International Journal of Computer Applications Technology and Research
Volume 5–Issue 9, 612-618, 2016, ISSN:-2319–8656

A Review of Agile Software Effort Estimation Methods

Samson Wanjala Munialo.
Department of Information Technology
Meru University of Science and Technology
Meru - Kenya

Geoffrey Muchiri Muketha
Department of Information Technology
Murang'a University College
Murang'a - Kenya

Abstract: Software cost estimation is an essential aspect of software project management and therefore the success or failure of a software project depends on accuracy in estimating effort, time and cost. Software cost estimation is a scientific activity that requires knowledge of a number of relevant attributes that will determine which estimation method to use in a given situation. Over the years various studies were done to evaluate software effort estimation methods however due to introduction of new software development methods, the reviews have not captured new software development methods. Agile software development method is one of the recent popular methods that were not taken into account in previous cost estimation reviews. The main aim of this paper is to review existing software effort estimation methods exhaustively by exploring estimation methods suitable for new software development methods.

Keywords: Lines of codes, Cost constructive model, Function point, Agile, software effort estimation.

1. INTRODUCTION

Demand for more functionality, higher reliability and higher performance has resulted to higher competitiveness among software developers. To stay competitive, software developers need to deliver software products on time, within the budget and to the agreed level of quality. Most projects fail due to planning issues such as cost, time and requirements specifications. A study on software projects in 2012 by Standish shows that 43% of projects were challenged and 18% failed due to over budget, late delivery and less than required features or functions [1]. This illustrates the necessity for reliable software development method and software cost estimation method.

For faster and quality delivery, software vendors are moving from structured development methods where requirements are well known in advance to agile development which welcomes customer changing requirements at later stages of software development [2]. The shift from traditional development methods to agile is due to the high cost of affecting changes request by users at later stages of software development. Agile encourage changes, therefore decreasing the cost of change and reduce the overall development cost. Agile make this possible as a result of simple design, collective ownership, continuous testing and short releases cycles.

Many estimating methods have been proposed since 1950's and many studies have evaluated their effectiveness. The most popular traditional cost estimation methods are Expert judgment, Analogy, Wideband Delphi, Source lines of codes, Function points [3], Object points and Cost constructive model (COCOMO) [4] [5]. However, they are not effective when dealing with agile software estimation. Estimating agile software is a problem due to varying requirements and incremental development. This prompted the introduction of cost estimation methods such as planning poker [6] in 2003 which is one of the most popular agile estimation methods. However, planning poker depends on expert experience on previous projects and its estimates are specific to the team; another team may estimate different story point for the same project

Recently other methods such as Bayesian Belief Network, AgileMOW[7] and Constructive Agile Estimation algorithm [8] were introduced to deal with uncertainty and iterative nature of agile software development. Estimation of effort and cost depends on accurate estimation of the software size which helps to predict the project scope. Apart from size, other indicators such as project complexity factors are considered when estimating effort. Therefore, a reliable software cost estimation method must include critical cost indicators for more accurate estimation. The main objective of this paper is to discuss existing software cost estimation methods including their features and situations where they are applicable. This paper is organized in 6 sections which include introduction, background, Traditional effort estimation methods, agile effort estimation methods, discussion and conclusion.

2. BACKGROUND

Software estimation is a critical component of software project management. Cost overruns increased from an average of 56% in 2004 to 59% in 2012 in sampled software projects while time overruns increased from 71% in 2010 to 74% in 2012 [1]. More accurate estimation helps software developers to gain profit and customers to be more satisfied. On the other hand, high costing can lead to lost profit by losing bidding while low costing can lead to cost overruns and poor quality of end product. Software estimation comprise of estimating software cost, size, effort and time required to develop the software [9]. Software developers require an effective software cost estimation model to facilitate project planning and eventually successful implementation of a software project.

Agile software development method is one method that provides a challenge to existing software cost estimation techniques. Agile software development is based on iterative development where requirements evolve through collaborations. Scope is continuously adjusted throughout the project and new tasks are discovered [10]. It emphasizes on working software, customer collaboration, response to change on demand and does not support well defined requirements like the traditional waterfall method. All these challenges make most of the existing software cost estimation techniques to appear limited when dealing with agile software.

Software developers have had the interest of estimating accurately the cost of developing software products. The first methods were only based on software size using lines of codes or function points to estimate the cost. Currently other cost drivers such as process

factors and human factors have been included in estimation methods to improve on software estimation accuracy [7]. However, demand for new functionalities, quick delivery of software such as mobile applications established a need for new software development methods. Currently other features such software re-use, component based development, distributed systems and iterative development are common features in software engineering industry. Evolution in software engineering industry provided a challenge to software estimations researcher to come up with methods that will estimate more accurately.

3. TRADITIONAL COST ESTIMATION METHODS

Cost estimating models are classified as non-algorithmic and algorithmic. Non-algorithmic methods estimation relies on experts who have experience on similar previous projects while algorithmic methods use parametric in their estimation.

3.1 Non-Algorithmic estimation methods

Most non-algorithmic cost estimation techniques are based on analytical comparison with previous similar projects and expert experience [10]. Most popular methods in terms of recent publications in this group are expert judgment, Analogy, top-down, bottom-up, price-to-win and Wideband Delphi.

3.1.1 Expert judgment

Expert Judgment technique is the most frequently applied cost estimation method where experts are responsible for estimating the size and cost of a software. This method is based on the project manager experience in similar software projects. Expert cost estimation method is helpful when there is limitation in finding data and gathering requirements [10] [11] [12]. Expert judgment is prone to human errors and biasness. Its success is based on expert judgment that is expert experience may differ from one expert resulting to varying estimates on the same type of project. However, it is helpful in small and medium sized software project and when the development teams and software attributes have not experienced significant changes as compared to previous projects.

3.1.2 Analogy technique

Analogy technique estimation is done according to the actual cost of one or more completed projects that are similar to the new project to be estimated [8][10] [11]. Estimation can be done at the total project level or at sub system level. The strength of estimation by analogy is that the estimate is based on actual project experience and estimation can be done in the absence of an expert. However, it does not take into consideration the extent of other relevant cost factors in the previous project such as the environment and functions which may differ with new project cost factors [13]. In addition, a lot of past information about past projects is required whereas in some situations there may be no similar projects developed in the past to compare with.

3.1.3 Price-to-win, Bottom-up and Top-up

Price-to-win estimation method is based on customer budget instead of software parameters or features. Example is when a customer is willing to pay for 6 persons-month and the project estimate is 8 persons-month then estimation is done as per the customer ability to pay. This may cause delays and force developers to work overtime [13]. Price-to-win method helps in getting the contract but it generally causes cost and time overruns.

Bottom-up estimation method estimates by separating each software component then summed to give the overall estimate for the product. It is possible only when the requirements and design of the system are known at an early stage of software development [11] [14]. While top-down method established an overall estimate for the project then the system is sub-divided into its functional components which are then estimated based on the overall estimate [13] [14]. The design and requirements must be well defined to partition software to its component.

3.1.4 Wideband Delphi

Wideband Delphi method is a cost estimation technique where effort and cost are estimated centered on team consensus. It is done by getting advices from experts who have extensive experiences in similar projects. Wideband Delphi technique was introduced by Barry Boehm and John Farquher in 1970s. It uses work breakdown structure as the basis for estimating project size, effort and cost [12] [15]. This method emphasizes on consultations, communication and interaction among participants.

Participants include customer representatives and technical team members that will be involved in development of the software product. Each member estimates for each task and identify changes and missing assumptions in work breakdown structure. Members with high or low estimates are asked to justify, and then members revise the estimates. The cycle repeats until when estimators agree on the estimates. The coordinator collects estimates from team members and assembles the tasks and estimates into a single final task list.

Wideband Delphi depends on team members experience and agreement among members and thus it is not appropriate method when applied to a software project that is unfamiliar to members [14] [15]. Furthermore, it is a preferred method when requirements are well defined and therefore, cannot work for software development methodologies where requirements are not clear. However, it encourages collaboration among estimators. Lastly, the technique is simple to apply and supports consensus-based estimates. Even though Wideband Delphi estimates are consensus-based, experts may be biased, optimistic or pessimistic in their estimation given that this method cannot be quantified.

3.2 Algorithmic software cost estimation methods

These models use a formula to calculate the software cost estimate [13]. They rely on a combination of related cost factors which are input to mathematical equation to do the estimation. Most common algorithmic software cost estimation methods includes Source line of codes (SLOC), Object points, Function-Point(FP)[3], Constructive Cost Model-I (COCOMO-I) [4] and Constructive Cost Model-II (COCOMO-II) [5].

3.2.1 Source line of codes (SLOC)

Source line of codes is a size metric that illustrates the number of program statements and data definition but does not include comments. SLOC is the earliest cost estimation method used to estimate the size of FORTRAN and assembly language which are line based programming languages. SLOC uses historical data of a previously completed project of the same size whose SLOC was computed before then compared with the actual one to estimate project size. The size estimate is eventually used to estimate the project scope, effort and cost [10]. SLOC is dependent on the programming language and therefore cannot compare different

programming language lines of codes. Source line of codes cannot estimate the size of non-procedural languages and software complexity is not taken into consideration when estimating size.

3.2.2 Function Point Analysis

Albrecht's introduced Function point analysis method in 1983 [3] which had better estimation than source lines of codes. Function point is a size metric that quantifies the size and complexity of a software system with regard to functions that the system will deliver. Function count is arrived at by counting the basic software components which include external inputs, external outputs, external inquiries, logical internal files and external interfaces. Each of the function is weighed by complexity factor ranging from low, average to high [3] [11] [14]. Each function component is multiplied with a respective complexity level then summed up to give Function Count (FC).

Function point can be applied at requirement specification or design phase of system development [14]. Furthermore, function point is independent of language or methodologies used in software development [3]. Lastly, Non-technical user can easily understand the method. However, Function point cannot be used in situation where requirements are not clear such as in agile software development.

3.2.3 Object Point

It estimates the size of software based on number and complexity of objects [11] [17]. The objects are screens, reports and 3GL components. The steps for estimation effort using object point include: counting the number of objects, classification of objects (simple, medium, average), weight objects with regard to difficulty as shown in table 1.

Table 1: Classification of objects weight

Object type	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3 GL components			10

Object point is determined by adding all the weights of object instances to get object point count.
Estimate percentage re-use then compute the overall object points (NOP) where, NOP = (Object Point) * (100-% reuse)/100

Furthermore, developers' productivity is weighted from low to highest then effort is estimated by dividing net object point by productivity [11]. It is easy to apply object point method at any stage of software development but on the other hand requirements must be well defined. Object point only considered 3GL and 4GL factors and thus cannot apply to current programming languages.

3.2.4 Constructive Cost Model (COCOMO)

COCOMO models were proposed by Barry Boehm [4]. These methods use parameters which were derived from previous experiences about software projects for estimation. Due to COCOMO methods popularity various studies have extended COCOMO framework to develop cost estimation methods with an aim of improving software estimation accuracy. The 4 COCOMO methods are simple COCOMO, Intermediate COCOMO, Detailed COCOMO and COCOMO II.

Basic COCOMO computes software effort and cost as a function of program size expressed in thousands lines of codes (KLOC) using the formula:

$$\text{Effort} = a(\text{KLOC})^b$$

Where a and b are complexity factors which are assigned weights according to software project complexity as shown in table 2.

Table 2: Complexity factor weights

Model	A	B
Organic (Simple)	2.4	1.05
Semi-detached(Average)	3.0	1.15
Embedded (Complex)	3.6	1.20

With the advancement in software development methods and environment, basic COCOMO was not able to capture all relevant cost factors in its estimation. Therefore, intermediate COCOMO was released to include emerging software attributes in their computation of software estimates.
Intermediate COCOMO uses Kilo lines of codes as in basic COCOMO but it includes EAF (Effort adjustment factors) which includes subjective assessment of products, hardware, personnel and project attributes [5] [13]. Effort adjustment factors consider a set of four factors, with each factor having a number of attributes. The complexity factors are hardware, personnel, project and product with the following attributes.

- Hardware attributes: Run-time performance constraints, Execution time constraint, Memory constraints, Volatility of the virtual machine environment and Required turnabout time.
- Personnel attributes : Analyst capability, Software engineering capability, Applications experience, Virtual machine experience, Programming language experience
- Project attributes: Use of software tools, Application of software engineering methods and required development schedule.
- Product attributes: Required software reliability, Size of application database and Complexity of the product, required reusability.

Each of the 17 attributes is rated on a 6 point scale that ranges from very low to very high. Based on the rating, an effort multiplier is determined and the product of all effort multipliers results is an *effort adjustment factor* (EAF). Typical values for EAF range from 0.9 to 1.4 The intermediate COCOMO model takes the form EFFORT = a* (KLOC)b * EAF.

Another COCOMO version is detailed COCOMO which incorporates all characteristics of intermediate COCOMO on each step of software development process (Analysis, Design, coding and testing). The 17 attributes are used in each step to estimate software development effort [5] [10] [11] [13].

COCOMO-II was introduced in 1997 is an extension of intermediate COCOMO. It predicts the amount of effort based on Person-Month (PM) in the software projects [5][13]. It uses Thousands lines of code or function point as the size metrics and the number Effort adjustment factors attributes were increased by 5 to 22 attributes. The Usage of COCOMO II is very wide and its results usually are more accurate. The 5 additional effort adjustment factors are:

- Precedents'(PREC)- Previous experience of the organization
- Development Flexibility (FLEX) –Degree of flexibility in development process.
- Risk resolution (RESL)- Extent of risk analysis carried out.
- Team cohesion (TEM)- How well development team knows each other
- Process maturity (PMAT)- Process maturity of the organization

COCOMO II formula takes the same format as intermediate COCOMO formula of estimating effort [5] [10] [11].

All COCOMO methods capture a wide range of parameter when estimating the cost of a project. So far COCOMO methods are the most popular methods with clear results. The use of COCOMO requires clear and well defined requirements [16]. However, a lot of data is required to estimate effort and the model is presented as black box to the user. However, COCOMO methods are challenged when requirements are not clear and when the project is subject to user request changes at later stages of software development.

3.2.5 Other software cost estimation methods

In recent years researchers have attempted to introduce more cost and effort estimation techniques to improve on estimation accuracy. One of the methods is Bayesian Belief Network which estimate software effort by forecasting software cost when information about the past and present is incomplete, vague and uncertain [18]. It includes a network of probabilities that captures the probabilistic relationship between variables in historical data [11]. The advantage of this method is not being dependent on knowing exact historical data. On the other hand, it requires knowledge of related parameters of previous project to be used in estimation.

The other method is Neural Networks which is based on the principle of learning from examples. Neural network use back propagation trained feed forward network to estimate software development effort [17] [11]. The network is trained with a series of inputs from previous projects to predict the effort of the current project. Neural network provided a more accurate estimate compared to other methods but it depends on data from previous projects.

4. AGILE COST ESTIMATION METHODS

The emergence of agile methods has presented many opportunities and challenges. One of the challenges is estimating the effort of developing agile software. Although traditional methods are used to estimate effort for agile software, they provide inaccurate results. Agile is a popular development method as it emphasize on collaboration with customer, communication among developers, rapid delivery of software and change of requirements on demand [20] [21]. Popular agile methods are Extreme programming, scrum, crystal, Feature driven development and learn development.

Some of the challenges of estimating agile methods include work assigned to a team and not an individual, emphasis is on collective effort and work is quantified in terms of effort rather than time and changing requirements on demand. Various studies were done in recent years and have come up with cost estimation methods suited for agile with the most popular one being planning poker [6]. Planning poker is a non-algorithmic method and is simple to implement. Other agile estimation methods introduced so far are constructive agile estimation algorithm [8] and AgileMOW [7] although their accuracy has not yet been calibrated by other researchers.

4.1 Planning Poker

Planning poker is an estimation method that is based on collaboration and consensus among team members like Wideband Delphi technique. It was initially proposed by Greening in 2003 and popularized by Cohn in 2005 [6] for agile software development such as scrum. Planning poker session is done at the beginning of an iteration of agile development involving a team of developers from different disciplines.

Each member in the team is given a deck of planning poker cards with values preferably Fibonacci sequence (1, 2, 3, 5, 8, 20, 40, 100) representing story points or ideal days. The nonlinear sequences reflect less uncertainty with smaller units and greater uncertainty when dealing with greater units [6]. A story in agile development is a brief description of functionality as viewed by the user or product owner. Story points are a relative unit of measure used to estimate the story size by taking into account effort, complexity and risk [19]. On the other hand, ideal days estimate a story with regard to the number of days or time it will take to translate a story to a system function or feature.

When a story has been fully discussed, each member privately estimates a story by selecting a card to represent the estimate. All cards are revealed at the same time and if the estimates are the similar then it becomes the agreed estimate. If not, high and low estimates are justified and discussed further. Then each member selects a card after the discussion and cards are revealed again. The process is repeated until consensus is achieved [19]. Two main reasons why planning poker is an effective way of estimating agile software is that it involves a team of experts from different disciplines who collaborate and justify their estimations to come with better results as compared to one expert providing estimate especially when there is high uncertainty and missing information.

4.2 Constructive Agile Estimation Algorithm

Constructive agile estimation algorithm was introduced in 2009 [8]. The algorithm uses vital factors namely project domain, performance, configuration, data transaction, complex processing, ease of operation and security which are weighed then incorporated in the estimation.

Constructive Agile Estimation algorithm divides estimation process into two phases called early estimation and Iterative estimation. The purpose of early estimation is to identify the initial scope just enough to draw the initial budget. Iterative estimation is done at the start of an iteration to include new requirements. In both cases story point is used to estimate the size of a feature as described by the user. Vital factors are identified on the grade of low, medium and high using Fibonacci series then multiplied to story point to get the final estimate.

Constructive Agile estimation algorithm identified factors that are critical in determining software effort but in addition people factors are also important especially in agile where collaboration and teamwork is an important ingredient for successful completion of a software project but they are not included in this algorithm.

4.3 AgileMOW

AgileMOW was introduced to estimate the cost of developing web applications using agile methods [7]. This method uses both expert judgment and algorithm to estimate effort. AgileMOW uses people and environment attributes described in COCOMO II which are aligned to agile manifesto. Factors used in this method include communication skills, proximity of team, feedback, courage, management skills, technical ability, reliability, ease of use and early delivery. The method use web objects to estimate size of a web application and people factors are weighed. Effort expressed in person-month is computed by multiplying web application size and weighted people and environment factors.

On main advantage of AgileMOW is that it identifies factors that align to the principles of agile software which focuses on communication and interaction. However, it cannot estimate the cost of other software rather than web application. Lastly, the method only focused on people factors whereas other factors such as product and process factors are also important when estimating agile software effort.

5. DISCUSSION

None of software cost estimation method is better or worse than the other, each has its own strength and weakness which are complementary to each other. Furthermore, software estimation methods are specific to a specific type of project or development method or software to be developed [5] [15]. Estimation methods such as Function point analysis, Object point and COCOMO are suitable when developing software in which requirements are fully known upfront such waterfall method. In contrast, these methods are challenged when requirements keep on changing such as in agile which require an estimation method that adapt to changes in such as planning poker estimation method.

Different situations and development environment determine the appropriate software cost method to be used. There are situations where accuracy in estimation is critical then a more accurate method should be employed, in other instance, winning a contract is important therefore, price-to-win becomes the most appropriate method [11]. Furthermore, small projects can easily be estimated using expert judgment but when the project becomes larger it requires more technical estimation method such as analogy and COCOMO. In addition, availability of data from previous project provides an opportunity to use analogy estimation method.

Several cost drivers should be considered to estimate software effort and cost. The most common cost driver among all estimation methods is the software size. Effort and cost can be estimated directly upon estimating the software size using one of the software size metrics such as source lines of codes, function point and object point. Agile size estimation is done using story point. Size is also used together with other factors to estimate software development effort when using most of algorithmic estimation methods. Therefore, software project managers must understand the key attributes in a project to identify an estimation method that will estimate accurately.

Each effort and cost estimation method has strengths and weaknesses based on the capabilities of the method. Table 4 shows a summary comparison of popular cost estimation methods.

Table 4: Comparison of software effort estimation method

Method	Strength	Weakness
COCOMO	- Clear results - Independent on programming language	- Much data required - Requirements must be clear. - Not adopted to changes in requirements
Function point	- Clear results - Independent on programming language	- Requirements must be clear. - Not adopted to changes in requirements
Expert	- Less data required - Adopt to special projects	- Its success depend on the expert
Analogy	- Based on similar project experience - More accurate	- Information about past projects is required - Historical data may not be accurate
Price-to-win	- Gets contract	- High overruns
Top-down	- Faster to implement System level focus - Minimal project details required	- Less stable - Less detailed
Bottom-up	- Based on detailed analysis - Support project tracking	- Difficult to estimate early in the life cycle - Time consuming
Wideband Delphi	- Reduced biasness by involving a team of experts	-Its success depend on the expert -Not adopted to changes in requirements
Planning Poker	- Adopt to changes in requirements - Reduced biasness by involving a team of experts	- Its success depend on team of experts - Estimation is relative to a team.

6. CONCLUSION

This paper provided a comprehensive overview of existing software cost estimation models describing their strengths and limitations. It is important for the software project manager to understand key factors relevant in estimating the cost of software and situations where an estimation method will be appropriate. No existing model can estimate the cost of software development with a high degree of accuracy, therefore the study of software cost estimation is necessary to improve on estimation accuracy.

With the emergence of new software development methods and techniques, future work will be to identify key estimation indicators in new software development methods and devise new cost estimation method.

7. REFERENCES

[1] The Standish group, 2013, Chaos Manifesto: Think big, Act small, The Standish Group Intenational.

[2] Coelho, E.,Basu, A.,2012, Effort Estimation in Agile Software Development using Story Points, International Journal of Applied Information Systems.

[3] Albrecht, A.J., Gaffney, G.E., 1983, Software Function, Source lines of Codes, and Development Effort Prediction: A Software Science Validation, IEEE Trans Software Engineering.

[4] Boehm, 1981, Software Engineering Economics, Prentice Hall.

[5] Boehm, B.W. et al, 2000, Software Cost Estimation with COCOMO, Prentice-Hall.

[6] Cohn, M.,2006, Agile Estimating and Planning, Pearson Education

[7] Litoriya, R., Kothari, A.,2013, An Efficient Approach for Agile Web Based Project Estimation: AgileMOW, International Journal of Computer Science and Computer applications.

[8] Bhalerao, S., Ingle,M.,2009, Incorporating Vital factors in agile estimation through algorithmic method, International Journal of Computer Science and Computer applications.

[9] Ziauddin, Tipu, S.K., Zia, S., 2012, An Effort Estimation Model For Agile Software Development, Advanced Computer Science and Its Applications.

[10] Khatibi, V., Jawawi, D.N., 2010, Software Estimation Methods: A review, Journal of Emerging Trends in Computing and Information Sciences.

[11] Borade, G, J.,Khalker, R,V.,2013, Software project effort and cost estimation techniques, International Journal of Advanced Research in Computer Science and Software Engineering.

[12] Gandomani, T.,Wei, T., Binhamid, K., 2014, Software Cost Estimation Using Expert Estimates, Wideband Delphi and Planning Poker Technique, International Journal of Software Engineering and its applications.

[13] Kumari, S., Pushkar, S., 2013, Performance Analysis of software cost Estimation methods: A Review, International Journal of Advanced Research in Computer Science and Software Engineering.

[14] Sharma, N., Bajpai, A., Litoriya, R., 2012, Software Effort Estimation, International Journal of Computer Science and Applications.

[15] Stellman, A., Greene, J.,2005, Applied Software Project management, O'Relly Media.

[16] Basha, S.,Dhavachelvan, P.,2010, Analysis of Empirical Software Effort Estimation Model, International Journal of Computer Science and Information Security.

[17] Bogdan, S., 2003, Software Development Cost Estimation Methods and Research trands", Computer Science.

[18] Angyan Y., Charlottesville, 2003, A Bayasian Belief Network approach to certifying Reliability of COTS software systems", Annual Reliability and maintainability Symposium, IEEE.

[19] Calefato, F., Lanubile, F., A Planning Poker Tool for Supporting Estimation in Distributed Agile Development, The Sixth International Conference on Software Engineering Advances

[20] Cao, L., 2008, Estimating Agile Software Project Effort: An empirical study, Association of Information Systems AIS Electronic Library(AISeL), Americas Conference on Information Systems.

[21] Schmietetendorf, A., Kunz, M., Dumke, R, 2008, Effort Estimation for Agile Software Development Projects, Proceedings 5th Software Measurement European Forum, Milan