as a cause of cost growth, Barry Boehm found that cost estimates made in the early stages of the software life-cycle could be off by as much as a factor of four. This inaccuracy is primarily due to the lack of clear understanding of the software requirements. (Boehm, 1981)

The below figure shows the rate at which the accuracy of cost estimates improves as requirements specificity increases. Estimates cannot be more accurate than requirements at any point. This graph is known as ”the Cone of Uncertainty” (Boehm, 1981).

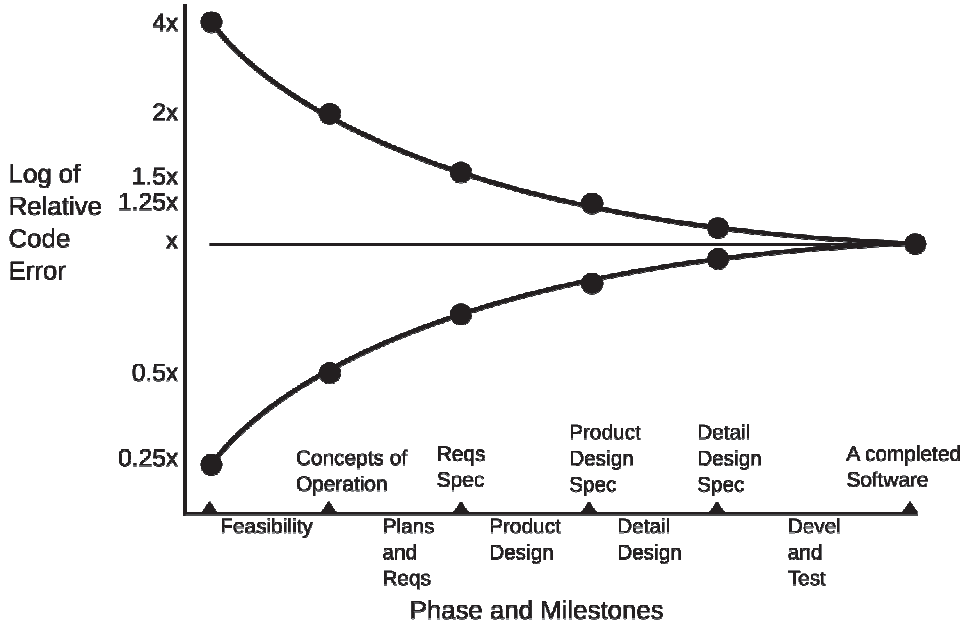

Figure 3: The Cone of Uncertainty (Boehm, 1981)

The table below summarizes some of the common reasons identified in the research literature for the inaccuracy of software cost estimates.

Table 2: Reasons for inaccurate software cost estimates identified in research literature

Reason for inaccurate estimates	Citations
Lack of data on completed projects and comparison on estimates vs actuals.	Heemstra 1992
Lack of time and resources to perform estimate accurately.	Heemstra 1992, Boehm 1981. Jørgensen and Moløkken-Østvold 2004.
Unclear, incomplete and unreliable are not available specifications.	Heemstra 1992. McConnell 2006. Lederer and Prasant 1995. Subramaniam and Breslawski 1995.
Characteristics of software and software development. For example, the level of abstraction, complexity, measurability of product and process, innovative aspects, etc.	Heemstra 1992, Boehm 1981. McConnell 2006. Van Genuchten 1991. Subramaniam and Breslawski 1995.
Number and complexity of cost drivers. Examples are size and complexity of the software, commitment and participation of the user organization, experience of the development team.	Heemstra 1992. McConnell 2006
Rapid changes in information technology (IT) and the methodology of software development are a problem for a stabilization of the estimation process.	Heemstra 1992
Lack of knowledge and experience in estimation.	Heemstra 1992, Boehm 1981
A bias of experts to towards underestimation by forgetting overhead functions and lower productivity due to sick-leaves, computer problems, other work, communication etc	Heemstra 1992, Boehm 1981, Hihn and Habib-agahi 2000. McConnell 2006. Phan 1990. Lederer and Prasant 1995. Van Genuchten 1991. Jørgensen and Moløkken-Østvold 2004.
People are basically optimistic and desire to please. Everybody would like the software to be small and easy. High estimates lead to confrontation situations, which people generally prefer to avoid.	Boehm 1981. McConnell 2006. Phan 1995.
The estimator estimates the time it would take to perform the task personally, ignoring the fact that a lot of work will be done by less experienced people, and junior staff with a lower productivity rate.	Heemstra 1992, Boehm 1981. Subramaniam and Breslawski 1995. Jørgensen and Moløkken-Østvold 2004.
There exists a serious mis-assumption of a linear relation between the required capacity per unit of time and the available time. This would mean that software developed by 25 people in two years could be accomplished by 50 people in one year	Heemstra 1992
The estimator tends to reduce the estimates to some degree, in order to make the bid more acceptable or to comply with earler rough estimates.	Heemstra 1992, McConnell 2006. Boehm 1981.

There does not seem to be any simple solution for improving the accuracy of software cost estimates. The key to establish reliable estimates is to use a variety of methods and tools and then analyze the results by different techniques to understand why the estimates provided by one might differ significantly from those provided by another (Boehm, 1981). If the estimator can explain such differences to a reasonable level of satisfaction, then it is likely that he or she has a good understanding of the factors which are driving the costs of the project at hand; and thus will be better equipped to support the necessary project planning and control functions performed by management.

3. SOFTWARE COST ESTIMATION PROCESS

In order to establish a reliable cost estimate for software, a structured approach with significant amount of work is needed. The software cost estimation can be seen as a small size project which needs to be carefully planned, managed and followed-up.

Many organizations have different processes for software cost estimation. These processes vary in many aspects and there does not seem to be one common process which is used in all organizations and in research. Two processes for software cost estimation have been presented below as examples. These originate from Barry Boehm's Software Engineering Economics (Boehm 1981) and NASA's Handbook for Software Cost Estimation (NASA, 2003).

Table 3: Software cost estimation process from Barry Boehm (Boehm, 1981)

Number	Action	Description
Step 1	Establish cost-estimating objectives	The objectives are needed to set the level of detail and work required to perform the subsequent steps in the process. These objectives should include at least the information needed for decision making, accuracy of the different estimates, and effort/cost/time spent on the estimation.
Step 2	Generate a project plan for required data and resources	A project plan should be generated for the necessary data and resources. This plan should state why, what, when, who, where, how time, effort and cost are spend on the estimating activity. The plan can vary from a simple textual notes to a full blown project plan depending on the objectives and size of the software.
Step 3	Pin down software requirements	The software requirements and specifications need to be clarified so that these are as unambiguous as possible when considering the estimating objectives. It is valuable to document any assumptions that were made in the estimating the cost of developing the software, particularly if they were generous or conservative.
Step 4	Work out as much detail as feasible	The target is to find out as much information and details on the software as possible and feasible considering the cost-estimating objectives. Usually the more information and details there are available, the more accurately the estimates will be.
Step 5	Use several independent cost estimation techniques and sources	In the actual estimating, it is recommended to use several different techniques as none of the alternatives is better than the others in all aspects. Most of the techniques have complementary strenghts and weaknesses.
Step 6	Compare and iterate different estimates	It is necessary to compare and iterate the estimating results from different techniques and sources. This is the most valuable aspect of using several independeny cost-estimation techniques as it enables the investigating of why the techniques give different estimates, which can significantly improve the estimates.
Step 7	Follow-up	Once the estimates have been completed and the software project itself has started, it makes sense to gather data on the actual costs and progress and compare these to the estimates.

Table 4: Software cost estimation process from NASA (NASA, 2003)

Number	Action	Description
Step 1	Gather and Analyze Software Functional & Programmatic	Analyze and refine software requirements, software architecture, and programmatic constraints.
Step 2	Define the Work Elements and Procurements	Define software work elements and procurements for specific project.
Step 3	Estimate Software Size	Estimate size of software in logical Source Lines of Code (SLOC).
Step 4	Estimate Software Effort	Convert software size estimate in SLOC to software development effort. Use software development effort to derive effort for all work elements.
Step 5	Schedule the effort	Determine length of time needed to complete the software effort. Establish time periods of work elements of the software project WBS and milestones.
Step 6	Calculate the Cost	Estimate the total cost of the software project.
Step 7	Determine the Impact of Risks	Identify software project risks, estimate their impact, and revise estimates.
Step 8	Validate and Reconcile the Estimate Via Models and	Develop alternate effort, schedule, and cost estimates to validate original estimates and to improve accuracy.
Step 9	Reconcile Estimates, Budget, and Schedule	Review above size, effort, schedule, and cost estimates and compare with project budget and schedule. Resolve inconsistencies.
Step 10	Review and Approve the Estimates	Review and approve software size effort, schedule, and cost estimates.
Step 11	Track, Report, and Maintain the Estimates	Compare estimates with actual data. Track estimate accuracy. Report and maintain size, effort, schedule, and cost estimates at each major milestone.

4. SOFTWARE COST ESTIMATION TECHNIQUES

4.1. OVERVIEW

The software cost estimation techniques can be categorized in different ways. One way is to divide the techniques into six major categories as shown in the figure below (Chulani et al., 1999). This section describes these categories and presents some examples from each category.

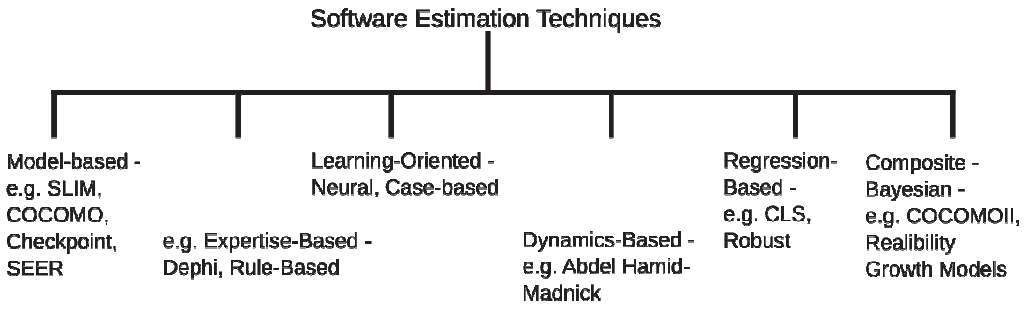

Figure 4: Software estimation techniques (Chulani et al., 1999)

Each category and/or technique within a category tends to its relative strengths and weaknesses. For example regression and model-based techniques give good results when they can be calibrated with high-quality data available from similar past experiences, but they fail to deal with exceptional conditions and with poor low-quality input and calibration data.

There exists a wide variety of research on different software cost estimation techniques and on their accuracy. Most of the researchers think that there is no superior technique compared to others in all aspects and that the strengths and weaknesses of different techniques are often complementary (Boehm, 1981, Heemstra, 1992, NASA, 2003). Hence it is important to use a combination of techniques, and to carefully compare and iterate the software cost estimates obtained from each in order to produce the most realistic estimates. The particular combination of techniques can be chosen based on the objectives of cost estimation.

4.2. MODEL-BASED TECHNIQUES

The model-based techniques contain a mathematical model which uses certain cost factors in determining effort, cost and/or schedule for software development project. Theory or

experimentation determines usually the functional form of the mathematical models. In many cases the model needs to be calibrated with the historical data from past projects.

The model-based techniques are objective and they are not influenced by human factor like desire to win or please. They are also repeatable and they often support sensitive analysis on the estimates. In addition, they can give accurate results when they are calibrated with high-quality data available from similar past experience.

On the other hand, the model-based techniques are unable to consider exceptional conditions with for example personnel and teamwork. In addition, they cannot compensate for poor and inaccurate input data. As the models are calibrated with data from past projects, it is uncertain to what extent the models can predict future projects using new technology, program languages, etc.

Next sub-sections present three different model-based techniques: Putnam / Software Life-cycle Model (SLIM), Function Point Analysis (FPA) and Constructive Cost Model (COCOMO). There are also many other popular model-based techniques like Price-S (Freiman and Park, 1979), Estimacs (Rubin, 1983), SEER-SEM (Jensen, 1983), and Checkpoint (Jones, 1997).

4.2.1. PUTNAM MODEL / SOFTWARE LIFE-CYCLE MODEL

The Putnam model was developed by Larry Putnam in the late 1970s (Putnam, 1978). It describes the time and effort required to finish a software project of specified size using Rayleign curve function. Software Life-cycle Model (SLIM) is the name given by Larry Putnam to the proprietary suite of tools his company, QSM Inc, has developed based on the Putnam model.

The central part of Putnam's model is called software equation:

$$S = E \times (Effort)^{1/3} t_d^{4/3}$$

, where td is the software delivery time in years, E is the environment factor that reflects the software development capability. The size, S, is in effective source lines of code (ESLOC) which includes new and modified programming lines without blanks and comments. Effort is the total effort applied to the project in person-years. The second important relation is:

$$Effort = D_0 \times t_d^3$$

, where D0 is a parameter called manpower build-up which ranges from 8 (entirely new software with many interfaces) to 27 (rebuilt software). New systems of a given size that interact with other systems will have the greatest entropy and take longer to develop. Rebuilds of old systems or composites where large portions of the logic and code have already been developed (and hence have reduced the entropy) will take the least time to develop. New stand-alone systems and other combinations will fall in between.

Combining these two equations, the following models for effort and schedule estimation can be obtained:

$$Effort = (D_0^{4/7} \times E^{-9/7}) \times S^{9/7}$$

and

$$t_d = (D_0^{-1/7} \times E^{-3/7}) \times S^{3/7}$$

The environment factor, E, can be replaced with a scaling factor, B, and process productivity. The scaling factor is a function of project size and the process productivity is the ability of a particular software organization to produce software of a given size at a particular defect rate.

The Putnam Model can be used to plot software development effort as a function of time as shown in figure below. The points along the curve represent the estimated total effort to complete the project at some time. The curves show that the total effort decreases as the time to complete the project is extended which is one of the distinguishing features of the Putnam model.

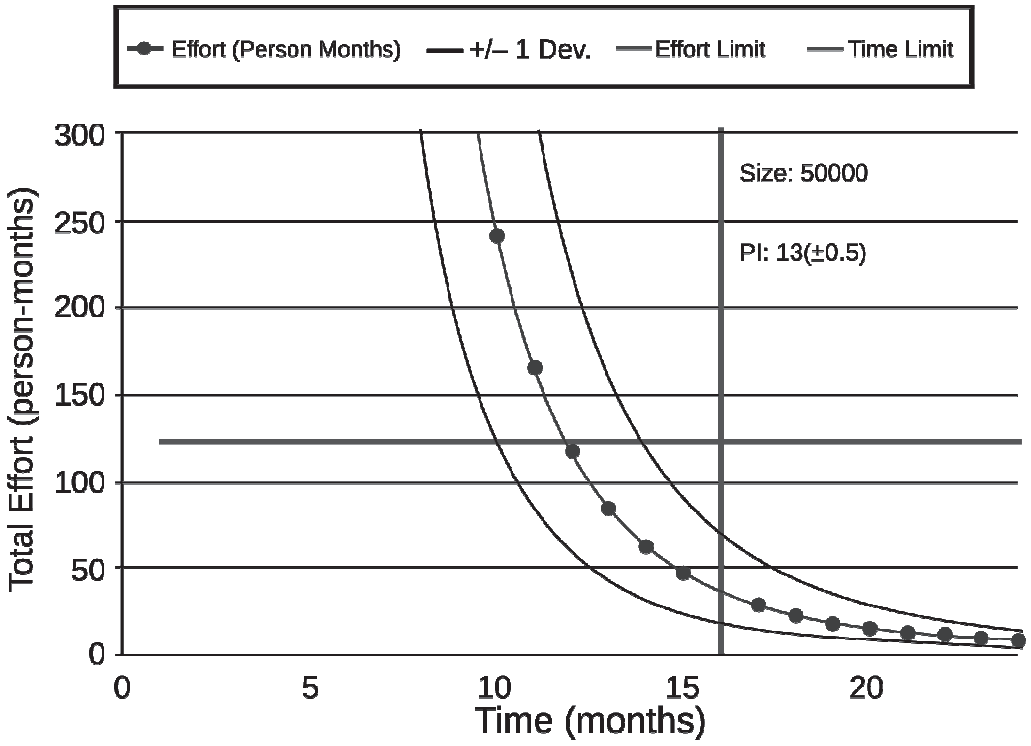

Figure 5: Software development effort as function of time in Putnam model (Putnam, 1978)

The Putnam model can be calibrated with data from past completed projects or if data is not available then a set of calibration questions. One of the key advantages to the model is the simplicity with which it is calibrated. Most software organizations, regardless of maturity level can easily collect size, effort and duration (time) for past projects.

4.2.2. FUNCTION POINT ANALYSIS

Function points and function point analysis (FPA) was developed by Allan Albrecht (Albrecht, 1979). Albrecht was looking for a method to measure productivity in software development and the functional point analysis was developed as an alternative measure to the number of lines of code. A function point is a unit of measurement to express the amount of business functionality an information system provides to a user.

The functional point analysis is simple method that is based on the number of functions in the software. It is programming language or fourth generation tool independent. In the method, functions of the software are identified and each one is categorized into one of the five types:

- The external input type
- The external output type
- The external inquiry type
- The logical internal file type
- The external interface file type.

Once a function is identified and categorized into a type, it is then assessed for complexity (simple, average, or complex) and assigned a number of function points based on complexity and type. The total sums function points for all functions is the unadjusted number of function points. This indication for nominal size is then adjusted, using 14 technical characteristics. The software development cost (in money or hours) of a single unit is calculated from past projects.

Function count		←—Max range: Factor * 2—→			
		Level of information processing function			
Type ID	Description	Simple	Average	Complex	Total
IT	External input	--*3 = --	--*4 = ---	--*6 = ---	-----
OT	External output	--*4 = --	--*5 = ---	--*7 = ---	-----
FT	Logical internal file	--*7 = --	--*10 = --	--*15 = --	-----
EI	External interface file	--*5 = --	--*7 = ---	--*10 = --	-----
QT	External inquiry	--*3 = --	--*4 = ---	--*6 = ---	-----
FC		Total unadjusted function points			

↑ Maximum range factor 2.5 ↓

General information processing characteristics					
	Characteristics	DI		Characteristics	DI
C1	Data communications	---	C8	On-line update	---
C2	Distributed functions	---	C9	Complex processing	---
C3	Performance	---	C10	Re-usability	---
C4	Heavily used configuration	---	C11	Installation ease	---
C5	Transaction rate	---	C12	Operational ease	---
C6	On-line data entry	---	C13	Multiple sites	---
C7	End-user efficiency	---	C14	Facilitate change	---
PC				Total degree of influence	---

DI	Values			
Not present or no influence	= 0	Average influence	= 3	
Insignificant influence	= 1	Significant influence	= 4	
Moderate influence	= 2	Strong influence, throughout	= 5	

FC	(Function count)	≡	Total unadjusted function points
PC	(Process complexity)	≡	Total degree of influence
PCA	(Process complexity adjustment)	=	0.65 + 0.01 * PC
FP	(Function point measure)	=	FC * PCA

Figure 6: Overview of Function Point Analysis (Albrecht, 1979)

Each identified function maps to an end-user business functionality, such as a data entry for an input or a user query for an inquiry. This distinction is important because it tends to make the functions measured in function points map easily into user-oriented requirements, but it also tends to hide internal functions (e.g. algorithms), which also require resources to implement.

Later the function point analysis technique has been refined and/or extended several times. For example feature point analysis extends the function points to include algorithms as a new class (Jones 1997) to better include internal functionalities into technique. An algorithm is defined as the set of rules which must be completely expressed to solve a significant computational problem. Each algorithm is given a weight ranging from 1 (elementary) to 10 (sophisticated algorithms) and the feature point is the weighted sum of the algorithms plus the function points. This measurement is especially useful for systems with little input/output and high algorithmic complexity.

4.2.3. CONSTRUCTIVE COST MODEL

The Constructive Cost Model (COCOMO) is an algorithmic cost estimation technique developed by Barry Boehm (Boehm, 1981). It is based on a study of 63 software development projects at TRW Aerospace during 1970's. These software development projects used waterfall model of software development and procedural programming languages like Assembly and COBOL. In 1990's the COCOMO was enhanced to COCOMOII in order to meet the requirements for estimating modern software development projects and processes (Boehm et al., 2000a).

COCOMO uses a simple regression formula where parameters have been derived from past project data and are adjusted based on the current project characteristics. The original COCOMO has three increasingly more detailed and accurate forms (Basic, Intermediate, and Detailed). These form and COCOMOII are presented in the following sub-sections.

4.2.3.1. MODES OF SOFTWARE DEVELOPMENT

COCOMO identifies three different modes of software development. These are organic, semi-detached and embedded. The different software development modes have cost-estimating relationships which are similar in form, but which yield different cost estimates for software products of the same size and hence it is important to distinguish between these different modes.

In the organic mode, relatively small team develops software in a highly familiar, in-house environment. Most people connected with the project have extensive experience in working with related systems within the organization, and have a thorough understanding of how the system under development will contribute to the organization's objectives. An organic-mode project is relatively relaxed about the way the software meets its requirements and interface specifications which is another reason for both the higher productivity and the smaller diseconomy of scale on an organic-mode project. Other factors characteristic of organic-mode software projects are:

- A generally stable development environment with very little concurrent development of associated new hardware and operational procedures
- Minimal need for innovative data processing architectures or algorithms
- A relatively low premium on early completion of the project
- Relative small size (<50 KDSI).

These factors usually correlate with higher project productivity and smaller project diseconomies.

The semidetached mode of software development represents an intermediate stage between the organic and embedded modes in either of two ways:

- An intermediate level of the project characteristic
- A mixture of the organic and embedded mode characteristics.

The embedded mode is distinguished by the factor that it needs to operate within very tight constraints. The software product must operate within a strongly coupled complex hardware, software, regulations, and operational procedures. In general, the costs of changing the other parts of this complex are so high that their characteristics are considered unchangeable and the software is expected both to conform to their specifications and to take up the slack on any unforeseen difficulties encountered or changes required within the other parts of the complex. These factors contribute both to lower productivity and to greater diseconomies of scale on larger projects.

4.2.3.2. BASIC COCOMO

Basic COCOMO can be applied to small-to-medium size software development projects in a familiar in-house environment. It includes phase distributions of effort, schedule, and activities. It is quick and easy to use, but its accuracy is very limited due its simplicity. It can be used for early and rough estimation of software costs.

Basic COCOMO estimates software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of delivered source instructions (KDSI). The basic COCOMO equations take the following forms:

$$Effort(ManMonths) = a_b \times (KDSI)^{b_b}$$

$$DevelopmentTime(Months) = c_b \times (Effort)^{d_b}$$

$$Productivity = KDSI / Effort$$

$$AverageStaffing = Effort / DevelopmentTime$$

The coefficients a_b, b_b, c_b and d_b are defined per project mode and they are given in the following table:

Table 5: Basic COCOMO coefficients

Software project	a_b	b_b	c_b	d_b
Organic	2,40	1,05	2,50	0,38
Semi-attached	3,00	1,12	2,50	0,35
Embedded	3,60	1,20	2,50	0,32

Basic COCOMO doesn't take into account any cost factors like hardware constraints, personnel quality, experience and other project attributes which might have a significant influence on software development costs.

4.2.3.3. INTERMEDIATE COCOMO

Intermediate COCOMO estimates software development effort as function of program size and a set of 15 cost factor attributes. These cost factor attributes are grouped into four categories: software product attributes, computer attributes, personnel attributes, and project attributes. Each of the cost factor attributes determines a multiplying factor which estimates the effect of the attribute on the software development effort. These multipliers are applied to a nominal COCOMO development effort estimate to obtain a refined estimate of software development effort.

In the intermediate COCOMO estimation begins by generating a nominal effort estimate, using scaling equations of the same form as in the Basic COCOMO. This nominal estimate is then adjusted by applying effort multipliers determined from the project's ratings with respect to the other 15 cost drivers. Each of the 15 attributes receives a project rating on a scale of importance that ranges from "very low" to "extra high". An effort multiplier from the table below applies to the rating. The product of all effort multipliers results in an effort adjustment factor (EAF). Typical values for EAF range from 0.9 to 1.4.

Table 6: Effort multipliers in Intermediate COCOMO

		Ratings					
Attribute Category	**Cost driver**	**Very Low**	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
Product attributes	Required software reliability	0,75	0,88	1,00	1,15	1,40	
	Size of application database		0,94	1,00	1,08	1,16	
	Complexity of the product	0,70	0,85	1,00	1,15	1,30	1,65
Hardware attributes	Run-time performance constraints			1,00	1,11	1,30	1,66
	Memory constraints			1,00	1,06	1,21	1,56
	Volatility of the virtual machine environment		0,87	1,00	1,15	1,30	
	Required turnabout time		0,87	1,00	1,07	1,15	
Personnel attributes	Analyst capability	1,46	1,19	1,00	0,86	0,71	
	Applications experience	1,29	1,13	1,00	0,91	0,82	
	Software engineering capability	1,42	1,17	1,00	0,86	0,70	
	Virtual machine experience	1,21	1,10	1,00	0,90		
	Programming language experience	1,14	1,07	1,00	0,95		
Project attributes	Application of software engineering methods	1,24	1,10	1,00	0,91	0,82	
	Use of software tools	1,24	1,10	1,00	0,91	0,83	
	Required development schedule	1,23	1,08	1,00	1,04	1,10	

Intermediate COCOMO formula takes the form:

$$Effort = a_i \times (KLoC)^{b_i} \times EAF$$

, where E is the effort applied in person-months, KLoC is the estimated number of thousands of delivered lines of code for the project, and EAF is the factor calculated based on the cost drivers. The Development time, D, calculation uses E in the same way as in the Basic COCOMO. Intermediate COCOMO uses the same estimating relationship for development schedule, phase distribution, and activity distribution as Basic COCOMO.

The coefficient a_i and the exponent b_i are given in the next table.

Table 7: Intermediate COCOMO coefficients

Software project	a_i	b_i	c_i	d_i
Organic	3,20	1,05	2,50	0,38
Semi-attached	3,00	1,12	2,50	0,35
Embedded	2,80	1,20	2,50	0,32

4.2.3.4. DETAILED COCOMO

Detailed COCOMO incorporates all characteristics of the Intermediate COCOMO with an assessment of the cost driver's influence on individual project phases. This is done by using different efforts multipliers for each cost drivers attribute in each phase. These multipliers are called Phase Sensitivity Effort Multipliers and these determine the amount of effort required to complete each phase of the project.

4.2.3.5. COCOMOII

As software engineering changed, Boehm continued to developed COCOMO model and COCOMOII was published in 2000 (Boehm et al., 2000a) The COCOMOII enables the use of source code lines and function points as reference parameters for the calculation of the projects' Size (S). There are three different COCOMOII-variations, which are Applications Composition, Early Design and Post-Architecture.

The Application Composition model is used to estimate effort and schedule on projects that use Integrated Computer Aided Software Engineering tools for rapid application development. These projects are too diversified but sufficiently simple to be rapidly composed from interoperable components. Typical components are GUI builders, database or objects managers, middleware

for distributed processing or transaction processing, etc. and domain specific components such as financial, medical or industrial process control packages. The Application Composition model is based on object point estimation which is similar to Function Point Analysis.

The Early Design model involves the exploration of alternative system architectures and concepts of operation. It is used in the early stages of a software project when very little may be known about the size and nature of the software to be developed. Usually there is not enough information available to make a detailed fine-grain estimate and hence the model is based on function points (or lines of code when available) and a set of five scale factors and seven effort multiplier (cost factors).

The Post-Architecture model involves the actual development and maintenance of a software product. It is used when top level design is complete and detailed information about the project is available. It estimates the entire development life-cycle and is a detailed extension of the Early-Design model. This model is similar to the Intermediate COCOMO '81. It uses Source Lines of Code and/or Function Points for the sizing parameter, adjusted for reuse and breakage; a set of 17 effort multipliers and a set of five scale factors which determine the economies/diseconomies of scale of the software under development. The five scale factors replace the development modes in the COCOMO '81 model.

Table 8: Effort multipliers in COCOMOII

Attribute Category	**Cost driver**
Product attributes	Required Software Reliability
	Database Size
	Product Complexity
	Required Reusability
	Documentation match to life-cycle needs
Platform attributes	Execution Time Constraints
	Main Storage Constraints
	Platform Volatility
Personnel attributes	Analyst and Programmer Capability
	Application Experience
	Platform Experience
	Language and Tool Experience
	Personnel Continuity
Project attributes	Use of Modern Programming Practices
	Use of Software Tools
	Multisite Development
	Required Development Schedule
	Classified Security Application

The effort equation of COCOMOII is:

$$Effort(PersonMonth) = A \times Size^{E} \times \prod_{i=1}^{17} EM_i$$

, where A is a constant, Size is measured in KSLOC or Function Points, E is a scale factor and EM_i are effort multipliers.

The constant A is a calibration factor that portrays the dimension of the productivity (standard value 2.94), but it can be calibrated with the company's past project data.

The scale factors (E) depends on five factors: development flexibility, architecture/ risk resolution, team cohesion, process maturity and precedentedness. Scale factors exponentially influence the effort of a software development project. These scale factors are cost factors in the same way as the effort multipliers (EM).

Effort multipliers are characteristics of the software development and they have a direct impact on the effort. Effort Multipliers are classified into categories ranging from very low to extra high. Numerical values have been assigned to these categories.

4.3. EXPERTISE-BASED TECHNIQUES

In the expertise-based techniques, the software cost estimates are based on expert opinions and judgmental processes. These techniques capture the knowledge and experience of subject matter experts familiar within a domain of interest, providing estimates based upon a synthesis of the known outcomes of all the past projects and theory.

Expertise-based techniques are usually very good for unprecedented projects and in the absence of quantified, empirical data from past projects. Experts can factor in differences between past project experiences and the new techniques, architectures, or applications involved in the future project. They can also take into account exceptional personnel characteristics and interactions, or other unique project considerations. Expert judgment has also been found relatively accurate if the estimator has significant recent experience in the software domain and estimation process (Hihn and Habib-agahi, 1991).

The weaknesses include lack of sensitivity analysis, dependency on experienced estimators, human-errors and objectivity of the estimators who may be biased, optimistic, pessimistic, or unfamiliar with key aspects of the project.

In the following sub-sections, five different expertise-based techniques are presented. These are Delphi, Work-Breakdown Structure (WBS), Analogy, Work distribution, Top-Down and Bottom-Up techniques,

4.3.1. DELPHI TECHNIQUE

One of the most famous expertise-based techniques is the Delphi technique. The Delphi technique, named after ancient Oracle of Delphi, seeks to develop a consensus among expert group through meetings, questionnaires and surveys. Although it was originally developed by the Rand Corporation in the late 1940s to forecast the impact of technology on warfare (Helmer,

1966), it has been applied to many other areas including cost estimation for software development projects.

The original Delphi technique avoided group discussion, but the Wideband Delphi technique accommodated group discussion between each assessment rounds (Boehm, 1981). This is a useful technique for coming to a conclusion regarding an estimate when the only information available is based more on "expert opinion" and there is no empirical data available.

The Wideband Delphi technique can be used for cost estimation of a software development project in the following way:

1. Project manager presents product specifications and an estimation form to each expert.
2. Project manager calls a group meeting in which the experts discuss estimation issues with the project manager and each other.
3. Experts fill out forms anonymously.
4. Project manager prepares and distributes a summary of the estimates.
5. Project manager calls a group meeting, specifically focusing on having the experts discuss points where their estimates vary widely.
6. Experts fill out forms, again anonymously, and steps 4 to 6 are iterated for as many rounds as appropriate.

During the iteration process the range of the estimates should shrink as the experts seek an estimate they can all agree to. If complete agreement is impossible, average scores can be used to obtain what will still be a quite reliable estimate.

Software development projects have a considerable specialist aspect in them and the project manager cannot be expected to be expert in software development. Therefore the Delphi technique can be used to obtain quite reliable estimates based on the expert consensus. As the Delphi process takes some time and requires many experts to participate, it could be reserved for

areas in which the project manager has little expertise or where the range of initial estimates provided by experts is wide.

4.3.2. WORK BREAKDOWN STRUCTURE

Almost a standard of engineering practice in the development of software, the Work Breakdown Structure (WBS) is delivery-oriented grouping of project elements into a hierarchy that organizes and defines the total work scope of a project (PMBOK, 2000).

WBS is developed by starting with the end objective and successively subdividing it into manageable elements in terms of size, duration, and responsibility (e.g., systems, subsystems, components, tasks, subtasks, and work packages) which include all steps necessary to achieve the objective. Each descending level in the hierarchy represents an increasingly detailed definition of the project work. The lowest level elements, work packages, provide a logical basis for defining activities or assigning responsibilities to a specific person or organization.

WBS helps to determine what work products are needed and what costs are being estimated. Without such definitions, software cost estimates and data lose precision and meaning. When costs are associated with each individual work package, an overall expected value can be determined from the bottom up for total project development cost. The expertise in the WBS method is needed in the determination of the most useful specification of the work packages within the structure and of those estimates and probabilities associated with each work package.

A software WBS consists of two structures, one representing the software product itself, and the other representing the activities needed to build that product (Boehm, 1981). The product hierarchy (Figure 7) describes the fundamental structure of the software, showing how the various software components fit into the overall system. The activity hierarchy (Figure 8) indicates the activities that may be associated with a given software component.

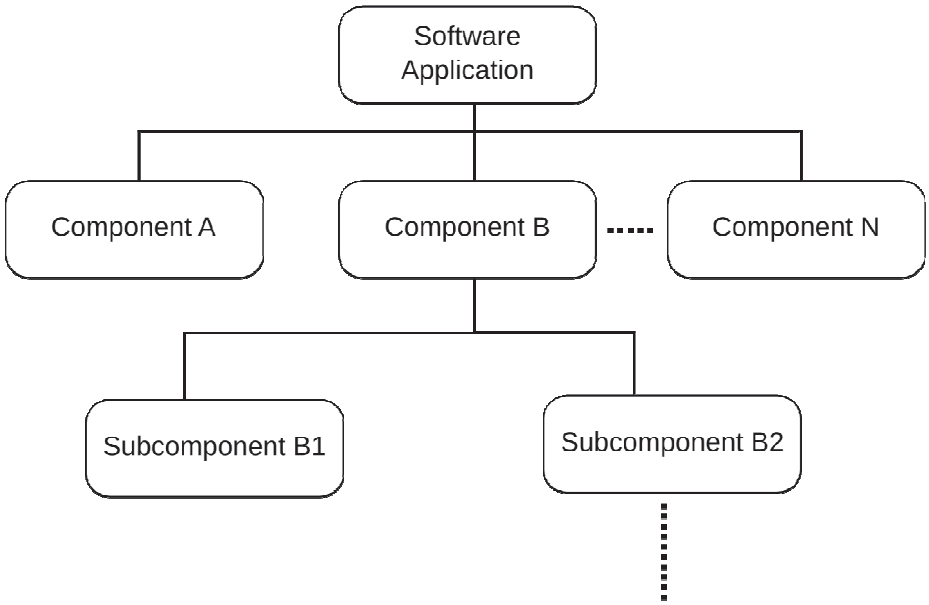

Figure 7: Product hierarchy

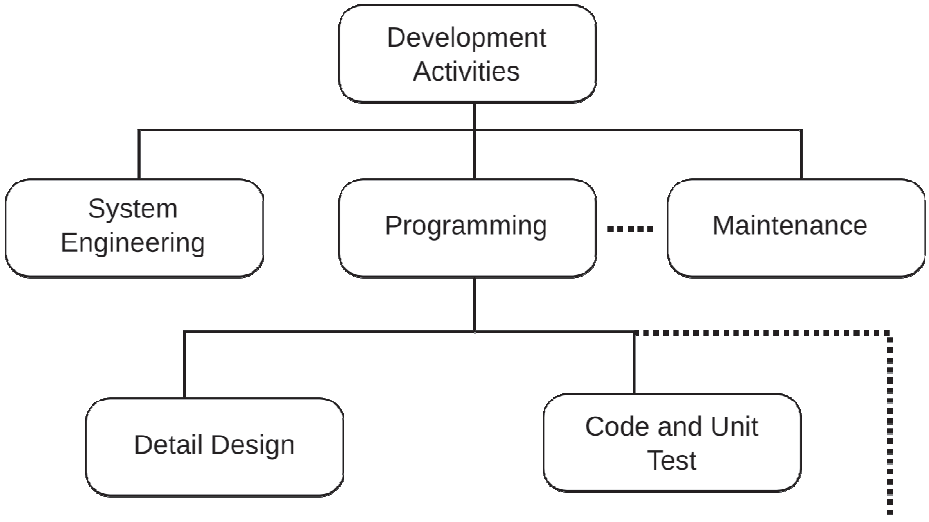

Figure 8: Activity hierarchy

Developing the WBS is a four-step process (Haugan, 2001):

1. Specifying the project objectives and focusing on the products, services or results to be provided to the customer

2. Identifying specially the products, services and results (deliverables or end items) to be provided to the customer

3. Identifying other work areas in the project to make sure that 100 percent of the work is covered and to identify areas that cut across the deliverables, represent intermediate outputs, or complement the deliverables

4. Subdividing each of the items in step 2 and 3 into successive, logical subcategories until the complexity and money value of the elements become manageable units for planning and control purposes (work packages)

The 100% rule is one of the most important principles guiding the development, decomposition and evaluation of the WBS. It states the WBS includes 100% of the work defined by the project scope and captures all deliverables – internal, external, and interim – in terms of the work to be completed, including project management. The rule applies at all levels within the hierarchy: the sum of the work at the "child" level must equal 100% of the work represented by the "parent" and the WBS should not include any work that falls outside the actual scope of the project, that is, it cannot include more than 100% of the work. It is also important to remember that the 100% rule also applies to the activity level. The work represented by the activities in each work package must add up to 100% of the work necessary to complete the work package. (Haugan 2001)

Besides helping in the cost estimation, WBS assists during the project life cycle in following ways (MIL-HDBK-881, 1998):

- It segregates a material item into its component parts, clarifying the relationship among the parts, and clarifying the relationship of the tasks to be completed —to each other and to the end product

- It facilitates effective planning and assignment of management and technical responsibilities
- It aids status tracking of technical efforts, risks, resource allocations, expenditures, and cost/schedule/technical performance
- It helps ensure that contractors are not unnecessarily constrained in meeting item requirements.

Another use of the WBS is to serve as the basis for software cost collection and reporting. Each of the WBS elements can be given a project budget and a job number for people to use in reporting the amount of time they have spend on different project activities. If an organization consistently uses a standard WBS for all of its projects, over time the organization will build up a valuable database reflecting its software development cost distributions. This data can be used to develop a software cost estimation model tailored to the organization's own experience and practices or as an input to the parametric software cost estimation methods. (Boehm, 1981)

4.3.3. ANALOGY

Estimation by analogy is done by comparing a project or some aspects of it with something similar (Boehm, 1981). The comparison can be one or more completed projects to relate their actual costs to an estimate of the cost of a similar project.

For example, an organization wants to estimate costs for software that can handle 5 million online shopping transactions per day. The organization has earlier developed similar software for another customer that can handle 10 million online shopping transactions per day. Using analogy, the costs can be estimated to be roughly 50% of the actual costs of the earlier project.

Estimating by analogy can be done either at the project level or at a subsystem level. The project level has the advantage that all components of the system cost will be considered while subsystem level has the advantage of providing a more detailed assessment of the similarities and differences between the new project and the completed project.

The main strength of estimation by analogy is that the estimate is based on actual experience on a project. This experience can be studied to determine specific differences from the new project, and their likely cost impact. The main weakness of estimation by analogy is that it is not clear to

what degree the previous project is actually representative of the constraints, techniques, personnel, and functions to be performed by the software on the new project. (Boehm, 1981)

Shepperd and Schofield have developed a formal five-step process and software tool, ANGEL (ANaloGy Estimation tool2), for estimation by analogy (Shepperd et al., 1996). This technique contains following steps:

1. Identify the data or features to collect
2. Agree data definitions and collections mechanisms
3. Populate the case base
4. Tune the estimation method
5. Estimate the effort for a new project.

In the ANGEL technique, organization takes a new project for which effort need to be estimated, and attempts to find other similar completed projects. Since these projects are completed, development effort will be known and can be used as a basis for estimating effort for the new project. Similarity is defined in terms of project features, such as number of interfaces, development method, application domain and so forth. Clearly the features used will depend upon what data is available to characterize projects. The number of features is also flexible.

In the first step, the data that is used to characterize analogies between projects is identified for collection. Factors to be considered include beliefs as to what features significantly impact development effort (and are measurable at the time the estimate is required) and what features can easily be collected. These can be for example programming language or number of interfaces in software.

The second step is to agree definitions as to what is being collected. Even within organizations there may be no shared understanding of what is meant by effort. Any estimation program will be flawed, possibly fatally, if different projects are measuring the same features in different ways. It is also important to identify who is responsible for the data collection and when they should

collect the data. Sometimes it can be beneficial to have the same person collecting the data across projects in order to increase the level of consistency.

In the third step, the case base must be populated. In general, more data is preferable although, in most cases, data collection will be an on-going process as projects are completed and their effort data becomes available. However, there appear to be some tradeoffs between the size of the dataset and homogeneity. Some experiences suggest that there is merit in the strategy of dividing highly distinct projects into separate datasets.

The fourth step is to tune the estimation method. The user also will need to experiment with the optimum number of analogies searched for, and whether to use a subset of variables, since some features may not usefully contribute to the process of finding effective analogies. Tuning can make quite a difference to the quality of predictions— typically tuning can yield a twofold improvement in performance—and for this reason the ANGEL tool provides automated support for this process.

The final step is to estimate for a new project. It must be possible to characterize the project in terms of the variables that have been identified at the first stage of the estimation process. From these variables, ANGEL can be used to find similar projects and the user can make a subjective judgment as to the value of the analogies. The value of estimation by analogy as an independent source of prediction will be somewhat reduced if the users discount values that are not consistent with their prior beliefs and for this reason there was no expert intervention or manipulation in any of the foregoing analysis.

4.3.4. WORK DISTRIBUTION TECHNIQUE

Some types of software development project are repeated multiple times within organizations and similar activities are repeated every time a project is implemented. The work distribution technique relies on data gathered from similar past project, which it apportions across the standard project life cycle to arrive at the proportions of effort or budget required by each stage (Roberts 1997). The work distribution can be done in many different ways including distribution based on project phases or resources. Two examples of a work distribution in a software project are presented in figure below.

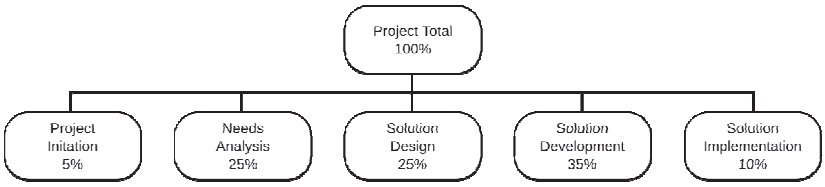

Figure 9: The work distribution technique (Roberts, 1997)

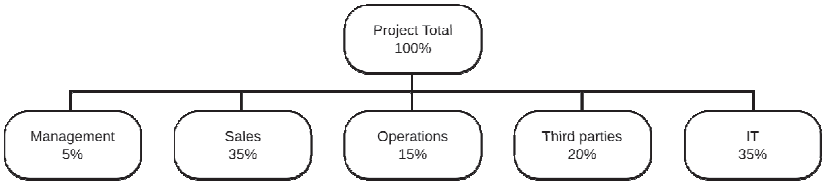

Figure 10: Spreading the proportions spent on projects across different types of resource (Roberts, 1997)

The success and accuracy of work distribution technique depends on the reliability of data or experiences from past projects and the projects themselves being similar. If data from past projects can be recorded and improved over time, the work distribution technique can be used quite flexibly and accurately. Many organizations and experts have also thumb rules for the work distribution in a software project. For example work distribution statistics from NASA (NASA, 2003) for new, modified and converted software are given in the table below.

Table 9: An example of work distribution statistics (NASA, 2003)

	Data		
Phase	New Software[11] %	Modify Existing Software %	Convert Software %
Requirements Analysis and Design	20%	15%	5%
Detail Design, Code and Unit Test	57%	10%	5%
SW Integration &Test	23%	40%	30%
Relative Effort	100%	65%	40%

4.3.5. TOP-DOWN

A top-down cost estimate for a project is derived from the global properties of the software product. The total cost is then split up among the various components or stages. (Boehm, 1981)

An example top-down estimate is shown in the figure below. Top-down estimating can be done in conjunction with other techniques.

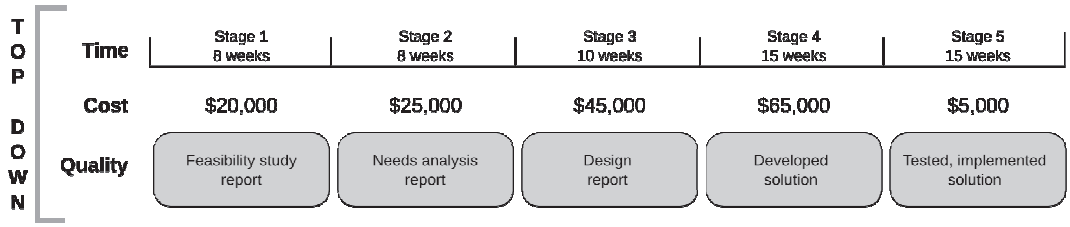

Figure 11: A top-down estimate (Roberts, 1997)

The main strength of top-down estimation is its system level focus. To the extent that the estimate is based on previous experience on entire completed projects, it will not miss the costs of system level functions such as integration, users' manuals, configuration management etc. (Boehm, 1981)

The major disadvantages of top-down estimating are that it often does not identify difficult low level technical problems that are likely to escalate costs; that it sometimes misses components of the software to be developed; that it provides no detailed basis for cost justification and iteration; and that it is less stable than a multicomponent estimate, in which estimation errors in the component have a chance to balance out (Wolverton, 1974).

4.3.6. BOTTOM-UP

Bottom-up estimation is complementary to top-down estimation. In the bottom-up estimation, the cost of each component or work product is estimated individually and then these costs are summed to arrive at an estimated cost for the overall software product (Boehm, 1981). Before the bottom-up estimation, the overall software product should be first decomposed into a set of smaller work products or components, for example using work breakdown structure.

An example Bottom-up estimate is shown in the figure below. Bottom-up estimating can also be done in conjunction with other techniques.

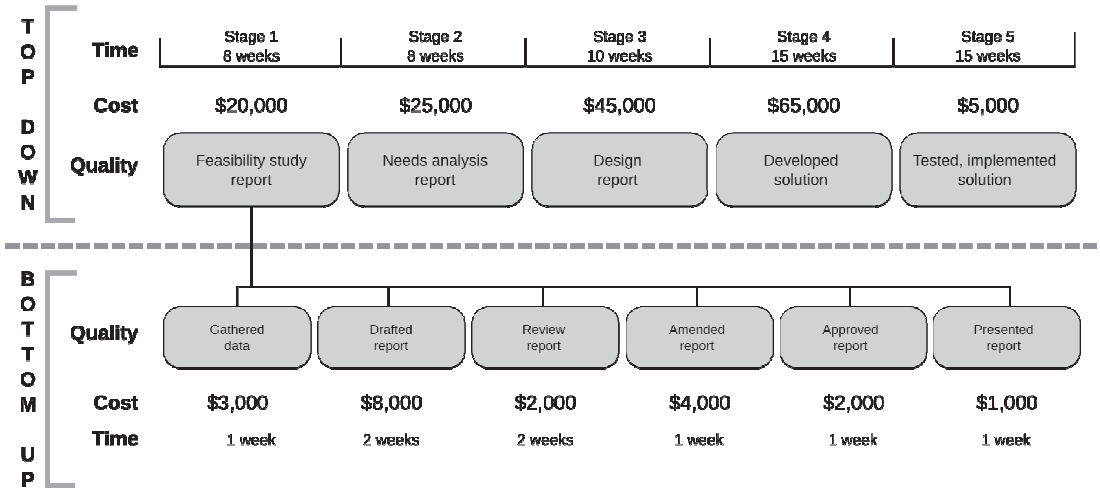

Figure 12: A top-down and bottom-up estimate (Roberts, 1997)

The strengths and weaknesses of bottom-up estimation are complementary to the top-down estimation. The bottom-up estimate tends to focus on the costs associated with developing individual components and hence it may overlook many of the system level and other project costs. It also requires more effort than does a top-down estimate, but it yields more stable estimates, since the estimation errors in the various components have a change to balance out. (Boehm, 1981)

4.4. LEARNING-BASED TECHNIQUES

Learning-based techniques use prior and current information to develop a software cost estimate. These techniques include Artificial Neural Networks and Case-Based Reasoning.

4.4.1. CASE-BASED REASONING

Case-Based Reasoning (CBR) is very similar to software cost estimation by analogy. Analogy has actually been identified as a simple form of Case-Based Reasoning (CBR) (Aamodt and Plaza, 1994). The CBR estimation technique contains usually four primary steps:

1. Retrieve the most similar case or cases, i.e., previously developed projects
2. Reuse the information and knowledge represented by the case(s) to solve the estimation problem
3. Revise the proposed solution
4. Retain the parts of this experience likely to be useful for future problem solving.

In the context of software cost estimation, a CBR model is based on the analogy assumption that similar software projects have similar costs. Initially, each software project (both historical and candidate projects) must be described by a set of attributes (environment conditions, constraints, technologies, decision, success, cost spent, effort used) that must be relevant and independent of each other. Subsequently, the similarity between the candidate project and each project in the historical database is determined. Finally, the known development effort values of historical (previously developed similar) projects is used to derive, i.e., case adaptation, an estimate for the new project. (Idri et al., 2002)

The source of case studies can be either internal or external to the estimator's own organization. Internal cases are likely to be more useful for the purposes of estimation because they will reflect the specific engineering and business practices likely to be applied to an organization's projects in the future, but well-documented cases studies from other organizations doing similar kinds of work can also prove very useful. (Boehm et al., 2000b).

The formal five-step process and software tool, ANGEL (ANaloGy Estimation tool2), developed by Shepperd and Schofield (Shepperd et al., 1996) can also be categorized as Case-Based Reasoning technique instead of a simple Analogy technique. The ANGEL was presented in section 4.2.3.

Idri and others have created a Case-Based Reasoning technique which is based on analogy-based reasoning, fuzzy logic, and linguistic quantifiers. In the Fuzzy Analogy approach, both linguistic and numerical data are represented by fuzzy sets. Furthermore, by using the linguistic quantifier to guide the aggregation of the individual similarities between two projects, the Fuzzy Analogy approach can easily be adapted and configured according to the requirements and specifications of each environment. (Idri et al., 2002)

4.4.2. NEURAL NETWORKS

Neural networks are based on the principle of learning from example; no prior information is specified. Neural networks are characterized in terms of three entities, the neurons, the interconnection structure and the learning algorithm (Karunanithi et al., 1992).

Most of the software models developed using neural networks use backpropogation trained feed-forward networks. The neural networks are created with an appropriate layout of neurons, or connections between network nodes. This includes defining the number of layers of neurons, the number of neurons within each layer, and the manner in which they are all linked. The weighted estimating functions between the nodes and the specific training algorithm to be used must also be determined. The network is then trained with a series of inputs and the correct output from the training data so as to minimize the prediction error. Training data should originate from past projects. Once the training is complete, and the appropriate weights for the network arcs have been determined, new inputs can be presented to the network to predict the corresponding estimate of the response variable. (Gray and MacDonell, 1997)

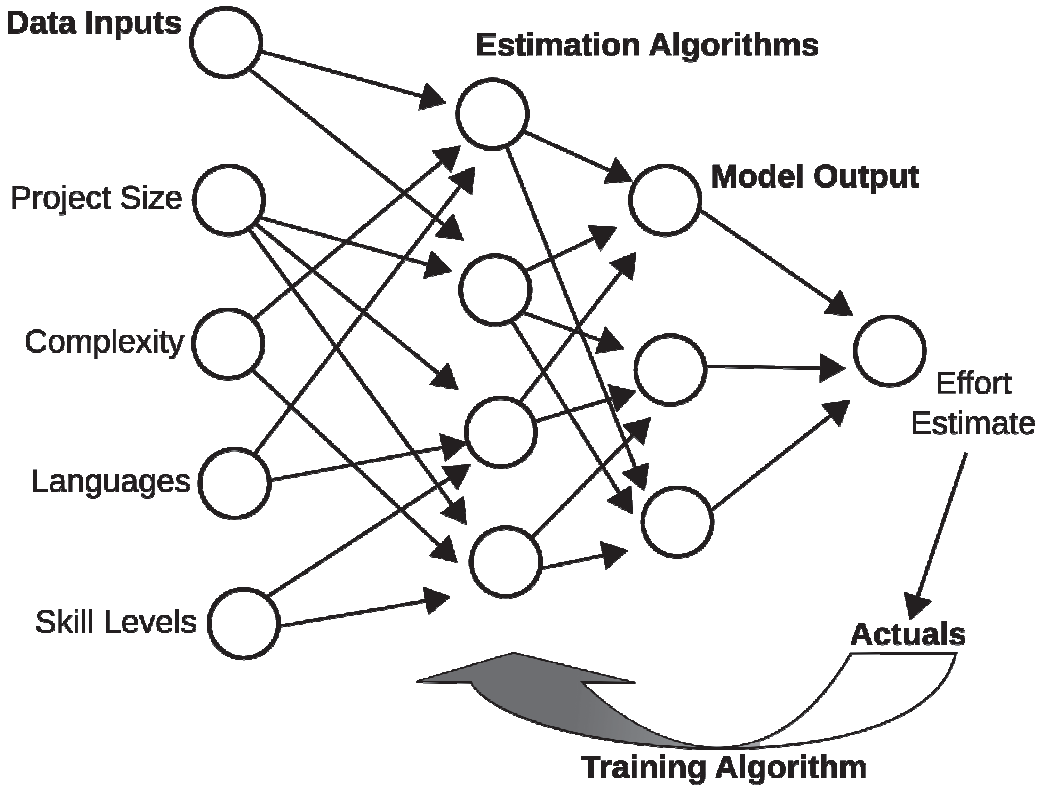

Figure 13: Neural network for software cost estimation (Boehm et al., 2000b)

Neural networks operate as 'black boxes' and they do not provide any information or reasoning about how the outputs are derived. And since software data is not well-behaved it is hard to know whether the well known relationships between parameters are satisfied with the neural network or not.

Wittig and Finne developed a software estimation model using neural networks and derived very high 29 prediction accuracies (Wittig and Finne, 1994). They also reported accuracies of within 10% for a model of this type when used to estimate software development effort (Wittig and Finne, 1997).

Neural networks are often subject to the same kinds of statistical problems with the training data as are the standard regression techniques used to calibrate more traditional models. In particular, extremely large data sets are needed to accurately train neural networks with intermediate structures of any complexity. Also, for negotiation and sensitivity analysis, the neural networks

provide little intuitive support for understanding the sensitivity relationships between cost factors and estimation results. (Gray and MacDonell, 1997)

4.5. DYNAMICS-BASED TECHNIQUES

Dynamics-based techniques acknowledge that software project effort or cost factors change over the duration of the software development process (Boehm et al., 2000b). Hence the effort and cost factors are dynamic rather than static with regards to time. This is a significant difference compared to many other techniques which use only static models and factors. In practice, cost and effort factors like functional requirements, project team, budgets, etc change during the software development process and hence they cause changes in the productivity of the project team. This has consequences, which are quite often negative, in the project schedule, quality and costs.

Dynamics-based techniques are often based on the system dynamics originated from Jay Forrester (Forrester, 1961). System dynamics is a methodology and computer simulation modeling technique for framing, understanding, and discussing complex system, issues and problems. In system dynamics, models are represented as networks modified with positive and negative feedback loops, stocks, flows, and information that changes over time and dynamically affects the flow rates between the feedback loops. Below figure from Madachy (Madachy, 1994) shows an example of a system dynamics model demonstrating the Brooks' Law. The Brooks' Law states that adding manpower to a late software project makes it even later (Brooks, 1975).

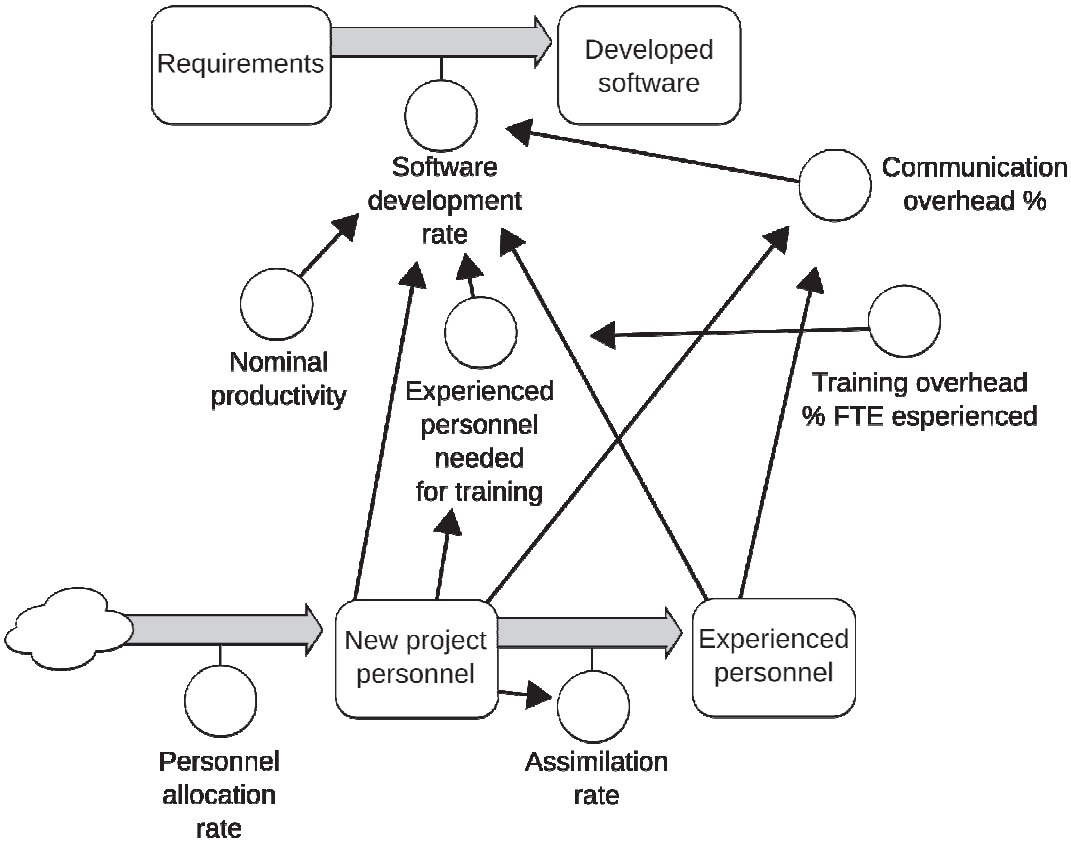

Figure 14: Madachy's System Dynamics Model of Software development (Madachy, 1994)

The system dynamics approach involves the following concepts (Richardson et al., 1991):

1. Defining problems dynamically over time
2. Striving for an endogenous, behavioral view of the significant dynamics of a system
3. Thinking of all real systems concepts as continuous quantities interconnected in information feedback loops and circular causality
4. Identifying independent levels in the system and their inflow and outflow rates
5. Formulating a model capable of reproducing the dynamic problem of concern by itself
6. Deriving understandings and applicable policy insights from the resulting model
7. Implementing changes resulting from model-based understandings and insights.

Mathematically, system dynamics simulation models are represented by a set of first order differential equations (Madachy, 1994):

$$x'(t) = f(x, p),$$

where

x = a vector describing the levels (states) in the model

p = a set of model parameters

f = a nonlinear vector function

t = time

The system dynamics technique has been quite successfully applied to software engineering estimation models by Abdel-Hamid and Madnick (Abdel-Hamid and Madnick, 1991). They have built models that will predict changes in project cost, staffing needs and schedules over time, as long as the initial proper values of project development are available to the estimator. In addition, they have found out initially beneficial relationship between the reuse of software components and project personnel productivity, since less effort is being spent developing new code (Abdel-Hamid and Madnick, 1993).

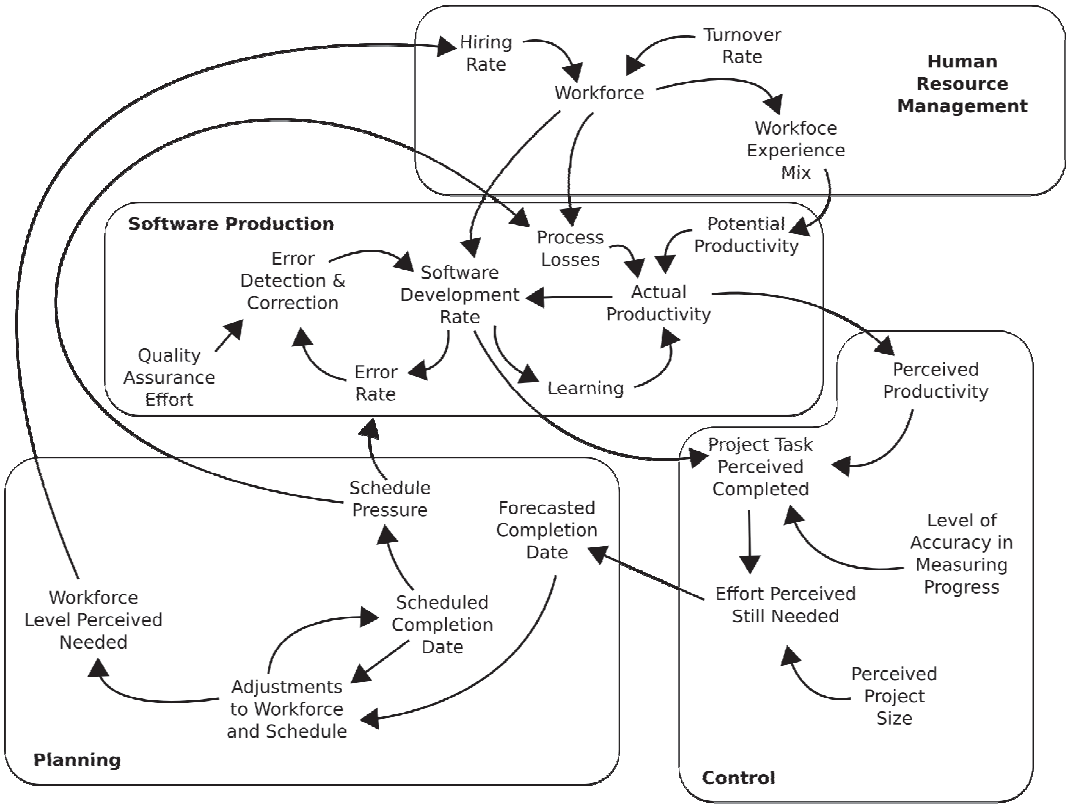

Figure 15: System dynamics model for software cost estimation (Abdel-Hamid and Madnick, 1991).

4.6. REGRESSION-BASED TECHNIQUES

Regression-based techniques provide one or more mathematical algorithms which produce a software cost estimate as a function of number of variables which are considered to be the major cost driver. The pure regression techniques include ordinary least squares method and Robust regression.

4.6.1. ORDINARY LEAST SQUARE TECHNIQUE

Ordinary Least Squares (OLS) technique is a classical mathematical method for estimating the unknown parameters using a linear regression model. It is presented in most of the advanced statistical and econometric books (for example: Keating, 1985). It is one of the common techniques used in the research literature due to its simplicity and easy accessibility from statistical software packages (Gray and MacDonell, 1997).

The Ordinary Least Squares method aims to find the best linear model that fits the dataset by minimizing the sum of squared residuals. Residuals are the distance between the observed in the dataset, and the responses predicted by the linear approximation.

A simple data set consists of n points (data pairs) (x_i,y_i), i = 1, ..., n, where x_i is an independent variable and y_i is a dependent variable whose value is found by observations (for example past project data). The model function has the form $f(x,\beta)$, where the m adjustable parameters are held in the vector β. The goal is to find the parameter values for the model which "best" fits the data. The least squares method finds its optimum when the sum, S, of squared residuals is at its minimum:

$$\min S = \sum_{i=1}^{n} r_i^2$$

The residuals r_i is defined as the difference between the value predicted by the model and the actual value of the dependent variable:

$$r_i = y_i - f(x_i, \beta)$$

Denoting the intercept as β_0 and the slope as β_1, the linear model function is given as:

$$f(x, \beta) = \beta_0 + \beta_1 x$$

The values for m adjustable parameters ($\beta_{optimum}$), which minimize the sum of squared residuals, can be found through simple first order derivation. Once the optimum valus are found, the model function, $f(x, \beta_{optimum})$, can be used for estimation.

There are also other variations from the Ordinary Least Squares method. These include Weighted Least Squares and Non-linear Least Squares methods. The Weighted Least Squares method aims to minimize the weighted sum of squared residuals. Hence it allows having different weights for the different cost factors which are represented by the adjustable parameters in the model. In Non-linear Least Squares method the data is fitted to a non-linear model function $f(x, \beta)$.

Least squares regression is well suited for use in situations (Gray and MacDonell, 1997):

- Many degrees of freedom are available i.e. there are many more observations than parameters to be estimated
- The data is well-behaved (in the statistical sense, for example there are no outliers or significant heteroscedasticity)
- A small number of independent variables are sufficient, after transformations if necessary, to linearly predict the possibly transformed output variable(s) so as to enable an interpretable representation
- There is no missing data.

These place a severe restriction on the use of this technique for software engineering data sets that rarely meet all of these conditions. In software engineering, collecting data is often difficult due to lack of funding by higher management, coexistence of several development processes, and lack of proper interpretation of the process. Extreme cases are also very often reported and many variables are correlated with each other. In addition, there is often data missing or incorrectly gathered from the past projects. (Boehm et al., 2000b)

4.6.2. ROBUST REGRESSION

Robust Regression is an improvement of Ordinary Least Squares (OLS) method which is not impact by extreme values often found in software engineering data sets. The main difference in robust regression compared to OLS is the residual measure (error term). In Robust Regression it is usually the mean median error, but it can also be other available measures of error or on some middle portion of the errors (for example, ignoring the top and bottom five percent of errors).

The use of robust regression can be especially attractive in software development data sets since they can be very small, and therefore extremely sensitive to the abnormal observations they contain, and often contain errors in measurement. Software project data can also have a lot of outliers due to disagreement on the definitions of software metrics, coexistence of several software development processes and the availability of qualitative versus quantitative data. (Gray and MacDonell, 1997)

An example of Robust Regression method is the Least Median Squares method. It is very similar to the OLS method described in the previous section. The only difference is that the method minimizes the median of all residuals:

$$\min_{\theta} median_{i} r_i^2$$

Least Median Squares regression provides estimates that cannot be affected to an arbitrary degree by up to 50 percent contamination (i.e. data values that do not reflect the underlying system being modeled for reasons that may include measurement error and an unusual system). This compares to least squares regression's estimates which can be arbitrarily affected by a single outlying observation. Example difference between Least Media Squares and Ordinary Least Squares method is presented in picture below (Gray and MacDonell, 1997)

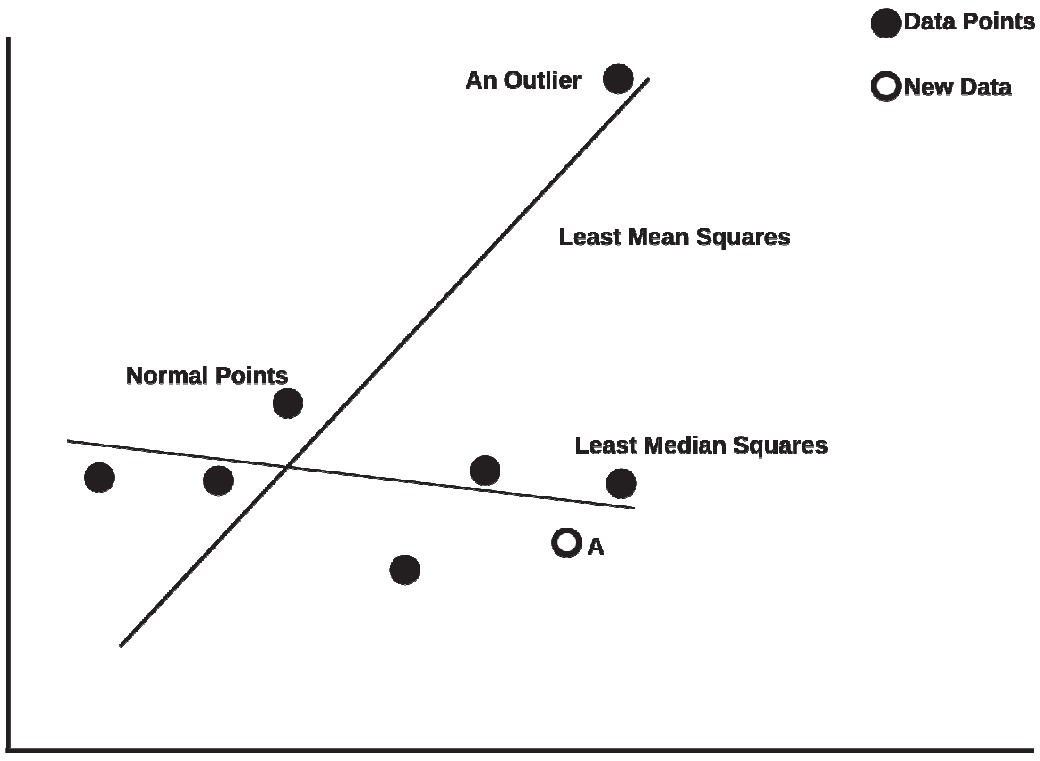

Figure 16: Difference of Least Mean Squares and Least Median Squares methods (Gray and MacDonell, 1997)

4.7. COMPOSITE TECHNIQUES

Composite techniques incorporate a combination of two or more techniques to formulate the most appropriate functional form for estimation.

4.7.1. BAYESIAN APPROACH

One of these composite techniques is the Bayesian approach (Chulani et al., 1999) which combines expertise-based and model-based (COCOMOII) techniques. In the Bayesian approach, the estimation is done by combining the sample (historical project data) and prior (expert-judgment) information. By using Bayes' theorem, the prior information is transformed to post-data or posterior view:

$$f(\beta|Y) = \frac{f(Y|\beta)f(\beta)}{f(Y)}$$

, where ß is the vector of parameters to be clarified and Y is the vector of sample observations from the joint density function f (β|Y). f (β |Y) is the posterior density function for ß summarizing all the information about ß, f (Y| ß) is the sample information and is algebraically equivalent to the likelihood function for ß, and f (b) is the prior information summarizing the expert-judgment information about ß. This transformation process can be viewed as a learning process.

The Bayesian approach described above can be used in the calibration of COCOMOII to improve its prediction accuracy. With the Bayesian approach, the COCOMOII can be calibrated to produce estimates within 30% of the actual 75% of the time compared to the earlier 52% of time (Chulani et al., 1999).

The Bayesian approach has similar advantages as "Standard" regression, but it also includes prior knowledge of experts. It aims to reduce the risks associated with imperfect data gathering which is a usual problem with software engineering data area. Usually a lot of good expert judgment based information is available on software processes and the impact of several parameters on effort, cost, schedule, quality etc.

4.7.2. COST ESTIMATION, BENCHMARKING, AND RISK ANALYSIS

The Cost Estimation, Benchmarking, and Risk Analysis method (CoBRA) was developed by Fraunhofer IESE (Institut für Experimentelles Software Engineering) in 1996 and it has been applied in several studies (Briand et al., 1998, Ruhe et al., 2003, Trendowicz et al., 2006). It combines expert-driven and data-driven cost estimation methods in order to provide a technique that can be used in organizations where only little data is available.

The basic formula used in CoBRA is following:

$$COST = NOMINALPRODUCTIVITY \times SIZE + COSTOVERHEAD$$

The nominal productivity determines the productivity in an optimal case (~in a project in which all cost factors are at optimal values). The cost overhead describes additional costs that occur in projects that are not done in an optimal environment. Thus the overhead is driven by set of influencing factors (e.g. team capability, requirements stability, etc). The overhead factors are organized into a causal model that determines their impact on project costs. An example of such causal model is shown in figure below.

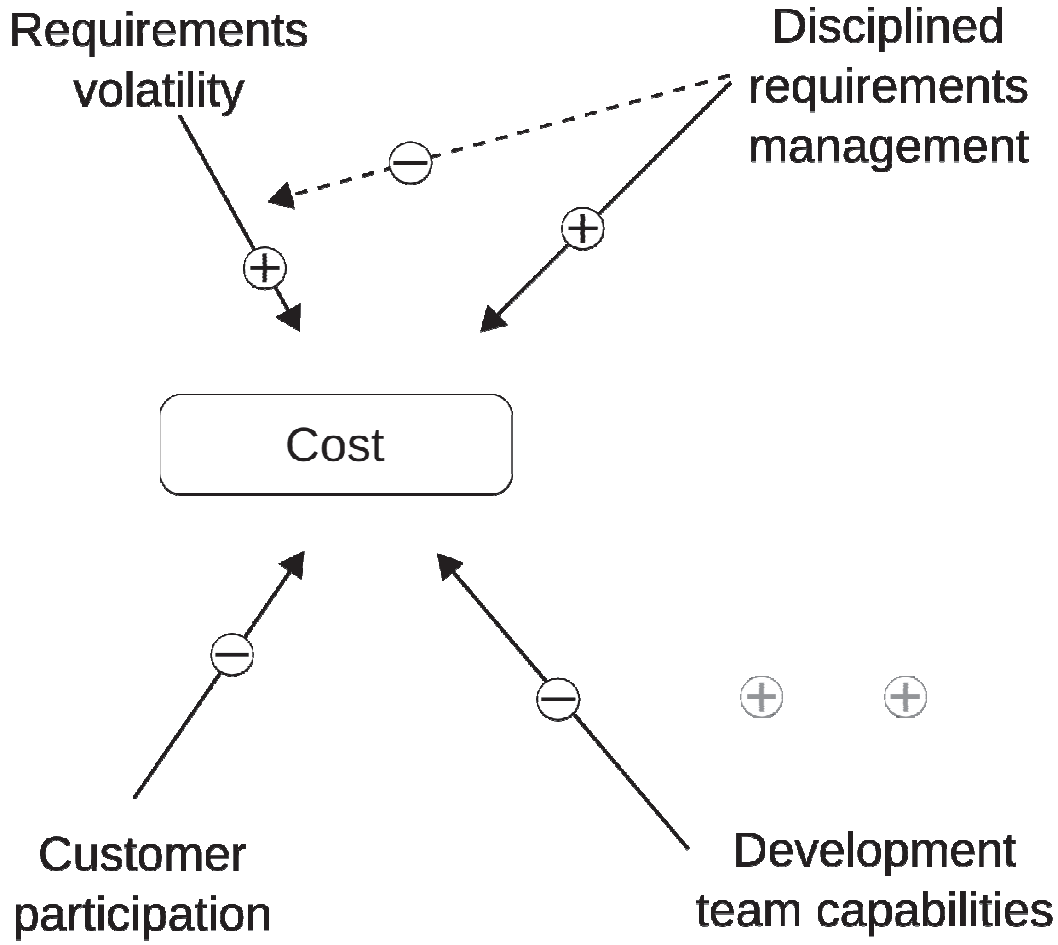

Figure 17: A causal model for software cost estimation (Lamersdorf et al., 2010)

The cost factors in a causal model can have a positive or negative impact in two ways. Direct relationships describe direct impact on costs overhead, which means that an increase of these factors will directly increase cost overhead. Indirect relationships determine the impact of another factor on cost overhead. Some factors can have both a direct and indirect impact on cost overhead.

In CoBRA, the development and quantification of the causal model is done by experts. Selection of the factors and determination of their causal relationships is done in group discussions. The nominal productivity is computed using data from past projects.

The CoBRA process can be summarized as follows:

1. Collect possible cost factors: A set of possible factors influencing cost overhead is collected

2. Rank and select cost factors: Each expert ranks the cost factors individually according to their impact on cost overhead. Ranking can be done within factor categories. The aggregated results of the ranking are presented to the experts, and the most important cost factors are selected in a group discussion

3. Build causal model: In another group discussion among the experts, the direct and indirect influences of the factors on cost overhead are determined and a causal model is developed

4. Quantify causal relationships: Based on the causal model, a questionnaire is prepared asking each expert to quantify the impact of the factors on cost overhead. In every judgment, each expert is asked to name minimum, most likely, and maximum overhead in percent over a nominal case

5. Analyze past projects: for a set of past projects, size and effort data is collected and the experts are asked to characterize them with respect to the cost overhead drivers. Based on these results, the nominal productivity is determined.

5. GLOBAL SOFTWARE DEVELOPMENT

5.1. OVERVIEW

Global software development (GSD) refers to software development that is done by multiple teams in different geographic locations. The teams are separated physically and they are located in different countries within one region or around the world. The teams can be from one organization or from multiple different organizations for example in outsourcing or collaboration scenarios.

For example a Finnish company can use software development teams located in headquarters in Finland and in a subsidiary in China. The same company could also outsource some parts of the software development directly from a company in India.

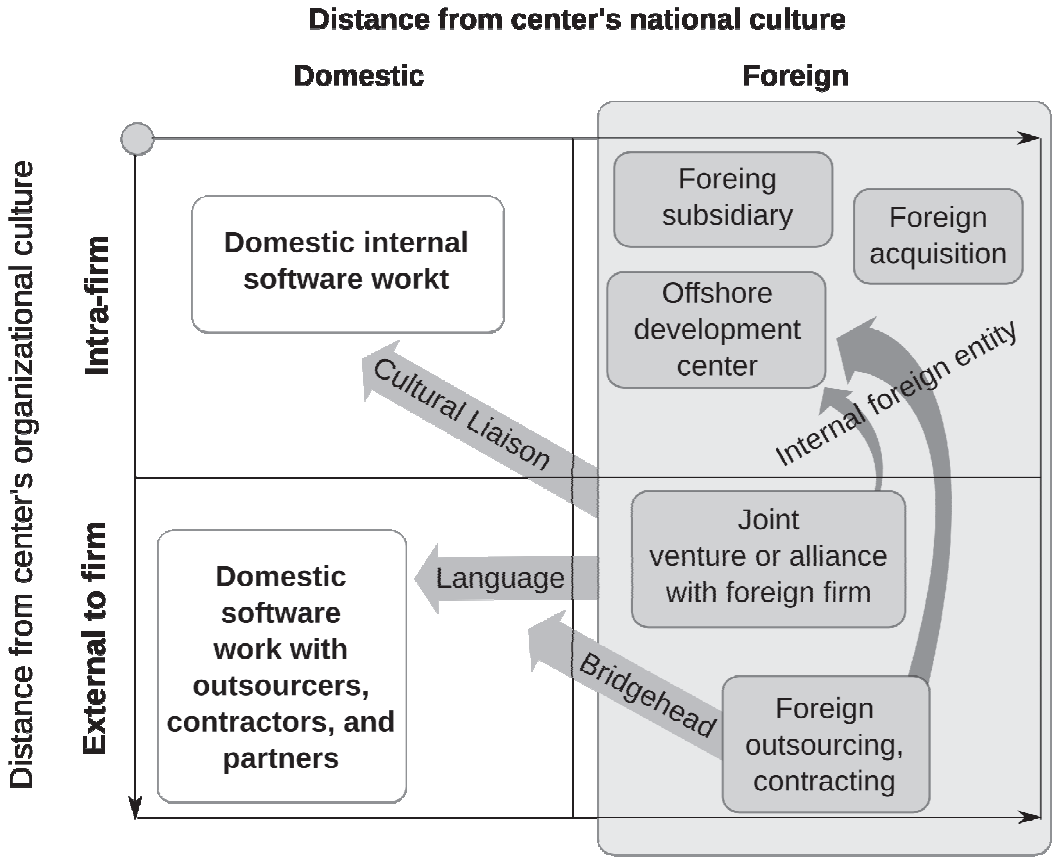

Figure 18: A taxonomy of structural arrangements for software development (Carmel and Agarwal, 2001)

Global software development is usually considered to be much more difficult than local software development due to the many different challenges associated with the development of software in a globally distributed setting. These challenges include negative impact of physical distance, cultural differences and many other complexity factors which are elaborated in the following subsections.

Past studies have shown that tasks take about 2.5 times longer to complete in a distributed setting versus a collocated setting (Herbsleb et al., 2001). Other studies reported that about 40 percent of GSD projects fail to deliver the expected benefits, due to the lack of theoretical basics and difficult complications in GSD project (Betz and Mäkiö, 2007). On the other hand, Stephanie Teasley and her colleagues reported that in colocated teams, productivity and job satisfaction are much higher (Teasley et al., 2002).

The additional activities and difficulties in global software development require additional effort for substantial planning, coordination and control overhead in the day-to-day governance of global software development. This additional effort should be considered in the effort and cost estimation. Hence the effort and cost estimation in global software development (GSD) is more complex than in local development.

5.2. OVERHEAD FACTORS

There are many challenges associated with global software development. Physical separation among project members has diverse effects on many levels. The following factors have been gathered from research literature (Noll et al., 2010) to have an impact on the amount of effort and cost required for global software development:

- Geographic distance
- Temporal dispersion due to being located in different time zones
- Differences in language and culture
- Social factors such as fear and trust
- Problems stemming from organizational structure
- Process issues
- Barriers deriving from infrastructure
- Barriers due to product architecture.

The above overhead factors are discussed in more details in the following sub-sections.

The below table summarizes a review of the 26 research papers on collaboration barriers in global software development (Noll et al., 2010). These results indicate that the most of the barriers are widely analyzed in the research literature except for product architecture which has been discussed only in few researches.

Table 10: Identified barriers for global software development (Noll et al., 2010)

Barrier	No. Papers	Per cent of Papers
Language and Cultural Distance	14	53,85 %
Temporal Distance	13	50,00 %
Geographic Distance	12	46,15 %
Process and Management Issues	11	42,31 %
Social Issues	9	34,62 %
Infrastructure	7	26,92 %
Organization	6	23,08 %
Product Archicture	3	11,54 %

Perhaps more critical to global software development is the observation that the overhead factors can have a compounding effect on each other, increasing the possibility of a negative impact on the development process (Noll et al., 2010). For example knowledge transfer will not occur smoothly unless a team spirit exists between the individuals or communication difficulties arise if roles and responsibilities have not been explicitly defined.

The barriers identified commonly in the research do not include factors which rise outside of software engineering. These include factors like set-up factors (contracting to a foreign country, founding an office in foreign country), illegal (corruption in many developing countries) and international trade related factors (export and import taxes and tariffs). These factors might also have significant impact on the overall costs of a software development project.

5.2.1. GEOGRAPHIC DISTANCE

Software development, particularly in the early stages, requires much communication (Perry et al., 1994). Kraut and Streeter (Kraut and Streeter, 1995) found that formal communication (e.g., structured meetings, specifications, inspections) is useful for routine coordination while informal

communication (e.g., hallway conversations, telephone calls, workshops) is needed in the face of uncertainty and unanticipated problems, which are typical of software development. They observed that the need for informal communication increases dramatically as the size and complexity of the software increases (Kraut and Streeter, 1995). In a large software organization, developers can spend on average up to 75 minutes per day for informal unplanned communication (Perry et al., 1994). Hence it seems that much of the vital coordination, communication and interaction in a software development project occurs in an informal "ad hoc" manner.

It appears that the ability to be able to discuss face to face and work closely improves the development of a shared understanding on the software being developed. This shared understanding is required indicating the level of completeness of documentation and specification and the common knowledge about goals (McChesney and Gallagher, 2004).

Shared understanding also increases productive and quality in software development. As a result of numerous informal interactions and discussions, team members are much more aware of the details of pieces of software developed, and are much more likely to communicate with those who work on them.

Geographic distance introduces many barriers to communication and collaboration, the most immediate being the lack of informal encounters that provide not only the opportunity to exchange implicit knowledge and build shared understanding, but also to develop personal relationships and avoid conflict, misalignments and rework. (Herbsleb and Grinter, 1999, Herbsleb et al., 2001, Carmel and Agarwal, 2001). In addition, geographic distance also increases organizational complexity (Carmel and Agarwal, 2001) and reduces the frequency of communication across all media (Sosa et al., 2002).

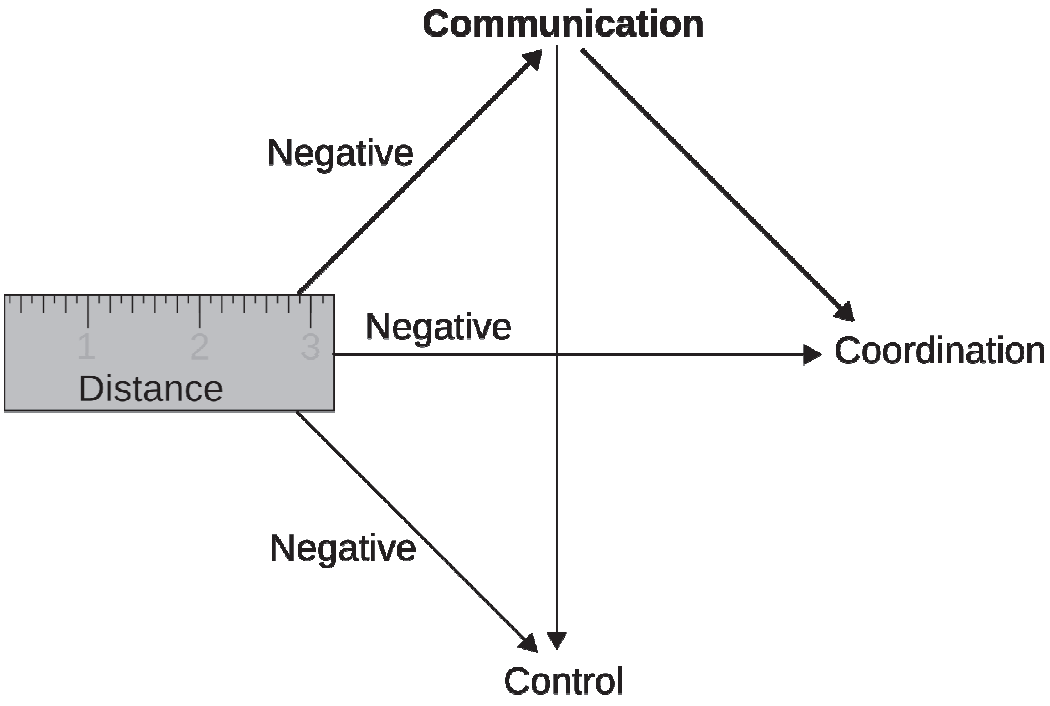

Figure 19: Impact of distance (Carmel and Agarwal, 2001)

Herbsleb and others (Herbsleb et al., 2001) reported in their research that the absence of ongoing conversations leads to surprises from distant sites, and results in misalignment and rework, which impacted coordination and control. It became difficult to synchronize the activities of all the teams because of the inability to obtain accurate effort and status information.

The physical separation requires additional activities and effort in software development especially across geographic distances, but even a short physical separation can have an impact as well. In a study of engineering organizations, Tom Allen (Allen, 1984) reported that the frequency of communication among engineers decreased with distance. Furthermore; he noticed that when engineers' offices were about 30 meters or more apart, the frequency of communication dropped to nearly the same low level as people with offices separated by many miles.

To reduce these risks related to geographic distance, many organizations increase the travel between different sites, which then increase other costs such as travel costs.

5.2.2. TEMPORAL DISPERSION

Second frequently identified overhead factor for global software development is the temporal dispersion. It refers to the time zone differences among project members, when development team is distributed around the world.

Temporal dispersion reduces the possibilities of synchronous interaction, which is a critical communicational attribute for real-time problem solving and design activities. In practice, the teams in different time zones have fewer hours in the work day when multiple sites can participate in a joint synchronous meetings and discussions. Temporal dispersion also makes misunderstandings and errors significantly more likely to happen. (Espinosa et al., 2007)

This leads to delay in response to asynchronous communication. For example an e-mail sent from one site arrives after working hours at the destination; as a consequence, the response cannot be sent until the next day begins and it will be visible to the sender only when he/she comes to office on the following day.

Herbsleb and colleagues observed a mean delay in receiving a response to an inquiry in a distributed development context that was over two-and-a-half times as long as that seen in a single-site context (Herbsleb and Mockus, 2003). Similarly, they observed that "modification requests" (requests for changes to a software component) took over twice as long to be completed in a distributed context (Herbsleb et al., 2005).

Espinosa and his colleagues have studied the effect on temporal dispersion further and they found that the a small time separation has no effect on accuracy, but that more time separation has a significant effect on accuracy. In addition, they found that a small amount of time separation has a significant effect on production speed, but surprisingly further increases in partial overlap have less significant effects on speed, and when there is no overlap speed actually increases, hence time difference had a "U-shaped" effect on productivity. (Espinosa et al., 2007)

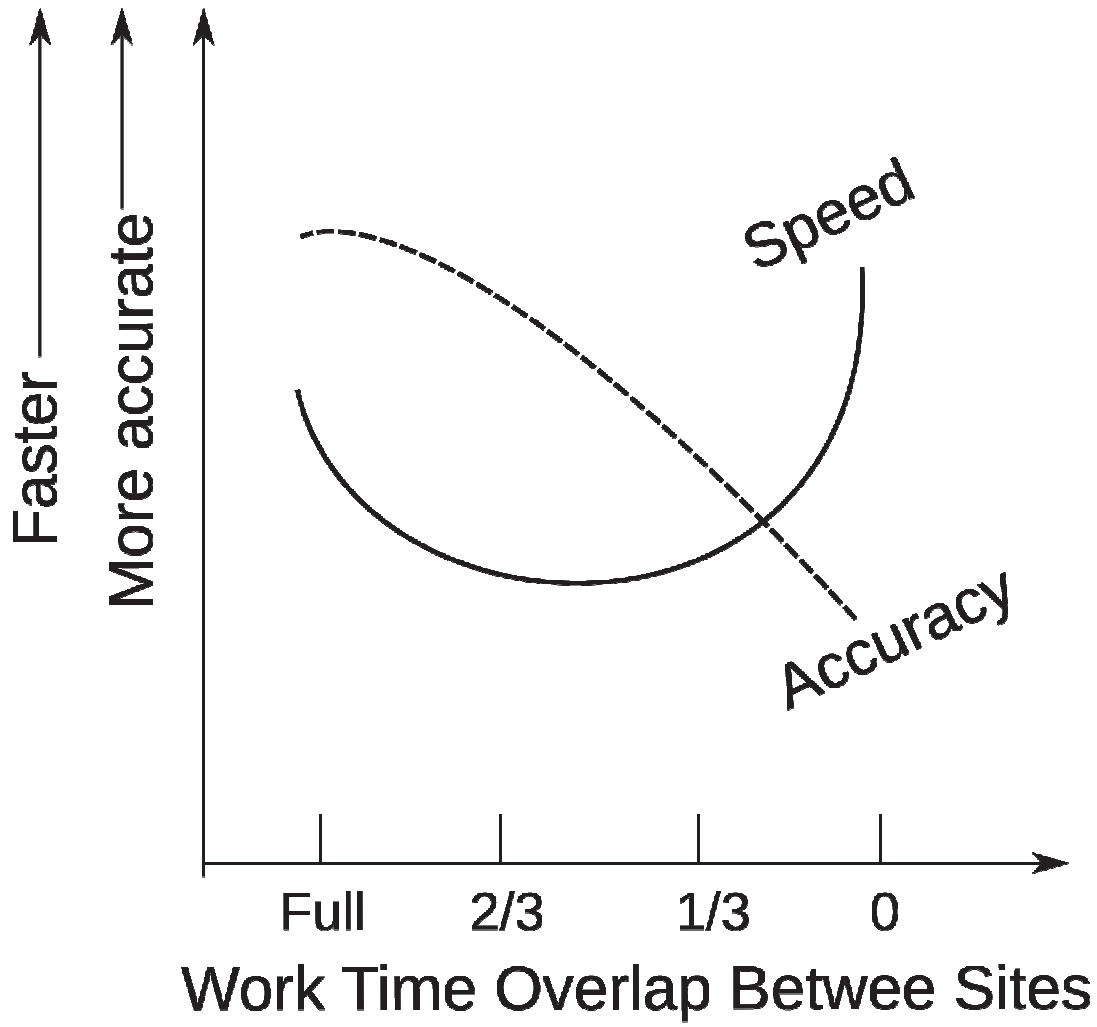

Figure 20: The time zone effects on speed and accuracy of communication (Espinosa et al., 2007)

The above results suggest that the frustrations experienced by team members when working across time zones may not necessarily translate into actual declines in performance levels. On the contrary, the effect of time separation on speed could be explained with the fact that when the tasks are relatively simple, teams can work efficiently and uninterrupted when they have no overlap and can articulate and interpret instructions clearly. However, when teams need to constantly change their interaction mode from synchronous to asynchronous they need to make mental adjustments in their work styles and coordination tactics, thus reducing their efficiency. (Espinosa et al., 2007)

5.2.3. CULTURAL DIFFERENCES

Global software development requires close cooperation of individuals with different cultural backgrounds which often creates another barrier for efficient work. Cultures differ on many critical dimensions, such as the need for structure, attitudes toward hierarchy, sense of time, and communication styles (Hofstede, 1997). These differences have been recognized as major barriers to communication (Leonard et al., 1997, Ebert and De Neve, 2001, Herbsleb et al., 2005). Culture also affects interpretation of requirements; domain knowledge used to fill in gaps or place requirements in context varies considerably across national culture (Herbsleb et al., 2005). The cultural difference manifests itself in one of two forms: organizational culture and national culture (Carmel and Agarwal, 2001).

While many people find such cultural differences enriching, these can lead to serious and frequent misunderstandings and communication problems, especially among people who do not know each other well. For example, an e-mail from someone in a culture where communication tends to be direct might seem abrupt or even rude to someone from a different background. Collectivist cultures, such as Asian countries, find even the telephone as less rich communication medium than the business memo, which is contrary to the rating provided by respondents from "individualist" countries, such as Western countries (Rice et al., 1998).

Culture also interferes with collaboration when cultural norms result in conflicting approaches to problem solving (Herbsleb et al., 2005). For example, American and European engineers were observed to have different views on the value of "up front" design, with Europeans tending toward more initial design effort, and Americans preferring to proceed quickly to implementation; the difference in approach caused conflict and negative impressions. A different sense of time can lead to acrimony over the interpretation and seriousness of deadlines. Polite expressions of acknowledgement by Asian engineers could be misinterpreted as agreement or commitment by their European and American colleagues.

Organizational culture encompasses the unit's norms and values, where the unit could range from a small technology company to a multinational enterprise. Organizational culture includes the culture of systems development, such as the use of methodologies and project management practices. Differences in "corporate culture" can lead to conflicting approaches to problem

solving and communication, which in turn might be misinterpreted as rudeness or incompetence even when teams share a common language and nationality (Herbsleb et al., 2005).

On the other hand cultural differences can have a positive impact as well. A major positive effect is innovation. Engineers with all types of cultural backgrounds actively participate to continuously improve the product, innovate new products, and make processes more effective. Achievements are substantial if engineers of entirely different educations and cultures try to solve problems. Best practices can be shared, and sometimes small changes within the global development community can have big positive effects. (Ebert and De Neve, 2001)

5.2.4. LANGUAGE DIFFERENCES

The lack of a common native language (known as "linguistic distance") creates further barriers to communication (Herbsleb and Grinter, 1999, Niinimäki et al., 2009). Linguistic distance limits the ability for coherent communication to take place (Casey and Richardson, 2006).

Language differences do not only affect the quality of communication, but the choice of communication media and style. The lack of proficiency in the chosen language can lead to a preference for asynchronous communication, which can be an impediment if video and teleconferencing are important communication media. (Niinimäki et al., 2009). In cases where people considered their language skills inadequate, they also preferred to use text-based medium for communication. Using text based communication medium allowed them to have some extra time to formulate their messages, even if they were using near-synchronous instant messaging as the communication tool. But text-based media do not convey the visual or auditory queues that convey important information such as how well a participant truly understands a conversation (Sosa et al., 2002, Niinimäki et al., 2009).

The probability of using written-asynchronous communication media, such as e-mail, rather than verbal-synchronous communication media, such as telephone, increases with cultural and language difference. Team members located in countries that do not share the same first language show higher probability of using e-mail communication than telephone communication (Sosa et al., 2002). They also find e-mail much more effective way. As one described it, "it's

hard to explain something to someone you don't know in your second language." (Herbsleb and Grinter, 1999).

Language skills can impede communication in more subtle ways. When parties to a conversation have different levels of proficiency the stronger party occupies a more powerful position and can appear to be more powerful, and thus suppress important communication through unintended intimidation (Lutz, 2009), (Niinimäki et al., 2009). This can cause conversations to become frequently very emotional, and take a great deal of time and energy (Herbsleb and Grinter, 1999).

5.2.5. SOCIAL FACTORS

Another fundamental challenge in global software development is the social factors like fear and trust. Fear and distrust can negatively impact the motivation, desire to work with, trust, cooperate, communicate and share knowledge with remove colleagues and hence it has a direct bearing on the success of implementing global software development (Casey and Richardson, 2008).

It is very difficult for individuals and groups to trust and build relationships with people who they fear are ultimately going to replace their jobs. On-site teams in expensive countries, are fearful of their job security when off-site teams are added in less expensive locations, this creates to mistrust to their off-site colleagues as well as their own management's motives. This can result in clear examples of not wanting to cooperate and share knowledge with remote colleagues (Casey and Richardson, 2008, Piri et al., 2009).

In some cases, people who have successfully worked together for up to year in a collocated situation, once a virtual team strategy was fully implemented these problems soon came to the fore (Casey and Richardson, 2008).

The organization's resistance to global software development often surfaces because of misalignment between senior and middle management on the intent and perceived benefits of global software development. Many individuals might believe their jobs are threatened, experience a loss of control, and fear the possibility of relocation and the need for extensive travel (Herbsleb and Moitra, 2001).

Geographic, temporal, and cultural distances have a significant impact on trust among globally distributed team members. The main sources of distrust derive mainly from the rather weak interpersonal relations between employees at onsite and offsite, more specifically the lack of informal and interaction face-to-face means team members have less opportunity to form personal relationships and develop emotional bonds that improve trust among individuals. Also the lack of or short history of collaboration prevents developing trust. (Piri et al., 2009)

These issues are even more complex in outsourcing, collaboration and partnership arrangements. The fear of loss of intellectual property or other proprietary information about products or schedules leads to restricted or filtered communication, often seriously impairing this critical channel (Herbsleb and Moitra, 2001).

5.2.6. ORGANIZATION, MANAGEMENT AND PROCESS ISSUES

Organizational, management and process issues are related to when, by whom and how should development activities be performed and tasks allocated. Once a particular set of project sites has been determined, deciding how to divide up and execute the work across sites is very difficult. Solutions are constrained by the resources and tools available at the sites, their levels of expertise in various technologies, the infrastructure, and so on.

The work organization highly impacts globally distributed software development. Ebert and De Neve (2001) reported that collocated teams achieved an efficiency improvement during initial validation activities of over 50 percent in global software development. They also found that allocation directly impacts overall project efficiency as small projects with highly scattered resources showed less than half the productivity compared to projects with fully allocated staff. Cycle time is similarly impacted—people switching between tasks need time 2 to 3 times the original effort to adjust to the new job.

Collocation means that engineers working on such a set of coherent functionality should sit in the same building, perhaps within the same room. Full allocation implies that engineers working on a project should not be distracted by different tasks in other projects. Coherence means splitting the work during development according to feature content and assembling a team that can implement a set of related functionality.

Distribution development may split the groups with decision-making and project-execution power, and the group with the knowledge of the stakeholder needs. Hence distributed teams require a more rigorous requirements change-management process (which increases overhead), as the information-handoff points and separation between decision makers increase (Bhat et al., 2006).

Lack of implicit knowledge resulting from limited informal communication means processes like change management, if not explicitly and thoroughly defined, can be applied differently at different sites. Lack of informal communication also limits process visibility, leading to misunderstandings and frustration on the part of remote teams (Bhat et al., 2006).

Distributed development also introduces the possibility of different (and incompatible) processes and tools at various sites. Having multiple processes, tools, templates, and methodologies that don't integrate or interoperate can lead to wasteful rework or loss of data during transfer from one tool to another, which can decreases quality (Bhat et al., 2006).

Organizational structure also establishes boundaries within the organization. People within such boundaries are subjected to organizational bonds which promote the development of a language and an identity inherent to the group. Allen (Allen, 1984) found that organizational bonds increased the probability of two team members engaging in technical communication.

When analyzing the effects of organizational bonds on communication, Sosa and others (Sosa et al. 2002) found that both telephone and e-mail communication frequencies are much more sensitive to the presence of strong organizational bonds. Additionally, organizational bonds explain a greater portion of variation of telephone and e-mail communications than they do for face-to-face communications. Both interdependence (as measured by the importance of the task-related relationship) and organizational bonds were positively correlated with communication frequency across all media (Sosa et al., 2002). Even when team members were not collocated, higher communication frequencies were observed for highly interdependent pairs.

Project management and processes also create barriers. The lack of synchronization, especially the lack of commonly defined milestones and clear entry and exit criteria across work assignment units, can cause a lot of difficulties in the global software development (Herbsleb and

Moitra, 2001). The complex, usually uncertain, and highly interdependent nature of project tasks, together with geographical, temporal, structural and cultural gaps fundamental to distributed teams, make management of virtual projects a relatively complex undertaking (Paré and Dubé, 1999).

5.2.7. BARRIERS DUE TO INFRASTRUCTURE

Geographic distance creates communication barriers due to information technology setup and differences (such as different availability and accessibility schemes, different levels of familiarity with the systems, and incompatible information systems) which need to be overcome to facilitate electronic information transfer between interdependent team members. Inadequate knowledge management infrastructure can inhibit the formation of a shared understanding among project teams (Bruegge et al., 2006).

Distributed teams must rely on video and teleconferencing infrastructure to communicate; even intermittent failures of these technologies can mean that communication simply does not take place and in addition, it takes time and effort to resolve the issue with the communication systems (Taweel et al., 2009). Tools and infrastructure used in all sites require often training as some users might not be familiar with these systems (Taweel et al., 2009). There might also be potential ramp-up and operational costs for setting up the infrastructure in remote sites (Keil et al., 2006).

Since networks spanning globally dispersed locations are often slow and unreliable, tasks such as configuration management that involve transmission of critical data and multisite production must be meticulously planned and executed (Herbsleb and Moitra, 2001).

A project with multiple distributed teams introduces the possibility of multiple data repositories which need to be integrated. There is also a risk of having incompatible data repositories, which can increase risk of data loss or interoperability issues (Bhat et al., 2006), and create additional effort in data conversion and transfer due to incompatible data formats (Herbsleb and Moitra, 2001).

The lack of effective information sharing mechanisms, poorly maintained documentation, and lack of collaboration on artifacts all exacerbate the issue of managing knowledge in a way so as

to harness the true potential of global software development. This was evident from the difficulties experienced by the remote teams in understanding the requirements and the architecture (Herbsleb et al., 2001)

5.2.8. BARRIERS DUE TO PRODUCT ARCHITECTURE

The software architecture and modularity has major influence on the effort needed to coordinate the development phase. In his research, Parnas defined a software module as "a responsibility assignment rather than a subprogram" to clearly show that dividing a software system is simultaneously division of labor (Parnas, 1972).

In global software development, the drastically attenuated communication across sites makes it difficult to manage dependencies among developers working on the same modules. Keil and others used the term architectural adequacy for this with indicators like modularity, interface match and dependencies, communicability of the architecture, etc. (Keil et al., 2006)

A good software architecture and design is one in which design decisions about each component can be made in isolation from decisions about other components. If the organizational design and work distribution follows the software architecture it helps to minimize the need for communication and coordination among geographic teams (Herbsleb and Grinter, 1999). The allocation of different components to the geographic teams has been found to increase the productivity of distributed development. Architecture that require multiple sites to implement a change increase the time required to complete the change. (Herbsleb et al., 2001, Ebert and De Neve, 2001)

Unstable architecture can cause confusion among distributed teams as to what their responsibilities are and it can cause integration tests to become out-of-sync with the emerging product, resulting in spurious test failures and re-work (Mullick et al., 2006). If the original design, plan, and development processes are unstable, substantial communication between teams and across organizational boundaries will be required in global software development (Herbsleb and Grinter, 1999).

Keil and others (2006) identified the novelty of the software as a factor for additional cost overhead. Highly innovative solutions and architectures usually require a high level of

spontaneous communication, specific domain knowledge and they increase the frequency of unforeseen changes.

Product structure can also introduce subtle obstacles among collaborating teams: when a component contains high-visibility functionality, cooperation may be replaced by competition among teams to claim ownership of the component; the inverse would be true if a component is risky or error-prone (Herbsleb et al., 2005).

5.3. COST ESTIMATION TECHNIQUES

The cost estimation has been in the focus of software engineering research for many decades and hence a high number of different estimation techniques have been developed as discussed in Section 4. Unfortunately most of techniques for software cost estimation have been developed before the recent trend on global software development. Hence many techniques assume that the software is developed locally and therefore they do not take into account the additional challenges of global software development discussed in previous section 5.2.

Cost estimation of global software development differs from cost estimation of local software development at least in two different ways. Firstly, there is a large overhead effort or cost caused by several cost factors such as language differences, cultural barriers, or time shifts between sites, etc. Secondly, many cost factors (such as the skills and experience of the workforce) are site-specific and cannot be considered globally for a project. In many projects, the development sites have very different characteristics and thus the productivity and cost rate is different between sites. A standard effort prediction model consisting only of global cost factors cannot account for these differences.

In the recent research, available cost estimation techniques have been developed further to address the additional challenges and characteristics in global software development. Many of these techniques are based on the COCOMOII. Four different techniques found for cost estimation of global software development are described in the following sub-sections.

5.3.1. REFINEMENT OF COCOMOII BY KEIL, PAULISH AND SANGWAN

Keil, Paulish and Sangwan (Keil et al., 2006) refined COCOMOII technique to take into account additional complexities of global software development. These complexities were included as effort-multipliers into COCOMOII Post-Architecture model. The complexity factors identified by Keil and others were categorized into product, personnel, and project factors according to the original COCOMOII categorization. These factors are presented in table below.

Table 11: Additional cost factors used by Keil, Paulish and Sangwan in COCOMOII

Category	Factor	Description
Product	Precedentedness	novelty of the software to be developed, indicating the degree of innovation which is directly proportional to the level of spontaneous communication, the need for specific domain knowledge and the frequency of unforeseen changes.
	Architectural Adequacy	work assignments have to be carefully crafted taking into account the organizational structure and the functional coupling among software units. Therefore, the architecture has major influence on the efforts needed to coordinate the development phase. Indicators for the degree of Architectural Adequacy might be modularity, interface match and dependencies, communicability of the architecture, etc.
Personnel	Cultural Fit	closeness of team members' mental models [21] which is influenced by the combination of countries involved, the international experience of the teams, etc.
	Skill Level	educational level and language skills, indicating the formal abilities of remote team(s).
	Shared Understanding	tacit knowledge that is required, indicating the level of completeness of documentation and specification and the common knowledge about goals
	Information Sharing Constraints	representing competitive restrictions on information distribution, e.g. when working with external subcontractors or in security-sensitive environments.
Project	Novelty of Collaboration Model	initial cost for the search of offshore partners and contract negotiation
	Tools and Infrastructure	representing the homogeneity of the tool chains used in all sites and potential ramp-up costs for setting up the infrastructure in remote sites.
	Physical Distance	representing the potential overlaps of working time and, accordingly, the intensity of use of asynchronous communication media and collaboration tools

The calibration and estimation is done in the same way in the refined COCOMOII model as in the original version.

5.3.2. AMPLIFICATION OF COCOMOII BY BETZ AND MÄKIÖ

Betz and Mäkiö (Betz and Mäkiö, 2007) studied how COCOMOII could be enhanced to meet the requirements of global software development in the outsourcing setting, i.e. when software development is done by a 3rd party in a foreign country.

Betz and Mäkiö analyzed the existing cost factors of the COCOMOII Post-Architecture model and found that several factors were affected by offshore outsourcing. These factors are presented in table below:

Table 12: Cost factors impacted by offshore outsourcing setting based on Betz and Mäkiö

Category	Factor	Description
Product	Documentation Match to Life-Cycle	Different developer teams have differing documenting effort notwithstanding the fact that the document has the same standard. The effort hence changes if the development teams are distributed.
Personnel	Analyst Capability	The employees' competence definitely varies since they have different experience, education and settings in different sites.
	Programmer Capability	The employees' competence definitely varies since they have different experience, education and settings in different sites.
	Personnel Continuity	Evaluates the staff continuity, and it too varies since fluctuation is a critical factor of offshore outsourcing software development projects.
	Application Platform	differ depending on the buyers' experience.
	Language Experience	differ depending on the buyers' experience.
Project	Multi-Site Development	The only factor of the COCOMO II Which is linked with geographical distributed software development. The effort sources of this cost driver are the geographical distance between the development teams and the complexity of the communication channels.
Scale Factors like Precedentedness, Team Cohesion and Process Maturity are also affected. Precedentedness portrays the experience of a software developer to the present project context. This factor considerably varies between the offshore provider and the onshore buyer. The same applies to the factors like Team Cohesion and Process Maturity. The first indicates the ability of the team to work as a team. This factor is logically affected by offshore outsourcing development. The latter quantifies the company's process maturity.		

Betz and Mäkiö also used the modular composition of the COCOMOII to integrate the additional cost factors into the model. These additional cost drivers included new cost factors typical for offshore outsourcing software development. These factors were grouped into 4 groups: Outsourcing Factors, Buyers Outsourcing Maturity, Providers Outsourcing Maturity and Coordination Factors. The factors were added to the COCOMOII as additional Effort Multipliers.

Table 13: Additional cost factors used by Betz and Mäkiö in COCOMOII

Category	Factor	
Outsourcing Factors	Cultural Distance	-
	Barrier of Language	-
	Different Time Zones	-
Buyers Outsourcing Maturity	Buyer's Outsourcing Experience	The actual experience of the buyer with offshore outsourcing projects.
	Buyer's Project Managers	The capabilities of the buyer's project manager vis-à-vis their offshore qualification.
	Contract Design	The complexity of a collaboration contract.
Providers Outsourcing Maturity	Provider's Outsourcing Experience	the actual experience of the provider with offshore outsourcing projects.
	Provider's Project Managers	the capabilities of the provider's project manager considering their offshore qualification.
	-	-
Coordination Factors	Outsourcer's Fit	the right selection of a particular partner. The wrong partner increases the effort.
	Project Management	the increased effort which is inherent in any offshore outsourcing project.
	Team Spirit	Team spirit decreases the possible effort by raising the team spirit through team building meetings, common goals, and a mixture of off- and onshore team members.

The calibration and estimation is done in the same way in the amplification of COCOMOII as it is done in the original COCOMOII.

5.3.3. EXTENSION OF COCOMOII BY MADACHY

Madachy (Madachy, 2007) presented a totally different solution for extending COCOMOII for the needs of global software development. Based on the fact that different sites in a GSD project

might have different characteristics, his model allows the variation of effort multipliers by phase and a separation of factors for local vs. global project attributes. The model partitions the cost factors between local team-level and global project-level attributes. It also allows team-level labor category distributions per phase with local hourly rates and currencies to be defined.

The model refines COCOMOII formulas for phase-specific effort multipliers, team work distributions and local team attributes. The standard top level effort formula for COCOMOII is first used:

$$EFFORT = A \times SIZE^{B} \times \prod_{i=1}^{N} EM_{i},$$

Where effort is in person-months, A is a constant derived from historical project data, size is in KSLOC (thousand source lines of code), or converted from other size measures, B is an exponent for the diseconomy of scale dependent on additive scale factors, EM_i is an effort multiplier for the ith cost factor. The geometric product of N multipliers is an overall effort adjustment factor to the nominal effort.

Next the top level effort is decomposed in the new model for each phase, team, labor category, and then aggregated across the same dimensions and time periods determined by the schedule outputs. The changed effort algorithms sum the effort across phases, where each phase accounts for effort multipliers unique per phase. The resulting effort phase distribution may differ from the default lifecycle distribution due to differences in the team cost factors being unevenly weighted within the phases.

To get phase-level estimates, first the nominal (unadjusted) effort for each phase in a project is determined with

$$EFFORT_{NOMINAL_P} = EFFORT\%_{P} \times A \times SIZE^{B}$$

, where $Effort\%_p$ is the nominal percent of lifecycle effort in phase p. Madachy's model then uses the team distributions per phase and their local effort multipliers to calculate an adjusted effort for each team in each phase per:

$$EFFORT_{ADJUSTED_{t,p}} = EFFORT_{NOMINAL_p} \times EFFORT\,\%_{t,p} \times \prod_{i=1}^{N} EM_{t,i}$$

where Effort%$_{t,p}$ is the percent of lifecycle effort for team t in phase p, and the effort multipliers are those unique to team t. The adjusted effort outputs from above equation are summed up across teams for each phase, matrixed with their labor distributions, and then finally spread over time per the phase schedule spans to get detailed labor outputs.

The approach addresses the problem of having site-specific characteristics in global software development. It also provides a comparison between the effort required for traditional local and global software development. This is often very important for decision making. On the other hand, this extension model does not regard the additional overhead in distributed development that is caused by global cost factors such as cultural or time zone differences.

5.3.4. EXTENSION OF COBRA BY LAMERSDORF ET AL.

Lamersdorf, Münch, Viso Torre, Sánchez, and Rombach developed a model using CoBRA techniques for cost estimation of global software development. As a result of their research, necessary cost factors for the specific global software development context were identified and their impact was quantified on an empirical basis. This model was based on initial development of a cost overhead model for a Spanish global software development organization. (Lamersdorf et al., 2010)

The cost factors indentified by (Lamersdorf et al., 2010) are presented in the causal model below (Figure 21). It shows that ultimately, 14 factors were selected with 12 of them directly influencing effort overhead and 2 having indirect influences on all other factors. Out of the 14 factors, there were 3 characteristics of sites, 4 of tasks, 3 of project and process, and 4 relationships between sites.

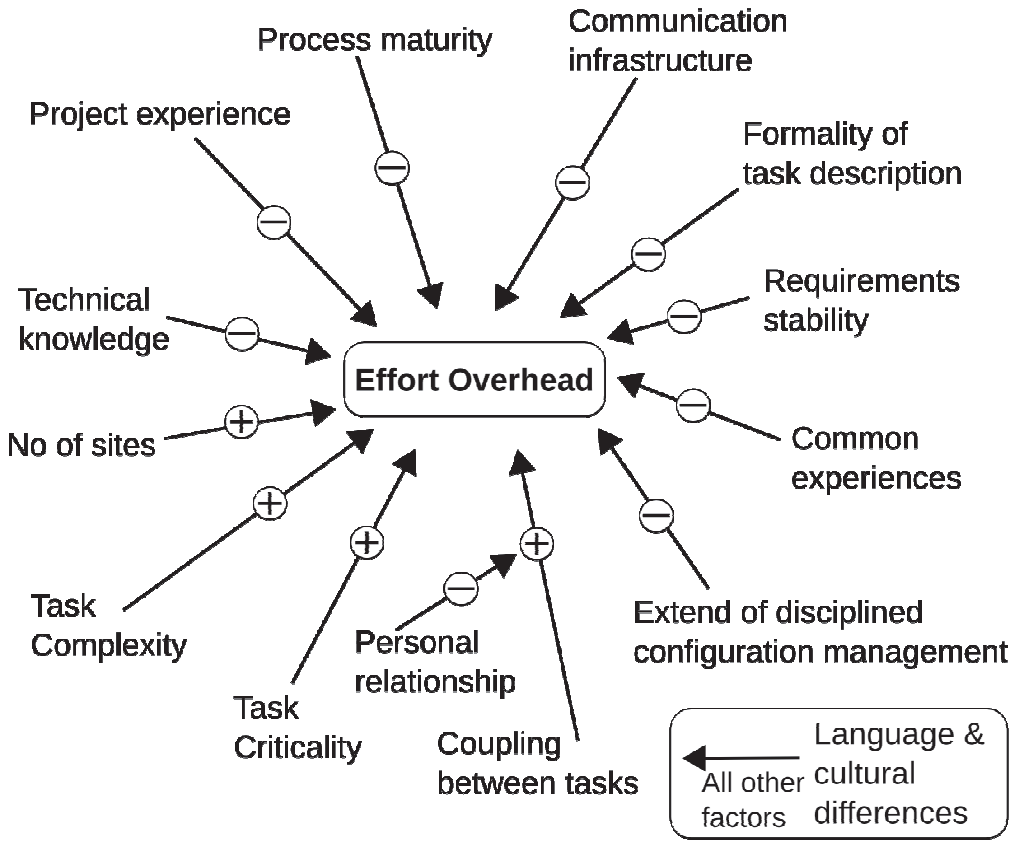

Figure 21: Causal model for cost estimation of global software development (Lamersdorf et al., 2010)

Table 14 below shows the aggregated estimation on the quantitative impact of each factor on effort overhead. It can be seen that process maturity, formality of the task description, and requirements stability are seen as the effort drivers with the highest impact – however, with different sensibilities towards language and cultural differences: While the impact on task description formality is rather small (only 10% distance between high and low cultural and language differences), the overhead of a low process maturity is very much dependent on language and cultural differences (nearly 20% distance). The individual rankings of the factors and especially the quantitative impact on productivity overhead are organization-specific and can be very different in different environments.

Table 14: Cost factors with values for global software development in CoBRA

Cost factor	Very low language and cultural differences	Very high language and cultural differences
Process Maturity (Site)	26,25 %	45,00 %
Project Experience (Site)	15,50 %	36,25 %
Technical Knowledge (Site)	18,75 %	33,75 %
Communication Infrastructure (Site dependent)	18,75 %	30,00 %
Common Experience (Site dependent)	9,25 %	16,50 %
Criticality (Task)	16,25 %	28,75 %
Complexity (Task)	17,50 %	32,50 %
Formality of Description (Task)	28,75 %	38,75 %
Requirement Stability (Project)	29,25 %	42,75 %
Number of Sites (Project)	10,50 %	19,00 %
Diciplined Configuration Management (Project)	19,75 %	27,75 %
Coupling - High relationships (Task)	9,50 %	16,25 %
Coupling - Low relationships (Task)	20,75 %	32,00 %

6. EVALUATION

6.1. EVALUATION QUESTIONS

This research aims to understand if and how the additional cost factors in global software development are included in the existing software cost estimation techniques. To evaluate this, eight different questions, i.e. evaluation criteria, were used. These are given in the table below.

Table 15: Evaluation questions

Topic	Evaluation Question
Geographic Distance	Is geographic distance considered as a cost factor in the technique?
Temporal Distance	Is temporal distance considered as a cost factor in the technique?
Cultural and Language Difference	Are cultural and language differences considered as cost factors in the technique?
Social Factors	Are social factors considered as cost factors in the technique?
Organization, Process and Project Management Factors	Are organization, process and project management factors considered as cost factors in the technique?
Infrastructure	Is infrastructure considered as a cost factor in the technique?
Product Architecture	Is product architecture considered as a cost factor in the technique?
Other	Does the technique include any other GSD related cost factors?

The evaluation questions were selected based on the overhead factors identified for global software development in the section 5.2

6.2. RESULTS

Next the different software cost estimation techniques were evaluated using the evaluation questions. The evaluation was performed by the researcher himself. The results and discussion for the results are described in the following sub-sections.

6.2.1. MODEL-BASED TECHNIQUES

The model-based techniques evaluated included Putnam model (SLIM), function point analysis and different versions of COCOMO (basic, intermediate, detailed, II).

The evaluation results for these techniques are presented in the table below.

Table 16: Evaluation results for model-based techniques

	Model-based cost estimation techniques					
Evaluation factors	**Putnam Model / SLIM**	**Function Point Analysis**	**COCOMO (basic)**	**COCOMO (intermediate)**	**COCOMO (detailed)**	**COCOMO II**
Geographic distance	No	No	No	No	No	Yes, Multisite development factor.
Temporal distance	No	No	No	No	No	No
Cultural and language differences	No	No	No	No	No	Yes, Team Cohesion
Social factors	No	No	No	No	No	Yes, Team Cohesion
Organization, Process, and project management factors	No	No	No	No	No	No
Infrastructure	No	No	No	No	No	No
Product architecture	No	Yes by using different cost/effort multipliers for different functions (locally / remotely developed).	No	No	No	No
Other consideration	No	No	No	No	No	No

With one exception, the evaluated model-based techniques did not consider GSD related cost factors in the estimation. Only COCOMOII contains two factors which can partially measure some of the overhead arising from global software development. These were effort multiplier called “Multisite Development” and scale factor called “Team Cohesion”.

“Multisite Development” is assessed by averaging the two factors: multisite collocation (ranges from fully collocated to international) and multisite communications (ranges from some mail to

interactive multimedia). It is intended to measure the impact of geographic distance and it has a linear impact on the estimate.

"Team Cohesion" is a scale factor with exponential impact on the estimate. It accounts for the sources of project turbulence and entropy due to difficulties in synchronizing the project's stakeholders: users, customers, developers, maintainers, interfaces, others. These difficulties may arise from differences in stakeholder objectives and cultures; difficulties in reconciling objectives; and stakeholder's lack of experience and familiarity in operating as a team. Hence the scale factor considers the impact of cultural differences and social factors on some level.

Putnam model (SLIM) uses software size, manpower build-up (combination of software and project type), scaling factor and process productivity as the cost factors for estimation. The process productivity is intended to measure the ability of a particular organization to produce software of a given size at a particular defect rate and it could be used as combined factor to reflect the overall productivity impact in GSD related estimation.

In Function point analysis, the number and type of different functionalities are used to estimate costs. Hence it does not include any GSD related factors. However it is possible to reflect differences in productivity at different sites (local or remote) by applying different cost multipliers for difference functionalities based on the location of planned development.

COCOMO variations have increasing number and complexity of cost factors. The basic version uses only software size and predefined project mode as the cost factors. The intermediate and detailed versions have 15 different cost factors and the COCOMOII contains 17 cost factors. Unfortunately all these factors are such that they cannot reflect the characteristics of GSD except for the mentioned "Multisite Development" and "Team Cohesion" factors in COCOMOII Post-Architecture model.

In general, the model-based techniques do not seem to be suitable for GSD related estimation as they do not reflect the additional overhead factor available in GSD. Hence these techniques cannot be used to explain the impact of GSD in the estimate or to compare cost differences between different GSD scenarios.

6.2.2. EXPERTISE-BASED TECHNIQUES

The expertise-based techniques evaluated were Delphi, Work-Breakdown Structure (WBS), Analogy, Work distribution, Top-Down and Bottom-Up techniques. The evaluation results for these techniques are presented in the table below.

Table 17: The evaluation results for expertise-based techniques

	Expertise based cost estimation techniques					
Evaluation Question	**Delphi**	**WBS**	**Analogy**	**Work distribution**	**Top-Down**	**Bottom-Up**
Geographic distance	No	No	No	No	No	No
Temporal distance	No	No	No	No	No	No
Cultural and language differences	No	No	No	No	No	No
Social factors	No	No	No	No	No	No
Organization, Process, and project management factors	No	Possible	No	No	No	No
Infrastructure	No	No	No	No	No	No
Product architecture	No	Possible	No	No	No	No
Other consideration		Break-down structures can be used to identify process and product architecture related overhead.				

None of the expertise-based techniques forces to consider GSD related cost factors in the estimation. All expertise-based techniques describe only a way how the costs can be estimated by the experts, but they leave the estimation and possible cost factors impacting the estimate to be determined by the experts. Hence it is up to the experts to provide the estimate based on their experience, skills and knowledge on theory.

Delphi technique describes a process that seeks to develop a consensus among expert group through meetings, questionnaires and surveys. Top-down and bottom-up techniques describe how to split costs between different parts of the software. As such these techniques can be applied to any software development project, local or global, in the same way.

Work-break down structure aims to organize project elements into a hierarchy (product or activity hierarchy) that is then estimated by the experts. It offers a way to take organizational, process and project management related additional activities into account through the activity hierarchy and product architecture related factors through the product hierarchy, but actually doing so is again left completely to the experts.

In analogy and work distribution techniques, the estimates are derived partly based on historical data and partly based on expert opinion. If the organization has experience and historical data from past GSD projects and these are used for the estimation, the techniques should be directly applicable and suitable for GSD. Without past experience and data from GSD projects, these techniques will not be useful for GSD projects.

In general, the expertise-based techniques are very good for unprecedented projects like GSD projects and in the absence of quantified, empirical data from past projects. As the accuracy and strength of these techniques relays completely on the judgment of the experts, these techniques required experts with past experience and knowledge on global software development.

Experts with solid knowledge and experience in GSD are able to consider all GSD specific overhead factors into the cost estimate. Good estimation experts can also factor in differences between past project experiences and the new techniques, teams, development sites, infrastructure involved in the project.

Without previous knowledge and expertise in GSD, these techniques do not provide any support for considering the impact of GSD related cost factors. In addition, the expertise-driven techniques are subject to human-errors and objectivity. In some cases, the experts might be biased towards GSD especially if they feel that their roles in the organization are threatened by the global software development.

6.2.3. LEARNING-BASED TECHNIQUES

The learning-based techniques include Case-Based Reasoning (CBR) and Artificial Neural Networks (ANN). The evaluation results for these techniques are presented in the table below.

Table 18: The evaluation results for learning-based techniques

	Learning based cost estimation techniques	
Evaluation Question	CBR	ANN
Geographic distance	No	No
Temporal distance	No	No
Cultural and language differences	No	No
Social factors	No	No
Organization, Process, and project management factors	No	No
Infrastructure	No	No
Product architecture	No	No
Other consideration	All overhead factors can be included into the set of relevant attributes describing a software project.	All overhead factors can be included as data input to the neural network.

In the Case-Based Reasoning (CBR) software projects (historical and future candidate projects) are described by a set of attributes (environment conditions, constraints, technologies, decision, success, cost spent, effort used) which are used in the estimation process. The CBR technique does not tell what attributes should be used to describe the software projects, but this is left to the person doing the estimation. Hence GSD related cost factors are not explicitly mentioned in the technique, but neither are any other cost factors.

All GSD related cost factors can be included as part of the describing attributes in CBG technique so that they are considered in the estimation. If the organization does not have past experience with GSD projects, external reference cases need to be found and used to produce accurate estimates. It is still recommended to use internal reference cases as those reflect the organization better.

Artificial neural network technique can also be used to estimate costs for GSD projects if the neural network is built to include the GSD specific cost factors as inputs. The neural network must also be trained with a set of data from past GSD projects.

As a draw-back neither of these techniques, CBR nor ANN, forces to use GSD related cost factors in the estimation. Both techniques describe only how the estimation model needs to be built, but they leave it up to the person doing the estimation to determine what attributes, inputs

or structure is used in the estimation. Hence it is up to the expertise of the estimator to include all relevant cost factors into the estimation.

Both learning-based techniques also require calibration, i.e. training, with data from past GSD projects. Unfortunately the amount of data needed for calibration increases with the complexity of the model (for example number of neurons or attributes). As there are many new cost factors involved in the GSD projects, these techniques require much more data from past GSD projects for calibration.

Unlike expertise-based techniques, learning-based techniques do not have problems with human-error and objectivity after the model (case attribute or the neural network) has been build. The estimation itself happens automatically once the model has been built.

6.2.4. DYNAMICS-BASED TECHNIQUES

In system dynamics, an estimation model is represented as a network with positive and negative feedback loops, stocks, flows, and information that changes over time and dynamically affects the flow rates between the feedback loops. This network represents cost and effort related factors and how they impact the productivity of the project team.

System dynamic models can be used for cost estimation of GSD projects if the additional cost factors are included to the used model. The system dynamics model for software cost estimation created by Abdel-Hamid and Madnick (Abdel-Hamid and Madnick 1991) does not include any of the GSD related cost factors, but it contains items like potential and actual productivity, workforce experience mix, error rate and software development rate which can be used to reflect some portion of the additional overhead in GSD projects.

Table 19: The evaluation results for system dynamics technique

	Dynamics-based cost estimation techniques
Evaluation factors	System Dynamics
Geographic distance	No
Temporal distance	No
Cultural and language differences	No
Social factors	No
Organization, Process, and project management factors	No
Infrastructure	No
Product architecture	No
Other consideration	All overhead factors can be included into the system dynamics models.

In order to fully use system dynamics technique for estimation of GSD project, one must build a new system dynamics models, which includes the additional GSD related cost factors. Hence it is again up to the expertise of the person doing the estimation to include all relevant cost factors into the estimation process.

It is important to note that system dynamics technique also requires that initial values of all cost factors are known. Hence these initial values need to be determined based on data from past GSD projects or by experts with experience and knowledge on global software development.

6.2.5. REGRESSION-BASED TECHNIQUES

The regression-based techniques use mathematical algorithms which produce a software cost estimate as a function of variables which represent the major cost factors. The regression-based techniques evaluated in this research were Ordinary Least Squares and ”Robust” regression. Neither of these techniques defines what variables should be used in the estimation and hence they do not force to have the GSD related cost factors as part of the regression variables. However it is, again, totally possible to include all these cost factors into the regression model as

variable, but this is left to the person doing the estimation. Hence this person needs to have expertise and knowledge to have GSD related cost factors as part of the regression model.

Table 20: The evaluation results for regression-based techniques

	Regression based cost estimation techniques	
Evaluation factors	Ordinary Least Squares	Robust Regression
Geographic distance	No	No
Temporal distance	No	No
Cultural and language differences	No	No
Social factors	No	No
Organization, Process, and project management factors	No	No
Infrastructure	No	No
Product architecture	No	No
Other consideration	All overhead factors can be included into the regression model.	

Regression-based techniques also require data from past GSD projects. This data is used to determine the regression coefficients. The amount of data required increases with the number of regression variables and hence GSD related projects will require much more data from past GSD projects compared to simpler models used for estimation of local software development.

6.2.6. COMPOSITE TECHNIQUES

Bayesian approach used by Chulani is based on COCOMOII and hence it includes the "Multisite development" and "Team Cohesion" as cost factors which reflect on some level the impact of geographic distance, cultural difference and social factors in GSD projects. Bayesian approach allows using other model-based techniques for the estimation as well and hence it can be used for GSD projects by selecting such technique which already includes all GSD related cost factors. This can be for example one of the GSD specific modifications of COCOMOII presented in Section 5.3.

CoBRA technique does not include GSD specific factors by default in the causal model, but those can be included into the model as already done by Lamersdorf and other (Lamersdorf et al., 2010). CoBRA has the benefit of visualizing the relationship and impacts of different cost factors as the factors are organized into a causal model that determines their impact on the project costs.

Through variation of these factors, it also enables an easy comparison between different GSD scenarios. In addition, it provides a simple way to compare the nominal productivity (optimal scenario) to the estimated scenario with the cost overhead.

Table 21: The evaluation results for composite techniques

	Composite techniques	
Evaluation factors	**Bayesian analysis**	**CoBRA**
Geographic distance	Yes, Multisite development factor from underlying COCOMO II	No
Temporal distance	No	No
Cultural and language differences	Yes, Team Cohesion	No
Social factors	Yes, Team Cohesion	No
Organization, Process, and project management factors	No	No
Infrastructure	No	No
Product architecture	No	No
Other consideration	All overhead factors can be included into the models.	

6.2.7. GSD SPECIFIC TECHNIQUES

The GSD specific techniques are extensions or modifications to COCOMOII or CoBRA techniques. All of them have been created to especially address the challenges and characteristics of global software development. The evaluation results for these techniques are given in the table below.

Table 22: The evaluation results for GSD specific techniques

	GSD specific techniques			
Evaluation factors	Refinement of COCOMO II by Keil, Paulish and Sangwan	Amplification of COCOMO II by Betz and Mäkiö	Extension of COCOMO II by Madachy	CoBRA extension by Lamersdorf et al
Geographic distance	Yes, physical distance + multisite development	Yes, enhanced multisite development	Yes, multisite development.	Yes, number of sites
Temporal distance	Yes, physical distance	Yes, different times zones	No	No
Cultural and language differences	Yes, team cohesion, cultural fit and skill level.	Yes, team cohesion, cultural distance and barrier of language	Yes, Team Cohesion	Yes, cultural and language differences
Social factors	Yes, team cohesion.	Yes, team cohesion and team spirit	Yes, Team Cohesion	No
Organization, Process, and project management factors	Yes, tools and infrastructure	Yes, project management	No	Yes, process maturity, formality of task description, and extend of disciplined configuration management.
Infrastructure	Yes, information sharing constraints and shared understanding.	Yes, Documentation Match to Life-Cycle and complexity of communication channels in multisite development evaluation.	No	Yes, communication infrastructure
Product architecture	Yes, precedentedness and architectural adequacy.	No	No	Yes, coupling between tasks
Other consideration	Additional factor for novelty of collaboration model.	Additional factors for outsourcer's fit and providers's and buyers outsourcing maturity.	The model partitions the cost drivers by having site specific factors.	Additional factors for requirement stability, common experiences, task criticality and complexity, technical knowledge, and project experience.

COCOMOII extensions, from Keil, Paulish and Sangwan as well as from Betz and Mäkiö, include all GSD related cost factors in the estimation model. Keil, Paulish and Sangwan have not themselves considered the social factors in their extension, but it has been addressed partly through "team cohesion" – scale factor in the original COCOMOII. Keil, Paulish and Sangwan have also added an additional cost factor for initial costs of setting up the global software development structure.

Betz and Mäkiö have included all GSD related cost factors except product architecture. Betz and Mäkiö use the perspective of outsourcing in their amplification of COCOMOII and hence they have included many additional cost factors for service provider's and buyers outsourcing maturity and capabilities.

Extension of COCOMOII from Madachy does not contain any new cost factors. Hence it has only "Multisite Development" and "Team Cohesion" factors which are already available in COCOMOII. However Madachy's extension enables to determine all effort multipliers and scale factors of COCOMOII separately for different sites. Hence for example personnel continuity can be set differently for local and remote sites. By doing so the extension allows considering site specific difference properly, but at the same time neglects the generic global overhead factors like cultural or language differences or infrastructure overhead in the estimation.

CoBRA extension from Lamersdorf and others contains all GSD related cost factor except for temporal distance and social factors. In addition their model has many other cost factors like requirement stability, common experiences, task criticality, task complexity, technical experience and project experience which are applicable to any software development projects, local or remote. The CoBRA extension highlights that the cultural and language differences have an impact on all other cost factors instead of having direct impact on the effort / productivity.

All other GSD specific techniques except for CoBRA require more data from past GSD projects for calibration. This is due to the more complex models which require more parameters to be defined. The amount of data needed for CoBRA does not increase with the complexity of the causal model as CoBRA only uses the data for calibration of the nominal productivity. It is not used for the calibration of causal model.

6.3. DISCUSSION

When most of the software cost estimation techniques have been developed before the trend of global software development, it was assumed initially that these techniques would not be applicable and suitable for estimation of global software development. However the evaluation results in this research show differently which is rather surprising.

Based on the evaluation, it seems that the model-based techniques are the biggest group of techniques which cannot be used for estimation of global software development projects. Only the COCOMOII and function point analysis are applicable from model-based techniques, but COCOMOII might give inaccurate results as it does not take into account all GSD related cost factors. Function point analysis requires using different cost multipliers for difference functionalities based on the location of planned development.

Other categories of techniques, and techniques within different categories, are suitable and applicable for global software development projects. For example, most of the expertise-based techniques can be used directly for GSD projects, but they require experts to have theoretical knowledge or past experience in global software development.

Other techniques including learning-based techniques, system dynamics, regression-based and composite techniques, can be used in global software development, but unfortunately these

techniques are quite theoretical. This means that the techniques present an approach for estimation and show an example how to use the approach, but they do not give an explicit list of cost factors to be used. This is left to the person using the technique. Hence these techniques require a person with experience and knowledge on global software development for the setup and calibration work before they can be used for estimation of GSD projects. Without solid experience and knowledge on GSD, it is unlikely that the estimation model would include all necessary GSD related cost factors.

It is also important to note that there might be quite a significant amount of effort in the setup and calibration of these techniques, but it is required only once before applying the model for future projects.

The recent research on cost estimation for global software development has emerged few GSD specific techniques which contain almost all of the common GSD related cost factors. Two of these techniques have a little bit different perspective and approach for the cost estimation of global software development. Betz and Mäkiö have taken the outsourcing perspective in their extension of COCOMOII and hence their model includes additional offshore outsourcing related cost factors. Madachy's extension of COCOCOII differs from the other techniques by having a totally different approach as it enables to determine all multipliers and factors in COCOMOII separately for different sites. Keil, Paulish and Sangwan have only added the GSD related cost factors to COCOMOII where as Lamersdorf and others have done the same for CoBRA technique.

It is very unfortunate that all GSD specific techniques are model-based techniques. Three of them are even based on the same COCOMOII where as one is based on CoBRA. The similarity of these techniques is rather unfortunate as the model-based techniques have common pros & cons and hence they are not suitable for all scenarios. For example, they are unable to consider exceptional or unexpected conditions, or they cannot compensate for poor and inaccurate input or calibration data. It also always remains an open questions how the models can represent future projects with new techniques and technology when they are calibrated with data from past projects.

As it is usually recommended to utilize multiple different techniques as part of the estimation process, there definitely seems to be a need and room for new adaptations on different types of estimation techniques for global software development. These could be for example adaptations of expertise or learning-based techniques for global software development.

Another important observation is the necessity of historical data from past projects as a precondition to use most of the techniques. It seems that almost all techniques require historical data for set-up or it is at least recommended for the model calibration. For example all model-based techniques are recommended to be calibrated with data from past projects. Also neural network, case-based reasoning, regression, system dynamics techniques require calibration or training before they can be used. Even some of the expertise based techniques like analogy and work distribution are useless without data from past projects.

In order to get accurate results, the data should be from similar past experiences, which means that it should originate from past global software development projects. This might create significant difficulties for many organizations as they are either lacking experience in GSD projects or have not collected data from their earlier projects. It might also be very difficult to find suitable data from external sources.

The need for data becomes more difficult for many of the techniques with global software development as it complicates the estimation model (neural network, regression model, set of describing attributes in CBR, COCOMOII's effort multipliers and scale factors). As the estimation models get more complicated with GSD related cost factors, more data is required to determine values for the different factors in the models.

7. CONCLUSIONS

This research is one of the first systematic reviews on applicability of different software cost estimation techniques to global software development. There does not seem to be any other researches available which systematically evaluate how global software development related overhead factors are considered in the different software cost estimation techniques.

In this research, a high number of different software costs estimation techniques have been reviewed which has led to a good understanding on how these techniques can be applied to the special conditions of global software development. In the beginning, the research problem was formulated with three questions. The answers based on this research for the research questions are summarized below.

7.1. WHAT FACTORS IMPACT COSTS IN GLOBAL SOFTWARE DEVELOPMENT?

There seems to be many different factors which impact the costs in global software development. The most important factors include geographic and temporal distance, differences in language and culture, social factors, problems arising from organization structure, processes and projects, barriers deriving from infrastructure and product architecture.

Most of the global software development related cost factors have usually a negative impact on productivity and hence they increase the effort and costs required for a software development project. The factors can also have a compounding effect on each other, increasing the possibility of a negative impact on the development process.

Geographic distance has a significant negative impact on the coordination, communication and interaction which occur in an informal “ad hoc” manner when team is located in the same physical location. These are vital for successful software development as a lot of coordination, communication and interaction is required to develop a shared understanding on the software being developed. The physical separation has an impact even on a very short physical separation as well, for example when team members are in different buildings.

Temporal distance reduces possibilities for synchronous communication, which is a critical communicational attribute for real-time problem solving and design activities. The temporal distance has a “U”- shape effect on productivity and production speed is best when there is either no temporal distance or these is no overlap between the teams. Temporal distance has also an increasing negative impact on accuracy of communication.

Cultural and language differences have a wider impact as these differences have been recognized as major barriers to communication. These differences can lead to serious and chronic

misunderstandings and communication problems. The cultural and language differences also increase the amount of time and effort used for communication itself. Cultural problems are manifested in two forms: organizational and national culture. Hence the problems can arise even within the same country.

Fear and distrust can have negatively impact the motivation, desire to work with, trust, cooperate, communicate and share knowledge with remote colleagues and hence they have a direct bearing on the success of implementing global software development. Geographic, temporal, and cultural distance makes these social problems worse due to the weaker interpersonal relations between different teams. These issues are even more complex in outsourcing, collaboration and partnership arrangements.

Organizational, process and project issues are related to when, by whom and how should activities be performed and tasks allocated. Collocated and coherent teams have been reported to have the best productivity. Organization structure also establishes communication boundaries and barrier within the organization. Different project management practices and processes can introduce re-work or loss of data due to misunderstandings, integration and interoperability problems.

Infrastructure setup and differences create communication barriers which need to be overcome to facilitate information transfer between interdependent team members. Inadequate knowledge management infrastructure can inhibit the formation of a shared understanding among project. The lack of effective information sharing mechanisms, poorly maintained documentation, and lack of collaboration on artifacts increases possible issues in managing knowledge, which can lead to lower productivity, quality and other difficulties.

The software architecture and modularity has an influence on the effort needed to coordinate the development phase. In global software development, the drastically attenuated communication across sites makes it difficult to manage dependencies among developers working on the same modules. If the organizational design and work distribution follows the software architecture it helps to minimize the need for communication and coordination among geographic teams, which increases productivity.

The cost factors identified in the research literature for global software development do not include cost factors which rise outside of software engineering. These include factors like set-up factors (contracting to a foreign country, founding an office in foreign country), illegal (corruption in many developing countries) and international trade related factors (export and import taxes and tariffs). These factors might also have significant impact on the overall costs of a software development project.

7.2. HOW THESE COST FACTORS ARE CONSIDERED IN THE AVAILABLE SOFTWARE COST ESTIMATION TECHNIQUES?

The global software development related cost factors are not properly considered in the evaluated software cost estimation techniques. Almost all techniques do not take any of the additional cost factors into account by default. Only COCOMOII technique contains two factors, "Multisite Development" and "Team Cohesion" which take into account some impact of geographic distance, cultural differences and social factors.

There are also four adaptations to COCOMOII and CoBRA techniques which have been developed to especially address the challenges and characteristics of global software development. Three of these adaptations (Keil et al., 2006, Betz and Mäkiö, 2007, Madachy, 2007) are for COCOMOII and one (Lamersdorf et al., 2010) is for CoBRA techniques.

COCOMOII extension from Keil, Paulish and Sangwan contains all GSD related cost factors in the estimation model. It has also an additional cost factor for initial costs of setting up the global software development structure.

Betz and Mäkiö have included all GSD related cost factors except product architecture. Betz and Mäkiö have taken the perspective of outsourcing in their adaptation and hence they have included many additional cost factors for service provider's and buyer's outsourcing maturity and capabilities.

Extension of COCOMOII from Madachy has only "Multisite Development" and "Team Cohesion" cost factors included from the standard COCOMOII, but the extension enables to determine all effort multipliers and scale factors separately for different sites. Hence it allows

considering site specific differences and characteristics very well, but at the same time it ignores the general global overhead factor like cultural and language differences.

CoBRA extension from Lamersdorf and others (Lamersdorf et al., 2012) contains all GSD related cost factors except for temporal distance and social factors. The CoBRA extension highlights that the cultural and language differences have an impact on all other cost factors instead of having direct impact on the effort / productivity.

7.3. HOW AVAILABLE SOFTWARE COST ESTIMATION TECHNIQUES CAN BE APPLIED IN GLOBAL SOFTWARE DEVELOPMENT?

Even though most of the evaluated software cost estimation techniques do not have any of the GSD related cost factors included by default, these techniques are still suitable and applicable for estimation of global software development project with some set-up and calibration work. Techniques like learning-based techniques, system dynamics, regression and composite techniques can be applied to global software development if the person doing the estimation model set-up is experienced in global software development and hence the person is able to include all necessary cost factors into the estimation model.

Also all expertise-based techniques can be directly applied for GSD projects, but they require experts with experience and knowledge on global software development. The four available GSD specific techniques can naturally be also directly applied for GSD projects.

Most of the model-based techniques cannot be used for estimation of global software development projects. Only COCOMOII and function point analysis, by varying cost multipliers for sites, seem to be applicable on some level for GSD projects.

As the earlier research on GSD has focused on the model-based techniques and as the older model-based techniques seems to be inadequate for estimation of GSD projects, it is new and unexpected to find that many of the other techniques are actually suitable and quite easily applicable to GSD projects.

Table 23: Suitability and application of different techniques for estimation of GSD projects

Software Cost Estimation Techniques	Can be applied for GSD projects?	How to apply for GSD projects?
Model-based		
Putnam Model / SLIM	No	
Function Point Analysis	Yes	Use different cost multipliers for difference functionalities based on the location of planned development.
Basic COCOMO	No	
Intermediate COCOMO	No	
Detailed COCOMO	No	
COCOMOII	Maybe	Use multisite development and team cohesion to reflect geographic distance, cultural differences and social factors. Use data from past GSD projects for calibration.
Expertise-based Techniques		
Delphi	Yes	Use experts with experience and knowledge on GSD.
Work-breakdown-structure	Yes	Use experts with experience and knowledge on GSD.
Analogy	Yes	Use data from past GSD projects and experts with experience and knowledge on GSD.
Work Distribution	Yes	Use data from past GSD projects and experts with experience and knowledge on GSD.
Top-Down	Yes	Use experts with experience and knowledge on GSD.
Bottom-Up	Yes	Use experts with experience and knowledge on GSD.
Learning-based Techniques		
Case-Based Reasoning	Yes	Include GSD related cost factors to the describing attributes. Use data from past GSD projects for calibration.
Artificial Neural Networks	Yes	Include GSD related cost factors to the network structure as neurons. Use data from past GSD projects for network training.
Dynamics-based Techniques		
System Dynamics	Yes	Include GSD related cost factors to the system model.
Regression-based Techniques		
Ordinary Least Squares	Yes	Include GSD related cost factors to the regression model as variables. Use data from past GSD projects for determining regression coefficients.
Robust Regression	Yes	Include GSD related cost factors to the regression model as variables. Use data from past GSD projects for determining regression coefficients.
Composite Techniques		
Bayeasian Approach	Maybe	Use multisite development and team cohesion to reflect geographic distance, cultural differences and social factors. Use data from past GSD projects for calibration.
CoBRA	Yes	Define causal model containing GSD related cost factors. Use data from past GSD projects for determining the nominal productivity.
GSD specific techniques		
COCOMOII / Keil et al. 2006	Yes	Use data from past GSD projects for calibration.
COCOMOII / Betz and Mäkiö 2007	Yes	Use data from past GSD projects for calibration.
COCOMOII / Madachy 2007	Yes	Use data from past GSD projects for calibration.
CoBRA / Lamensdorf et al. 2010	Yes	Use data from past GSD projects for determining the nominal productivity.

Most of the software cost estimation techniques require data from past projects as a precondition for set-up or calibration. In order to get accurate results from these techniques, the data should originate from past experiences similar to the new project being estimated. This means that the data should originate from past global software development projects, which might create significant difficulties for many organizations as they are either lacking experience in GSD projects or have not collected data from their past projects. It might also be very difficult to find suitable data from external sources. This prerequisite is often forgotten or ignored in the research.

The need for historical data becomes more difficult with global software development as it complicates the estimation model (neural network, regression model, set of describing attributes in CBR, new effort multipliers and scale factors in COCOMOII adaptations). As the estimation models get more complicated, more data is required to determine or calibrate values for the cost factors.

In order to get accurate results and understand the estimates for a software development project, it is recommended to use multiple different techniques as part of the estimation process. As all techniques especially developed to address the challenges and characteristics of global software development are model-based techniques, it becomes more difficult to do so. The similarity of the GSD specific techniques is rather unfortunate also due to the fact that the model-based techniques have their own pros & cons which makes them suitable only for certain scenarios. Hence it seems that there is a strong need for new adaptations on different type of estimation techniques for global software development. For example creation of neural network or system dynamics model, or defining and verifying a regression model for global software development.

REFERENCES

Aamodt, A. and Plaza, E. (1994). "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches", AI Communications, IOS Press, 1995, vol. 7:1, pp. 39-59.

Abdel-Hamid, T. and Madnick, S. (1991). Software Project Dynamics, Prentice-Hall, 1991.

Abdel-Hamid, T. and Madnick, S. (1993). "Modeling the Dynamics of Software Reuse: an Integrating System Dynamics Perspective", Presentation to the 6th Annual Workshop on Reuse, Owego, NY, November 1993.

ACM (2006). Globalization and Offshoring of Software – A report of the ACM Job Migration Task Force. Association for Computing Machinery, 2006.

Albrecht, A. J. (1979). "Measuring application development productivity", Proceeding of the Joint SHARE, GUIDE and IBM application development symposium, IBM Corporation, 1979.

Allen, T. (1984). Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information within the R&D Organization, MIT Press, Cambridge, MA, 1984.

Betz, S. and Mäkiö, J. (2007). "Amplification of the COCOMO II regarding offshore software projects", Workshop on Offshoring of Software Development-Methods and Tools for Risk Management at the second International Conference on Global Software Engineering, 2007.

Bhat, J. M., Mayank, G. and Murthy, S. N. (2006). “Overcoming requirements engineering challenges: Lessons from offshore outsourcing”, IEEE Software, Vol. 23:1, pp. 38–44, 2006.

Boehm, B. (1981). Software Engineering Economics, Prentice Hall, 1981.

Boehm, B., Abts, C. A., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. J. and Steece, B. (2000a). Software cost estimation with COCOMO II, Prentice Hall, 2000.

Boehm, B., Abts, C. A. and Chulani, S. (2000b). “Software Development Cost Estimation Approaches – A Survey”, Annals of Software Engineering, Vol 10:1-4, pp. 177-205, 2000.

Briand, L., Emam, K., El, and Bomarius, F. (1998). “COBRA: A hybrid method for software cost estimation, benchmarking, and risk assessment”, International Conference on Software Engineering, ICSE, pp. 390-399, 1998.

Brooks, F. (1975). The Mythical Man-Month, Addison-Wesley, Reading, MA, United States, 1975.

Bruegge, B., Dutoit, A. H. and Wolf, T. (2006). “Sysiphus: Enabling informal collaboration in global software development”, In Proceedings of the IEEE international conference on Global Software Engineering (ICGSE ’06), IEEE Computer Society, Florianopolis, Brazil, pages 139–148, 2006.

Carmel, E. (1999). Global Software Teams: Collaborating across Borders and Time Zones, Prentice Hall, Upper Saddle River, NJ, 1999.

Carmel, E. and Agarwal, R. (2001). “Tactical approaches for alleviating distance in global software development”, IEEE Software, Vol 1:2, pp. 22–29, 2001.

Casey, V. and I. Richardson. (2006). “Uncovering the Reality within Virtual Software Teams”, in First International Workshop on Global Software Development for the Practitioner, ICSE 2006, Shanghai, China, 2006.

Casey, V. and Richardson, I. (2008). “The impact of fear on the operation of virtual teams”, In Proceedings Proceedings of the 2008 IEEE International Conference on Global Software Engineering (ICGSE ’08), IEEE Computer Society, Bangalore, India, pages 163–172, 2008.

Chulani, S., Boehm, B. and Steece, B. (1999). “Calibrating Software Cost Models Using Bayesian Analysis,” IEEE Transactions on Software Engineering, Special Issue on Empirical Methods in Software Engineering, 1999.

Conchúir, E. Ó., Ågerfalk, P.J., Olsson, H.H., and Fitzgerald, B. (2009). “Global Software Development: Where are the Benefits?”, Communications of the ACM, Vol. 52:8, August 2009.

Ebert, C. and De Neve, P. (2001). “Surviving global software development”, IEEE Software, vol. 18:2, pp. 62-69, March-April 2001.

Espinosa, J.A, Nan, N. and Carmel, E. (2007). “Do Gradations of Time Zone Separation Make a Difference in Perfomance? A First Laboratory Study”, International Conference on Global Software Engineering (ICGSE), IEEE Software Society, Munich, Germany, August 2007.

Forrester, J. (1961). Industrial Dynamics, MIT Press, Cambridge, MA, 1961.

Freiman, F.R. and Park, R.E. (1979). The Price software cost model: RCA government systems division., IEEE, 1979.

Gray, A. R. and MacDonnell, S.G. (1997). “A Comparison of Techniques for Developing Predictive Models for Software Metrics”, Information and Software Technology, Vol 39, 1997.

Haikala, I. and Märijärvi, J. (1997). Ohjelmistotuotanto, Suomen Atk-kustannus Oy, 1997.

Haugan, G. (2001). Effective work breakdown structures, Project Management Institute, 2001.

Heemstra, F. J. (1992). “Software cost estimation”. Information and Software Technology, vol. 34:10, pp. 627-639, 1992.

Helmer, O. (1966). Social Technology, Basic Books, NY, 1966.

Herbsleb, J. D., and Grinter, R. E. (1999). “Splitting the organization and integrating the code: Conway’s law revisited”, Proceedings of the 21st international Conference on Software Engineering (ICSE 99), IEEE Computer Society Press, Los Alamitos, CA, pp. 85-95, May 16-22, 1999.

Herbsleb, J. D., Mockus, A., Finholt, T. A. and Grinter, R. E. (2001). ”An empirical study of global software development: Distance and speed”, International Conference on Software Engineering (ICSE 2001), pp 81-90, 2001.

Herbsleb, J. D. and Mockus, A. (2003). “An empirical study of speed and communication in globally distributed software development”, IEEE Transactions on Software Engineering, Vol. 29:6, pp. 481–494, 2003.

Herbsleb, J.D. and Moitra, D. (2001). “Global Software Development”, IEEE Software, Vol. 18:2, pp. 16-20, 2001.

Herbsleb, J.D., Paulish, D., and Bass, M. (2005). "Global software development at Siemens: experience from nine projects," Proceedings of the 27th International Conference on Software Engineering, St. Louis, MO, USA, pp. 524–533, 2005.

Hihn, J. and Habib-agahi, H. (1991). "Cost Estimation of Software Intensive Projects: A Survey of Current Practices", Proceedings of the 13th International Conference on Software Engineering (ICSE), IEEE Computer Society Press, pp. 276-287, 1991.

Hofstede, G. (1997). Cultures and organizations: software of the mind, McGraw-Hill, London, 1997.

Idri, A., Abran, A. and Khoshgoftaar T. M. (2002). Fuzzy Case-Based Reasoning Models for Software Cost Estimation", Soft Computing in Software Engineering, Springer-Verlag, 2002.

Jensen, R. (1983)."An Improved Macrolevel Software Development Resource Estimation Model", Proceedings 5th ISPA Conference, pp. 88-92, April 1983.

Jones, C. (1997). Applied Software Measurement, McGraw Hill, 1997.

Jørgensen, M. and Moløkken-Østvold, K. (2004). "Reasons for Software Effort Estimation Error: Impact of Respondent Role, Information Collection Approach, and Data Analysis Method", IEEE Transactions on Software Engineering, Vol. 30:12, December 2004.

Karunanithi, N., Whitley, D., and Malaiya, Y. K. (1992). "Using Neural Networks in Reliability Prediction", IEEE Software, vol. 9:4, pp. 53-59, July 1992.

Keating, G. (1985). The Production and Use of Economic Forecasts, Methuen & Co Ltd, London, 1985.

Keil, P., Paulish, D. J., and Sangwan, R. (2006). “Cost estimation for global software development”, International workshop on Economics Driven Software Engineering, Shanghai, China, pp 7-10, 2006.

Kraut, R. E. and Streeter, L. A. (1995).”Coordination in software development”, Communications of the ACM, Vol. 38:3, March 1995.

Lamersdorf, A., Münch, J., Viso Torre, A. F., Sánchez, C. R. and Rombach, D. (2010), “Estimating the Effort Overhead in Global Software Development”, International Conference on Global Software Engineering (ICGSE ’10), IEEE Computer Society, pp. 267-276, 2010.

Lederer, A.L. and Prasad, J. (1995). “Causes of Inaccurate Software Development Cost Estimates”, Journal of Systems and Software, Vol. 31, pp. 125-134, 1995.

Leonard, D. A., Brands, P. A., Edmondson, A. and Fenwick, J. (1997). “Virtual teams: Using communications technology to manage geographically dispersed development groups,” in Sense and Respond: Capturing Value in the Network Era, S. P. Bradley and R. L. Nolan, Eds. Cambridge, MA: Harvard Business School Press, 1997.

Lutz, B. (2009). “Linguistic challenges in global software development: Lessons learned in an international SW development division”, In Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering (ICGSE ’09), IEEE Computer Society, Limerick, Ireland, pp. 249–253, 2009.

Madachy, R. (1994). “A Software Project Dynamics Model for Process Cost, Schedule and Risk Assessment”, Ph.D. dissertation, Madachy, R., University of Southern California, United States, 1994.

Madachy, R. (2007). “Distributed global development parametric cost modeling”, in Software Proces Dynamics and Agility, Lectures notes in Computer Science, Springer, Vol. 4470/2007, pp. 159-168, 2007.

McChesney, I. and Gallagher, S. (2004). “Communication and co-ordination practices in software engineering projects”, Information and Software Technology, vol. 46:7, 473-489, 2004.

McConnell, S. (2006). Software Estimation: Demystifying the Black Art, Microsoft Press, Redmond, Washington, 2006.

MIL-HDBK-881 (1998). Work Breakdown Structures for Defence Material Items, Department of Defence, United States of America, 1998.

Mullick, N., Bass, M., Houda, Z. E., Paulish, D. J., Cataldo, M., Herbsleb, J. D., Bass, L. and Sangwan, R. (2006). “Siemens global studio project: Experiences adopting an integrated GSD infrastructure”, In Proceedings of the IEEE international conference on Global Software Engineering (ICGSE ’06), IEEE Computer Society, Florianopolis, Brazil, pp. 203–212, Oct. 2006.

NASA (2003) – Handbook for Software Cost Estimation, NASA, Jet Propulsion Laboratory, Pasadena, California, 2003.

Niinimäki, T., Piri, A. and Lassenius, C. (2009). “Factors affecting audio and text-based communication media choice in global software development projects”, In Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering (ICGSE ’09), IEEE Computer Society, Limerick, Ireland, pp. 153–162, 2009.

Noll, J., Beecham, S., and Richardson, I. (2010). “Global software development and collaboration: barriers and solutions”, ACM Inroads, Vol. 1:3, pp. 66-78, ACM, New York, NY, USA, September 2010.

Paré, G. and Dubé, L. (1999). “Virtual teams: an exploratory study of key challenges and strategies”, In P. De and J. I. De Gross, editors, Proceedings of the 20th International Conference on Information Systems, Association for Information Systems, Charlotte, North Carolina, USA, pp. 479–483, December 1999.

Parnas, D.L. (1972). "On the Criteria to be Used in Decomposing Systems into Modules," Communications of the ACM, Vol. 15:12, pp. 1053-1058, 1972.

Perry, D. E., Staudenmayer, N. A. and Votta, L. G. (1994). “People, Organizations, and Process Improvement”, IEEE Software, Vol 11:4, pp. 36-45, 1994.

Phan, D. (1990). Information Systems Project Management: an Integrated Resource Planning Perspective Model, in Department of Management and Information Systems, Arizona: Tucson, USA, 1990.

Piri, A., Niinimäki, T. and Lassenius, C. (2009). ”Descriptive analysis of fear and distrust in early phases of GSD projects”, In Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering (ICGSE ’09), IEEE Computer Society, Limerick, Ireland, pp. 105–114, 2009.

PMBOK (2000). A Guide to the Project Management Body of Knowledge, Project Management Institute, 2000.

Putnam, L. (1978). ”A general empirical solution to the macro software sizing and estimating problem”, IEEE Transaction on Software Engineering, pp. 345-361, July 1978.

Rice, R.E., D’Ambra, J. and More, E. (1998). “Cross-cultural comparison of organizational media evaluation and choice,” Journals of. Communications, vol. 48:3, pp. 3–26, 1998.

Richardson, G. P, Fishwick, P.A. and Luker, P.A. (1991). System Dynamics: Simulation for Policy Analysis from a Feedback Perspective, in Qualitative Simulation Modeling and Analysis, Springer Verlag, New York, USA, 1991.

Roberts, P. (1997). Guide to project management, Profile Books Limited/The Economist, 1997.

Rubin, H. (1983). “Interactive macro-estimation of software life cycle parameters via personal computer: a technique for improving customer/developer communication”, in Proc. Symp. on application & assessment of automated tools for software development, IEEE, San Francisco, 1983.

Ruhe, M., Jeffrey, R., and Wieczorek, I. (2003). “Cost Estimation for Web Applications”, 25th International Conference on Software Engineering (ICSE 2003), pp. 285-294, 2003.

Shepperd, M., Schofield, C., and Kitchenham, B. (1996). ”Effort Estimation Using Analogy”, Proceedings of the 18th international conference on Software engineering (ICSE ‘96), Berlin, Germany, p.170-178, March 25-29, 1996.

Sosa, M., Eppinger, S., Pich, M., McKendrick, D. and Stout, S. (2002). “Factors that influence technical communication in distributed product development: an empirical study in the telecommunications industry”, IEEE Transactions on Engineering Management, vol. 49:1, pp. 45-58, February 2002.

Subramanian, G. H. and Breslawski, S. (1995). “An Empirical Analysis of Software Effort Estimate Alterations,” Journals of Systems and Software, vol. 31:2, pp. 135-141, 1995. (OK)

Taweel, A., Delaney, B., Arvanitis, T.N. and Zhao, L. (2009). “Communication, knowledge and co-ordination management in globally distributed software development: Informed by a scientific software engineering case study”, In Proceedings of the 2009

Fourth IEEE International Conference on Global Software Engineering (ICGSE '09), IEEE Computer Society, Limerick, Ireland, pages 370–375, 2009.

Teasley, S.D., Covi, L.A, Krishnan, M.S. and Olson, J.S. (2002). "Rapid Software Development through Team Collocation", IEEE Transactions on Software Engineering, vol. 28:7, pp. 671–683, 2002.

Trendowicz, A., Heidrich, J., Münch, J., Ishigai, Y., Yokoyama, K., and Kikuchi, N. (2006). "Development of a hybrid cost estimation model in an iterative manner", 28th International Conference on Software Engineering (ICSE 2006), pp. 331-340, 2006.

Van Genuchten, M. (1991). "Why is Software Late? An Empirical Study of Reasons for Delay in Software Development," IEEE Transactions on Software Engineering, vol. 17:6, pp. 582-590, June 1991.

Wittig, G.E. and Finne, G.R. (1994). "Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort", Australian Journal of Information Systems, 1994.

Wittig, G.E. and Finne, G.R. 1997. "AI tools for software development effort estimation", Software Engineering and Education and Practice Conference, IEEE Computer Society Press, pp. 346-353, 1996.

Wolverton, R W. (1974). "The Cost of Developing Large-Scale Software", IEEE Trans Computers, pp. 615-636, June 1974.