Chapter 2

ANALYSIS OF SOFTWARE ESTIMATION APPROACHES

This chapter depicts literature referred related to software estimation approaches. Initially, literature related to existing software estimation techniques, models, methods and tools, is analyzed. Further, the literature referred is classified according to early and intermediate stage of software development.

2.1 SOFTWARE ESTIMATION METHODS

Brundick, Bill compiled useful information in Parametric Cost Estimation Handbook[34]. This handbook contains description of software cost estimation processes, risk involved, estimation methodologies, standards and tools. The estimation methods such as Analogy, top-down, bottom-up, expert judgment and parametric or algorithmic models are described in brief. Expert judgment involves consulting with human experts to use their experience and understanding of a proposed project to provide an estimate for the cost of the project. Expert can factor differences between past project experience and requirements of the proposed project. The expert can also factor in project impacts caused by new technologies, applications and languages. Expert judgment always compliments other estimation methodologies. Major disadvantage is that the estimates are highly dependent upon expertise and judgment. Documenting the factors used by expert, has been bottleneck to repeat the accuracy. This handbook also listed calibrated and validated tools such as REVIC (Revised Enhanced Version of Intermediate COCOMO), PRICES, SASET (Software Architecture, Sizing and Estimating Tool), SEER-SEM (Software Evaluation and Estimation of Resources-Software Estimation Model), SLIM (Software Lifecycle Model), SOFTCOST-R, SYSTEM-4. The software sizing tools are listed as ASSET-R, CA-FPXpert, CEIS, SIZEEXPERT, SEER-M. The document was concluded that software estimating organization must be able to customize estimating tool to their own software development
environment. The historical data from past completed projects accurately provide the inputs that the software tool requires. The requirement of specially trained staff is also mentioned in this handbook. The use of two or more software estimating tools using different methodology has been recommended because parametric tool has following advantages

- Equations are based on previous historical development projects
- Outputs are repeatable and formulas can be analyzed
- They can be customized to fit the user’s environment
- They require minimal time and effort to use
- They are particularly useful in earlier phases of software development
- They are most frequently used by DOD agencies

Lum K., Bramble M., Hihn J., Hackney J., Khorrami M., Monson E., authored “Handbook for Software Cost Estimation”, Jet Propulsion Laboratory, Pasadena, California, USA[33]. This handbook described steps for software cost estimation for software projects ranging from a completely new software development to reuse and modification of existing software. This document also describes the historical data that needs to be collected and saved from each project to benefit future cost estimation efforts. This document covered the activities and support required to produce estimates from the requirement analysis phase through completion of the system test phase of software life cycle. The estimation methods listed are Historical analogy, Expert judgment, Model based and Rule of thumb. Following steps are recommended for software estimation.

- Gather and analyze software functional and programmatic requirements
- Define the work elements and procurements
- Estimate software size
- Estimate software development effort
- Schedule the effort
- Calculate the cost
- Determine the impact of risks
- Validate and reconcile the estimates via models and analogy
- Reconcile estimates, budget and schedule
- Review and approve estimates
- Track, Report and Maintain the estimates

The last step is elaborated here due to its importance and unique type of details. The purpose of this step is to check the accuracy of the software estimates over time and save for use in future software project estimates.

- Track the estimates to identify when, how much and why the project may be over-running or under-running the estimates. Compare current estimates and ultimately actual data with past estimates and budgets to determine the variation of the estimates over time.

- Document changes between current and past estimates and budgets

- In order to improve estimation and planning, archive software estimation and actual data each time an estimate is updates and approved, usually at each major milestone. It is recommended that the following data be archived:
  - Project contextual and support information
- Project name
- Software organization
- Platform
- Language
- Estimation method(s) and assumptions
- Date(s) of approved estimates(s)
- Estimates and actual size, effort, cost and cost of procurements by WBS work element
- Planned and actual schedule dates of major milestones and reviews
- Risks identified and their estimated and actual impacts.

The handbook elaborated the case study of estimating for ROM Flight Software.

2.1.1 Parametric Estimation Model

Zhihao, Chen in Ph.D. thesis titled “Reduced-parameter modeling for cost estimation models”, depicted details of estimation models [17]. Parametric software cost estimation models and their related calibration methods has been analyzed. Parametric estimation models (also referred as formal / empirical / algorithmic models) construct the mathematical / statistical models based on the parameters also called as independent variables, cost drivers or features. The model describe the nature and characteristics of the project. The cost, effort and schedule are determined based on algebraic relationships between a dependent variable and independent variables. These models are based on historic data and incorporate the characteristics for development processes, project environment, project information and project constraints. This research also claimed that combining
machine learning techniques and statistical techniques increased the effort prediction power of COCOMO and variability is decreased.

2.1.2 Parametric Estimation Model: COCOMO II

The widely accepted COCOMO II can be used for the following major decision situations[10].

- Making investment or other financial decisions involving a software development effort
- Setting project budgets and schedules as a basis for planning and control
- Deciding on or negotiating tradeoffs among software cost, schedule, functionality, performance or quality factors
- Making software cost and schedule risk management decisions
- Deciding which parts of a software system to develop, reuse, lease, or purchase
- Making legacy software inventory decisions: what parts to modify, phase out, outsource, etc
- Setting mixed investment strategies to improve organization's software capability, via reuse, tools, process maturity, outsourcing, etc
- Deciding how to implement a process improvement strategy, such as that provided in the SEI CMM

The COCOMO II is based on COCOMO series earlier introduced by Dr. Barry Boehm [12][18]. The primary objectives of the recent model are
To develop a software cost and schedule turned to the life cycle practices of the 21st century

To develop software cost database and tool support capabilities for continuous model improvement

To provide a quantitative analytic framework and set of tools and techniques for evaluating the effects of software technology improvements on life cycle effort, costs and schedules

COCOMO II has further three sub-models that are suitable at different stages of process model for particular project life cycle.

Application Composition Model addresses applications which are too diversified to be created quickly and composed of different components. It is also suitable for Prototyping process model. Object Point has been introduced as a sizing parameter to estimate effort.

Following steps are suggested to estimate by using Application Composition Model of COCOMO-II.

a. Assess object counts: estimate the number of screens, reports and 3GL components that will comprise this application.

b. Classify each object instance into simple, medium and difficult complexity levels depending on values of characteristics using scheme as in Table 2.1

c. Weigh the number in each cell using the following scheme. The weights reflect the relative effort required to implement an instance of that complexity level as indicated in Table 2.2
### Table 2.1 Classification of object

<table>
<thead>
<tr>
<th>Number of Views contained</th>
<th>For Screens</th>
<th>For Reports</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># and source of data tables</td>
<td>Number of sections contained</td>
</tr>
<tr>
<td></td>
<td>Total &lt; 4</td>
<td>Total &lt; 8</td>
</tr>
<tr>
<td>&lt; 3</td>
<td>Simple</td>
<td>Simple</td>
</tr>
<tr>
<td>3 – 7</td>
<td>Simple</td>
<td>Simple</td>
</tr>
<tr>
<td>&gt;8</td>
<td>Medium</td>
<td>Difficult</td>
</tr>
</tbody>
</table>

### Table 2.2 Complexity weights based on object type

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Complexity-Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Complex</td>
</tr>
<tr>
<td></td>
<td>Simple</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td></td>
<td>Difficult</td>
</tr>
<tr>
<td>Screen</td>
<td>1</td>
</tr>
<tr>
<td>Report</td>
<td>2</td>
</tr>
<tr>
<td>3 GL Component</td>
<td></td>
</tr>
</tbody>
</table>
d. Determine object-points, add all the weighted object instances to get one number, the Object-point count

e. Estimate percentage of reuse you expect to be achieved in this project. Compute the New Object Points to be developed in following equation.

\[(2.1)\]

f. Determine a productivity rate from the scheme depicted in Table 2.3

<table>
<thead>
<tr>
<th>Developers’ experience and capability ICASE maturity and capability</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROD</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

g. Compute the estimated person-months

\[(2.2)\]

Where \( PM \) - Effort in Person-Months

\( NOP \) - New Object Points
PROD - Productivity Rate

Early Design model used in the early stages of software project when very little may be known about the size of the product to be developed, the nature of the target platform, nature of personnel to be involved in the project or detailed specifics of the process to be used. The size is measured in the form of Function Points (FP) and it is converted into Source Line of Code (SLOC) and then KSLOC (1000 SLOC). In the Early Design model a reduced set of cost drivers are used. Each cost driver is rated in the six discrete qualitative values ranging from Very Low to Extra High.

The Early Design model involves exploration of alternative software/system architectures and concepts of operation. At this stage, not enough is generally known to support fine-grain cost and effort estimation. The corresponding COCOMO II capability involves the use of function points and a course-grained set of 7 cost drivers. The Early Design model uses KSLOC for size. Unadjusted Function Points are converted to the equivalent SLOC and then to KSLOC.

\[(2.3)\]

Where

PM - Person Months of estimated effort

A - Constant set as 2.5

BRAK - Breakage: Percentage of code thrown away due to Requirement volatility
SF - Scale Factors: PREC, FLEX, RESL, TEAM, PMAT

EM - Effort Multipliers:
   RCPX, RUSE, PDIF, PERS, PREX, FCIL, SCED

Post Architecture model is most detailed and intended to be used when a software architecture has been developed. Function Point is used as sizing parameters. Seventeen Cost Drivers also known as Effort Multipliers are used. They are grouped into four categories: product, platform, personnel and project.

The Post-Architecture model involves the actual development and maintenance of a software product. This stage proceeds most cost-effectively if a software life-cycle architecture has been developed; validated with respect to the system’s mission, concept of operation, and risk; and established as the framework for the product. The corresponding COCOMO II model has about the same granularity as the previous COCOMO and Ada COCOMO models. It uses source instructions and/or function points for sizing, with modifiers for reuse and software breakage; a set of 17 multiplicative cost drivers; and a set of 5 factors determining the project’s scaling exponent. These factors replace the development modes (Organic, Semidetached, or Embedded) in the original COCOMO model, and refine the four exponent-scaling factors in Ada COCOMO.

\[(2.4)\]

Where

PM - Person Months of estimated effort
A - Constant set as 2.5

BRAK - Breakage: Percentage of code thrown away due to Requirement volatility

SF - Scale Factors: PREC, FLEX, RESL, TEAM, PMAT

EM - Effort Multipliers
RELY, DATA, CPLX, DOCU, RUSE, TIME, STOR, PVOL,
ACAP, PCAP, PCON, AEXP, PEXP, LTEX, TOOL, SITE, SCED

ASLOC = Adapted Source Line of Code
AT = Percentage of components that are Automatically Translated
ATPROD = Automatic Translation Productivity

The quantitative values are also provided for qualitative values for Cost Driver and Scale Factors. These numeric values based upon the result of calibration done with data from 83 projects at CSSE of University of Southern California are presented in Appendix B.

Barry Boehm, A. Winsor Brown and Cyrus Fakharzadeh presented MBASE/RUP (Model Based Architecting and Software Engineering / Rational Unified Process) phases and applicability of COCOMO II Model at these phases as indicated in Figure 2.1
Figure 2.1 Software Project Life Cycle Phases and applicability of COCOMO II Models


2.1.3 Expert Judgment Method

Dr. Magne Jorgensen in his literature questioned regarding formal model application universally for accuracy. The website BESTWeb (Better Estimation for Software Tasks) has been maintained to support the research on software cost and effort estimation. This website provides a classification of 1200 research papers published in major journals and proceedings of conferences. His research deals with improvement on practices and Expert Judgment Estimation method.

Magne Jorgensen and Martin Shepperd have done systematic review of software development cost estimation studies [13]. In this review 304 related papers in 76 journals were identified and classified according to research topic, estimation approach, research approach, study context, and data set. IEEE transactions on Software Engineering was found dominant.
considering 57 papers with share 17% and number of citations made by other researchers. Estimation approaches were identified as regression, analogy, expert judgment, work breakdown, function point, classification and regression tree, simulation, neural network, Bayesian and combination of estimates. It has been identified that analogy and expert judgment based research interests are increasing. Regression based estimation approach was found dominating. The regression based includes most common parametric estimation model COCOMO.

Jorgensen, Magne authored a paper titled “A Review of studies on Expert Estimation of Software Development Effort” [70]. Extensive review of studies related to expert estimation of software development effort. He claimed that expert estimation is the most frequently applied estimation strategy for software projects and proved to be more accurate than formal estimation models because expert have domain knowledge which is not included in formal or algorithmic models. This paper presented expert estimation best practices guidelines which are listed below

i. Evaluate estimation accuracy, but avoid high evaluation pressure

ii. Avoid conflicting estimation goals

iii. Ask estimators to justify and criticize their estimates

iv. Avoid irrelevant and unreliable estimation information

v. Use documented data from previous development tasks

vi. Find estimation experts with relevant domain background and good estimation records

vii. Estimate top-down and bottom-up, independently of each other

viii. Use estimation checklists
ix. Combine estimates from different experts and estimation strategies

x. Assess the uncertainty of the estimate

xi. Provide feedback on estimation accuracy and development task relations

xii. Provide estimation training opportunities.

Magne Jorgensen analyzed in his review paper titled “Estimation of Software Development Work Effort: Evidence on Expert Judgment and Formal Models” [69]. The average accuracy of expert judgment-based effort estimates was better than the average accuracy of the models. Further it has been added that when expert judgment is combined with models, mean of estimation accuracy of the combination-based methods was similar to the best of that of the other estimation methods.

SIMBID is a tool developed at Simula Research Laboratory [9]. The purpose of this tool was conducting scientific experiments on bidding behavior under various conditions. System has provided facility to conduct bidding process rounds. Host were provided a facility to conduct bidding rounds and change or modify requirement slightly and bidders were provided with interface to participate and modify bids. The study helped to understand estimation process from bidders perspective.

Magne Jorgensen Authored paper titled “Practical Guidelines for Expert-Judgment-Based Software Effort Estimation” [21]. The terminology is introduced as below

pX :- Estimator believes that X percent probability of not exceeding p effort

Planned effort :- Effort used in plan, the effort may be derived from combination of goals related to accuracy, project management and work efficiency. To avoid overrun
risk one can use p80 estimates. If efficiency is guaranteed p30 can be used by project leader.

Effort to win :- Effort accepted by client or market while bidding

Following guidelines are issued for improvement in software estimation process:

- Don’t mix the estimation, planning and bidding
- Combine estimation methods
  - Useful Combination of estimation methods
    - Top-down and bottom-up methods
    - Analogy and linear regression methods
    - Expert judgment and formal methods
    - Expert judgments made by software professionals with different project experiences
    - Expert judgments made by software professionals in different roles

There is very important suggestion made that if estimator present a effort estimates, which kind of justifications are acceptable or unacceptable:

- Acceptable justifications
  - Reference to actual effort to similar project
  - Breakdown of the project into activities with reference to effort for performing similar activities in other projects and to the proportion of effort spent on unplanned activities in projects with a similar level of uncertainty
• Application of validated, organization-specific relationships between actual effort—preferably with documentation of the validation process

• Unacceptable justifications

  • Any estimates justified by belief, intuition, client’s cost constraints, reference to non-calibrated estimation model with no organization-specific data on its performance.

Finally it has been pointed from a case study that formal models are not accurate and more than 50% of project failed to estimate within the accepted range.

2.1.4 Expert Judgment or Formal Models

Magne Jorgensen & Barry Boehm in interview debated over the topic of estimation method[23]. Magne advocated Expert-Judgment where Barry defended strongly for formal models.

Magne’s arguments were as follows:

• In spite of massive effort and promotion, available empirical evidence shows that formal estimation models aren’t in much use. They are not succeeded so judgment-based estimation must be carried ahead with more research.

• First 10 out of 16 studies reported that judgment-based effort estimation led to more accurate estimates than using sophisticated formal models.

• Poor size measurement is another cause of inaccurate estimates. The relation between effort and size was never stable. Expert-judgment outperformed well in changing requirement and resource constraints.
• The doubt is raised about the relationship of variable with effort, for proper use of estimation model requires more investment in data collection, model calibration, education and training.

• Models are not objective. Input to the models is based on expert judgment and can consequently be impacted by outside pressure. Expert-judgment can be improved by various methods but it is not black box as stated by Barry.

Barry’s arguments were as follows:

• With reference to Standish Group survey only 30% projects surveyed successfully deliver their software within estimates. Parametric models contain good deal of information about which factors cause software costs to go up or down. Hence research must be continued with both the methods.

• Another question was raised that who is expert? does it mean experienced? it is very difficult for person to become an expert in all domains. Hence expert-judgment cannot be relied who is susceptible to pressure from client, organization etc.

• On the sampled population of projects, one-size-fits-all decision on using models has edge over experts in all situations, which don’t appear to be a good idea.

• Several evaluators use Basic COCOMO without cost driver and don’t get accurate estimates.

• One of the great advantage with parametric model is that it doesn’t modify its estimates when customers, managers or marketers apply pressure where expert easily get pressurized.

• Use of calibrated parametric model enables negotiation of the price of a software development contract to be driven by objective adjustment of the project’s size or
productivity-driver parameters, rather than by a contest of wills between self-described experts.

Finally they both agreed to improve both Formal / Parametric Model and expert-judgment based model for estimation. Final proposal was made that instead of competing each other for replacement and claiming more accuracy, both can be complementary to each other, hence combination of both models is strongly recommended.

2.2 EARLY STAGE ESTIMATION

Early stage estimation is based on software project document matching for identifying similar project from enterprise-wide knowledge-base and using estimates of similar project completed. Semantic similarity identification is main focus. The literature related to knowledge representation techniques, ontology engineering, lexical database Wordnet has been referred to make use of these in the context of effort estimation for software development.

2.2.1 Knowledge Representation

A knowledge representation is most fundamentally a surrogate, a substitute for the thing itself, that is used to enable an entity to determine consequences by thinking rather than acting, that is, by reasoning about the world rather than taking action in it.

- It is a set of ontological commitments, that is, an answer to the question, In what terms should I think about the world?

- It is a fragmentary theory of intelligent reasoning expressed in terms of three components:

  i. the representation’s fundamental conception of intelligent reasoning

  ii. the set of inferences that the representation sanctions, and
iii. the set of inferences that it recommends.

- It is a medium for pragmatically efficient computation, that is, the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance that a representation provides for organizing information to facilitate making the recommended inferences.

- It is a medium of human expression, that is, a language in which things about the world are described.

These roles create multiple, sometimes competing demands, requiring selective and intelligent trade-offs among the desired characteristics. These five roles also aid in clearly characterizing the spirit of the representations and the representation technologies that have been developed. This view has consequences for both research and practice in the field. On the research front, it argues for a conception of representation that is broader than the one often used, urging that all five aspects are essential representation issues. It argues that the ontological commitment that a representation supplies is one of its most significant contributions.

2.2.2 Ontology Engineering

Johannes Pretorius in his literature depicted an important distinction that should be drawn is between the notions of Ontology and ontology [73]. The difference is subtle but important. The former, written with a capitalized ‘O’, is an uncountable noun with no plural. It refers to the philosophical discipline that studies the nature of being. It addresses questions such as: ‘What is being?’ and ‘What characteristics do all beings have in common?’ When written with a lowercase ‘o’, and still considered in a philosophical sense, ontology refers to a system of categories (or frames of reference) that account for a certain view of the world. In this form it is a countable noun for which a plural form, ontologies, exists.
Barry Smith et.al. mentioned the core ideas of information systems ontology were developed by thinkers working completely from scratch[74]. It was John McCarthy who first recognized the overlap between work done in philosophical ontology and the activity of building the logical theories of AI systems.

McCarthy affirmed already in 1980 that builders of logic-based intelligent systems must first list everything that exists, building ontology of our world. The significance of the term grew, and as the disparate fields of knowledge engineering, conceptual modeling, and domain modeling began to converge and discover each other, so, too, did the range of variations in its meaning.

2.2.2.1 Approaches for Ontology Learning

Ontology Learning puts a number of research activities, which focus on different types of inputs, but share their target of a common domain conceptualization. It is a complex multidisciplinary field that uses the knowledge of natural language processing, data and web mining, machine learning and knowledge representation.

As per Alexander Maedche and Steffen Staab classification [52]:

1. Ontology learning methods from texts: Consist of extracting ontology by applying natural language analysis techniques to texts. The most well-known approaches from this group are:

   a. Pattern-based extraction: A relation is recognized when a sequence of words in the text matches a pattern. For instance, a pattern can establish that if a sequence of $n$ names is detected, then the $n-1$ first names are hyponyms of the $n^{th}$.

   b. Association rules: defined on the database field as follows: “Given a set of transactions, where each transaction is a set of literals (called items), an association rule is an expression of the form $X$ implies $Y$, where $X$ and $Y$ are sets
of items. The intuitive meaning of such a rule is that transactions of the database which contain X tend to contain Y”. Association rules are used on the data mining process to discover information stored on databases if we already have a rough idea of what we are looking for. The association rules have been used to discover non–taxonomic relations between concepts, using a concept hierarchy as background knowledge.

c. **Conceptual clustering**: Concepts are grouped according to the semantic distance between each other to make up hierarchies. The formulae to calculate the semantic distance between two concepts may depend on different factors and must be provided in these methods.

d. **Ontology pruning**: The objective of ontology pruning is to build domain ontology based on different heterogeneous sources. It has the following steps.

i. Generic core ontology is used as a top-level structure for the domain-specific ontology.

ii. A dictionary, which contains important domain terms described in natural language, is used to acquire domain concepts. These concepts are classified into the generic core ontology. Domain-specific and general corpora of texts are used to remove concepts that were not domain specific. Concept removal follows the heuristic that domain-specific concepts should be more frequent in a domain-specific corpus than in generic texts.

iii. **Concept learning**: A given taxonomy is incrementally updated as new concepts are acquired from real-world texts.

2. **Ontology learning from dictionary**: Bases its performance on the use of a machine-readable dictionary to extract relevant concepts and relations among them.
3. Ontology learning from a knowledge base: aims to learn an ontology using as source existing knowledge bases.

4. Ontology learning from semi-structured data: looks for eliciting ontology from sources, which have any, predefined structure, such as XML schemas.

5. Ontology learning from relation schemas: aims to learn an ontology extracting relevant concepts and relations from knowledge in databases.

2.2.2.2 Ontology Representation

When talking about representation languages, it is useful distinguish between different abstraction levels used to structure the representation itself: extensional, intensional, and Meta, respectively. The extensional level: this is level where the basic objects of the domain of interest are described, together with their relevant properties. The intensional level: this is the level where objects are grouped together to form concepts, and where concepts and their properties are specified. The meta-level: this is the level where concepts singled out in the intensional level are abstracted, and new, higher level concepts are specified and described, in such a way that the concepts of the previous level are seen as instances of these new concepts.

2.2.2.3 Framework for Ontology Representation

The framework discussed in [24] is based on three main classification criteria in presenting languages and formalisms. These classification criteria are concerned with the following three aspects:

- What to express : The first criterion takes into account that ontology is a generic term for denoting domain representation, but specific ontology languages may concentrate on representing certain aspects of the domain.
- **Class/relation**: used for referring to languages aiming at representing objects/classes/relations.

- **Action/process**: used for referring to languages that provide specialized representation structures for describing dynamic characteristics of the domain, such as actions, processes, and workflows. These languages may also incorporate mechanisms for the representation of the static aspects of the domain (e.g., objects and classes), but they usually provide only elementary mechanisms for this purpose, whereas they are much more sophisticated in the representation of the dynamic aspects.

- **Everything**: used for referring to languages that do not make any specific choice in the aspects of the domain to represent, and, therefore, may be in principle used for any kind of contexts and applications.

- **How to express**: This criterion takes into account the basic formal nature of the ontology languages. Classes of languages
  - **Programming languages**
  - **Conceptual and semantic database models**
  - **Information system/software formalisms**
  - **Logic-based**
  - **Frame-based**
  - **Graph-based**
  - **XML-related**
  - **Visual languages**
• Temporal languages

• How to interpret the expression: This criterion takes into account the degree in which the various languages deal with the representation of incomplete information in the description of the domain. Classes of languages:

• Single model. Languages of this class represent a domain without any possibility of representing incomplete information. An ontology expressed in this language should be interpreted as an exact description of the domain, and not as a description of what we know about the domain. In terms of logic, this means that the ontology should be interpreted in such a way that only one model of the corresponding logical theory is a good interpretation of the formal description. This assumption is at the basis of simple ontology languages.

• Multiple models (or, several models). Languages of this class represent a domain taking advantage of the possibility of representing incomplete information. An ontology expressed in this kind of language should be interpreted as specifying what we know about the domain. This point of view is at the basis of sophisticated ontology languages: for example, languages based on First-order logic represent a domain as a logical theory. It might be the case that this theory allows for different legal interpretations, and such interpretations correspond exactly to several models. Thus, languages based on First-order logic are classified as several models – ontology languages.

2.2.2.4 Ontology Languages

There are different languages and formalisms proposed in the last years for expressing and specifying ontologies. The languages for the specification of the intensional and the extensional levels are given below. Various languages are grouped according to the first
classification criterion mentioned in the previous section, namely the one regarding how to express.

- **Programming Languages**

Programming languages allow representation and manipulation of data in several ways and according to various paradigms. OO programming languages, such as Java, provide a sophisticated framework for modeling taxonomies of classes and manipulating their instances. Also, Languages like Prolog and LISP have always been used inside knowledge representation systems. LISP is at the base of various ontology representation languages like Ontolingua and OCML.

- **Frame Based Languages**

Frame based languages are based on the notion of frame. In a typical frame-based systems, constructs are available that allow to organize frames that represent classes into taxonomies. In a taxonomic hierarchy, each frame is linked to one or, in some systems, more than one parent frame. Through taxonomic relations, classes may be described as specializations of other more generic classes. Classes in a taxonomy inherit from their super classes features like slot definitions and default values. Examples of frame-based languages are Monolingual and Operational Conceptual Modeling Language (OCML).

- **Conceptual and Semantic models**

Semantic (or conceptual) models, introduced after the traditional data models (relational, hierarchical and network) were primarily schema design tools: a schema could be first designed in a high level semantic model and then translated into one of the traditional, record oriented models such as the relational one, for ultimate implementation. Examples of proposed semantically data models are The ER and Extended ER data model, FDM (Functional data model), SDM (Semantic Data Model).
Graph Based Models

Semantic networks are knowledge representation schemes involving nodes and links (arcs) between nodes. The nodes represent objects or concepts and the links represent relations between nodes.

Conceptual Graph is a bipartite graph containing two kinds of nodes, called respectively concepts and conceptual relations. Arcs link concepts to conceptual relations, and each arc is said to belong to a conceptual relation. There are no arcs that link concept to concept or relations to relations.

Topic Maps are a way of modeling a set of subjects, the relationships between them and additional information about them, with the same logics of a back-of-book index.

Logic Based Languages

Logic based languages generally express a domain-ontology, i.e. a conceptualization of the world, in terms of the classes of objects that are of interest in the domain, as well as the relevant relationships holding among such classes.

- **KIF**: Knowledge Interchange Format (KIF) is a formal language for the interchange of knowledge among disparate computer programs. Its main purpose is to facilitate the independent development of knowledge-manipulation programs.

- **CycL**: The purpose is to specify the largest common-sense ontology that should provide Artificial Intelligence to computers. Such a knowledge base would contain general facts, heuristics and a wide sample of specific facts. CycL is the language in which knowledge base is encoded. It is a formal language whose syntax derives from first-order predicate calculus and from LISP.
• **OWL - Web Ontology Language**: The Web Ontology Language (OWL) is a language for defining and instantiating Web ontologies. OWL ontology may include descriptions of classes, along with their related properties and instances. OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans. It facilitates greater machine interpretability of Web content. OWL is seen as a major technology for the future implementation of a Semantic Web.

Chris Biemann authored a paper titled “Ontology learning from text: A survey methods” expressed formal ontology is conceptualization whose categories are distinguished by axioms and definitions[50]. It is revealed from the paper that broadly different ontology are categorized as formal, terminological and prototype based. In this research to find similar document, terminological ontology is explored. The paper also presents that hierarchical structures in decision tree format can be used. When inserting new concepts, it is tested whether they fit best to the actual node or one of the daughter node. The tree is traversed top-down from the root until an appropriate position is found. But the success of finding similarity is dependent upon general nature of top level concepts, otherwise which may lead to wrong path in the beginning of the process. Use of small sub-tree of an existing Wordnet hierarchy is proposed to confine the search, which is used in this research but in another context.

Khaled Shaban presented research work Ph.D. thesis titled “A Semantic Graph Model for Text Representation and Matching in Document Mining”. This work has been concerned with text representation models, developing distance measures to estimate similarity between documents. Further this similarity between documents is utilized for document clustering, classification, information retrieval, information filtering and information extraction. The present research follows semantic similarity finding between software project documents, particularly semantic graph representation and matching of software project documents.
2.2.3 Latent Semantic Analysis

It is a theory of knowledge induction and representation that provides a method for determining the similarity of meaning of words and passages by analysis of large text corpora. LSA produces measures of word-word, word-passage and passage-passage relations that are well correlated with several human cognitive phenomena involving association or semantic similarity. LSA is closely related to neural net models, but is based on singular value decomposition, a mathematical matrix decomposition technique[46].

It is important to note that the similarity estimates derived by LSA are not simple contiguity frequencies, co-occurrence counts, or correlations in usage, but depend on a powerful mathematical analysis that is capable of correctly inferring much deeper relations (thus the phrase “Latent Semantic”), and as a consequences are often much better predictors of human meaning-based judgments and performance than are the surface level contingencies that have long been rejected.

LSA differs from some statistical approaches in two significant respects. First, LSA uses as its initial data not just the summed contiguous pair-wise (or Tuple-wise) co-occurrences of words but the detailed patterns of occurrences of very many words over very large numbers of local meaning-bearing contexts, such as sentences or paragraphs, treated as unitary wholes. Another way to think of this is that LSA represents the meaning of a word as a kind of average of the meaning of all the passages in which it appears, and the meaning of a passage as a kind of average of the meaning of all the words it contains.

LSA’s ability to simultaneously – conjointly - derive representations of these two interrelated kinds of meaning depends on an aspect of its mathematical machinery that is its second important property. LSA assumes that the choice of dimensionality in which all of the local word-context relations are simultaneously represented can be of great importance, and that reducing the dimensionality (the number parameters by which a word or passage is described) of the observed data from the number of initial contexts to a much smaller - but
still large - number will often produce much better approximations to human cognitive relations. This is this dimensionality reduction step, the combining of surface information into a deeper abstraction that captures the mutual implications of words and passages. Thus, an important component of applying the technique is finding the optimal dimensionality for the final representation.

Finally, LSA, unlike many other methods, employs a preprocessing step in which the overall distribution of a word over its usage contexts, independent of its correlations with other words, is first taken into account; pragmatically, this step improves LSA’s results considerably.

So, LSA is very efficient method to derive measures of the similarity of meaning due to these reasons.

- The meaning similarities derived closely match those of humans
- LSA's rate of acquisition of knowledge from text approximates that of humans
- These accomplishments depend strongly on the dimensionality of the representation.

In this way, LSA performs a powerful and, by the human-comparison standard, correct induction of knowledge. Thus, LSA is a fully automatic mathematical/statistical technique for extracting and inferring relations of expected contextual usage of words in passages of discourse.

2.2.4 Semantic Similarity of Document

Thomas L. Griffiths, Mark Steyvers, Joshua B. Tenenbaum authored paper titled as “Topics in Semantic Representation”[67]. The paper elaborated methodologies applied to find similarity between documents. The retrieval of concepts from memory is facilitated if one can infer the gist of sentence, conversation or documentation. The abstract computational problem underlying the extraction and use of gist is analyzed. Novel approach to semantic
representation is described in which word meanings are represented in terms of a set of probabilistic topic. The claim is made that topic model performs well in predicting word association and effects of semantic association and ambiguity on a variety of language processing and memory tasks. The suggestion is made that this provides a foundation for developing more richly structured statistical models of language.

Thorsten Brants, Reinhard Stolle, published paper titled “Finding Similar Documents in Document Collections”, Proc. of Third International Conference on Language Resource and Evaluation (LREC-2002), Spain, Jun.2002 [45]. This paper explains about finding similar documents in natural language document collections is a difficult task that requires general and domain-specific, deep analysis of documents and inference. Use of Probabilistic Latent Semantic Analysis for finding similar document has been used. The experiment is carried in the domain of photo-copier repair. The similarity types are identified as surface and conceptual. The surface similarity subtypes are same, similar and different. The conceptual similarity subtypes are same, similar, subset and different. This statistical model did not considered negation and relation between documents and depicted need for more semantic information.

Andreas Hotho, Alexander Maedche, Steffen Staab authored a paper titled “Ontology-based Text Document Clustering”[51]. The approach of applying background knowledge during preprocessing in order to improve clustering helped selection. The ontology-based heuristics is used for feature selection and aggregation. Multiple clustering results using K-Means are computed. The results may be distinguished and explained by corresponding selection of concepts in the ontology.

Mohamed Yehia Dahab, Hesham A. Hassan, Ahmed Rafea, “TextOntoEx:Automatic Ontology Construction from Natural English Text”[52]. The claim is made that TextOntoEx constructs ontology from natural domain text using semantic pattern-based approach which acts as a chain between linguistic analysis and ontology engineering. This tools analyses
natural domain text to extract candidate relations and then maps them into meaning representation to facilitate construction of ontology.

Vladimir Oleshchuk, Asle Pedersen, authored the paper “Ontology Based Semantic Similarity Comparison of Documents”[53]. The graph-based model reflecting semantic relationship between concepts has been applied for text analysis and comparison. The document footprints are compared with concepts from ontology. The claim is made that using ontology we can enhance the footprints by adding concepts that are not present in the original document that helped to find similar documents.

George A. Miller, who is Principal Scientist authored the paper titled “WordNet: A Lexical Database for English”[78]. In this paper it is mentioned that Wordnet provides a more effective combination of traditional lexicographic information and modern computing suitable for natural language processing and establishing semantic relationship. Wordnet is an online lexical database designed for use under program control. English nouns, verbs, adjectives and adverbs are organized into sets of synonyms represented in lexicalized concepts where semantic relations i.e. synonymy, antinomy, hyponymy, metonymy, troponymy and entailment, are linked.

Ted Pedersen, Siddharth Patwardhan, Jason Michelizzi authored a paper “Wordnet::Similarity-Measuring the Relatedness of Concepts”[54]. The paper demonstrated use of Wordnet::Similarity, software package to measure semantic similarity between pair of concepts or word senses. The six measures of similarity and three measure of relatedness are implemented which takes two concepts and return numeric values indicating similarity.
2.3 INTERMEDIATE STAGE ESTIMATION

This stage refers to modeling phase in generic life cycle context and elaboration phase with reference to Rational Unified Process. Literature related UML based estimation, Neuro-Fuzzy Application in combination with COCOMO II is referred.

2.3.1 Usecase Based Estimation

Bente Anda, Hege Dreien, Magne Jorgensen and Dag Sjoberg presented paper titled “Estimating Software development effort based on Use cases – Experience from industry”[26]. This paper was written to provide guidance to organization that want to improve their estimation process. Effort estimation based on use case model. The experiment has been carried on three projects and estimates were compared with expert estimates. Estimate by use case method were found close to the actual effort as well as expert’s estimates. It has been pointed out that the choice of structure for use case model has an impact on the estimates. The size was represented in terms of use case point. This method can’t reflect effort required for training and establishing a new programming environment. It has been pointed that use cases to be described at level of detail where each transaction is specified.

Kirsten Ribu in his thesis titled “Estimating Object Oriented Software Projects with Usecases”, revealed that sizing can be done by measuring size and complexity of usecases [28]. Since usecase writing is not standardized so the method based on usecase point could not become popular. The method of software estimation with usecase even if it is not written in full. The work also shows that how usecases can be sized alternatively. Extension to the existing counting of usecase point is proposed with simplification. The threat is identified with this method is that if usecase description test is incomplete then accuracy is deviated. The results are summarized with industrial case study and students project that as compared to the tools Optimize and Enterprise Architect, the estimates by this method was near to actual effort spent on project.
The comparison of estimates is depicted in Table 2.4.

### Table 2.4 Comparison of estimates

<table>
<thead>
<tr>
<th></th>
<th>Expert Estimate</th>
<th>UC estimate</th>
<th>UC without TCF</th>
<th>Optimize</th>
<th>EA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Effort</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13933</td>
<td>12600</td>
<td>14965</td>
<td>14528</td>
<td>14961</td>
<td>14965</td>
</tr>
</tbody>
</table>

Though results are promising the main question about universal applicability is unanswered. Just by one set of case study it is very difficult to believe the accuracy and usefulness. But this concept is quite promising and detail validation need to be done.

The framework for estimation based on use cases has been presented. The framework takes account of the idea of use case level, size and complexity, for different categories of system and does not resort to fine-grain functional decomposition. The front end for estimate professional is proposed to ease the burden of calculation. In this white paper, author agrees that estimates by expert judgment or wideband Delphi method and domain knowledge essentially available at hand before applying usecase based estimation[32].

#### 2.3.2 Classpoint Based Estimation

Class Point sizing is like Function Point. Which was conceived to estimate the size of object oriented products. Empirical validation is performed to assess usefulness and effectiveness of size measure to predict development effort. The class point size estimation is done divided in steps as identification and classification of user classes, evaluation of class complexity level, estimating total unadjusted class point and technical complexity factor calculation. The class point size measure CP1 and CP2 are proposed at different stages. CP2 measure resulted better performance with high degree of confidence, because attributes of classes were exploited in estimation of system size. The validation is done with graduate course students.
project, which is threat to validity. Industrial data need to analyzed for validation. It is also suggested that processing complexity adjustment does not significantly affect the prediction accuracy of the class point measures. Thus further investigations are needed to understand whether technical complexity factor is effectively useful or unadjusted Class Point measures should be adopted[29].

2.3.3 Neuro-Fuzzy Approach for software estimation

Neural network recognize patterns and adapt themselves to cope with changing environment. Fuzzy inference system incorporate human knowledge and inference mechanism and decision making. Integration of these two complementary approaches referred as neuro-fuzzy approach [44].

Conventional AI focuses on attempt to mimic human behavior by expressing it in language or symbolic form. AI system manipulates symbols in knowledge-base which provides basis for modeling human experts in narrow problem areas if explicit knowledge is available. The brain model as continuous-time nonlinear dynamic system in connectionist architectures that are expected to mimic brain mechanisms to simulate intelligent behavior. This book also elaborated fuzzy computing based on fuzzy set theory that provides systematic calculus to deal with such information linguistically and performs numerical computation by using linguistic labels stipulated by membership functions. The selection of fuzzy if-then rules forms the key component of fuzzy inference system that can effectively model human expertise in a specific application. Fuzzy systems lacks the adaptability to deal with changing external environments. Thus neural network learning concepts are incorporated in fuzzy inference systems, resulting in neuro-fuzzy modeling which is important technique in soft computing. The characteristics of neuro-fuzzy systems are listed as biologically inspired human expertise computing models, new optimization techniques & application domains, numerical computation, model-free learning, goal driven and fault tolerant real world applications.
Neuro-fuzzy Constructive Cost Model for software estimation carries feature of learning ability and interpretability complementing to COCOMO[41]. The expert can easily validate the model and capable of generalization. It allowed a input to be linguistic values. The detail learning algorithm is presented. In this paper COCOMO II is referred but data used for testing and validation is from COCOMO 81. The question is raised that whether this learning from data is applicable for projects being estimated today. The authors pointed that in COCOMO II literature, each cost driver value is represented by either of six discrete qualitative values termed as linguistic value ranging from Very Low to Extra High. The calibration resulted in single quantity for each linguistic value. In this context, if two projects whose effort multipliers values are at edges, represented by different discrete qualitative values resulted in drastically different values of effort.

In two similar hypothetical projects P1 and P2 cost drivers whose multiplier values are increasing, P1 is just below the lower limit of linguistic value such as “High” and P2 is just above this limit. Project having multiplier values are decreasing P1 is just above the lower limit of linguistic value such as “High” and P2 is just below the limit. The effort estimated for P1 was 203 person-months and P2 was 2886 person-months. This drastic difference of effort estimated in similar project was pointed. Further proposal is made that allowing continuous variable rating values as inputs eliminated serious problem due to its gradual transition. In neuro-fuzzy model claim is made that fuzzy sets with triangular member function have characteristics of linear interpolation [41]. In this paper the claim of accuracy is not justified in detail.

Each cost driver having discrete linguistic term is further represented by Fuzzy term with triangular membership function, This formed basis for further investigation. To answer following questions
- Why only triangular?
- How triangular function for all linguistic values are suitable?
- How even distribution of fuzzy terms is suitable?

For each cost driver universe of discourse is set from 1 to 6 representing fuzzy term Very Low, Low, Nominal, High, Very High and Extra High. COCOMO II literature has single quantity for each derived from calibration process.

What is the basis of assigning each fuzzy terms membership functions parameters or it is intuitively distributed?

In this paper triangular membership is used having general form as in equation 2.5 of the same is

\[
\frac{\text{XXX}}{\text{YYY}} \quad \text{(2.5)}
\]

How a, b and c values need to be derived, in this paper b is assumed as single value representation from COCOMO II literature. Hence the scope is found further to improve accuracy by adapting correct trend.

Soft-computing framework is presented to improve accuracy of formal estimation models[71]. Preprocessing Neuro-Fuzzy Inference System (PNFIS) is used to handle the dependencies among contributing factors and decouple the effects of the contributing factors into individuals. Neuro-fuzzy bank is used to calibrate cost drivers or contributing factors. Integrating fuzzy logic, neural network and algorithmic models into one scheme, expert knowledge and project data is used for good interpretability and robustness to imprecise and uncertain inputs. Researchers answered the adaptation problem raised in their earlier paper.
Other problem remained unsolved. Proper handling of dependencies among contributing factors is essential to improve estimation accuracy. Utilization of expert knowledge is also the key to handle this problem. PNFIS solves this problem by encoding expert knowledge in fuzzy rule in generalized form. Preprocessing neuro-fuzzy inference system and neuro-fuzzy bank both are following Adaptive Neuro-Fuzzy Inference System architecture[44].

PNFIS encodes expert knowledge in fuzzy rule in generalized form. Preprocessing neuro-fuzzy inference system and neuro-fuzzy bank both are followed by Randall et.al. and explained in his article that the notion of Knowledge Representation can best be understood in terms of five distinct roles that it plays, each is crucial to the task at hand [72].

2.4 SURVEYS RELATED TO SOFTWARE ESTIMATION

Moløkken-Ostvold, Kjetil, Jørgensen Magne et.al. conducted survey on software estimation in Norwegian industry during the year 2004[16]. In this paper, the surveys conducted earlier were also referred except the CHAOS report published by Standish Group, because the research method was not described. The information was collected through structured interviews with senior managers from 18 different companies and project managers of 52 different projects. Analysis was done about estimation approach, effort estimation accuracy and bias, schedule estimation, delivered functionality and other estimation related information. Result of the survey has revealed that effort overruns were 41% where 76% of projects used more than estimated effort, 62% of projects completed after schedule, expert estimation was dominating method. The research team finally concluded that the survey related to only Norwegian industry, those have smaller projects in hand compared to companies in UK and USA. The choice of estimation method and level of estimation accuracy and bias were found stable, being independent of year, technology and location. A possible reason for this observation is that alternative estimation approaches have failed to provide evidence that their use increase estimation accuracy. The further research focus was
set to see how different process improvement efforts or technological changes affect estimation performance in Norway and other countries.

Da, Yang et.al. conducted survey on software cost estimation in Chinese software industry during early 2007[68]. It is revealed from study that only human expert can estimate accurately with little and vague information available during initial stages of project. The aim of this survey was to analyze magnitude and distribution of software cost estimation accuracy. Survey revealed the importance of software cost estimation for the purpose of budgeting, tradeoff and risk analysis, software project planning and control along with software process improvement analysis. It is evident from literature that estimation carried out at project proposal stage and later revised during analysis and design phase. This survey reported that 3% projects had estimated values accurately, 47% of projects were over estimated by 5% and 22% projects were overestimated by 20%. Percentage of overscheduled project was more than that of effort. Large software projects had lower estimation accuracy, more prone to overrun and higher variance of estimation than smaller projects. Expert consultation and Analogy were found as dominant estimation method. Analysis of survey revealed that 62% estimators were found neutral about the satisfaction, 22% reported unsatisfied and 16% reported satisfied. It was pointed that the estimations were made at lifecycle stages such as initial project proposal (57%), feasibility study (67%), requirement(74%), design(36%) and implementation(18%).

Major causes of inaccurate estimation were listed as:

- Unclear and volatile requirements
- Pressure from clients to change estimation results
- Unavailability of enough resources and historical data
- Lack of appropriate estimation methods, tools and training
Lack of involvement of stakeholders in estimation process

Predefined bidding requirement

Business pattern

Three factors, as barriers and difficulties in the application of software estimation were identified, which were

- high adoption cost of estimation models,
- insufficient investment for improving software cost estimation
- insignificant benefits from using estimation models.

Finally researcher’s team concluded that new estimation technology must be introduced and acceptance as well as usage of these can be improved by improving process along with performance expectancy, effort expectancy, social influence and facilitating conditions.

The survey conducted by Chen revealed that many projects overrun their estimated effort, cost and schedule and landed up in abandoning[17]. This was supported by Standish report which was criticized by Jorgensen et. al.

The research work has focus on reducing the deviation at initial stages. AI model for software estimation are thought for the same reason.