Software Engineering Estimation - Results from an MLR

abhijit.mahal2@mail.dcu.ie, faizan.shah3@mail.dcu.ie, oluwaseun.aderibigbe2@mail.dcu.ie, zohaib.ali2@mail.dcu.ie, noah.coynetyrrell2@mail.dcu.ie

Abstract {{Oluwaseun}}

The successful implementation of project planning, cost control, and resource allocation depends highly on software engineering estimates. However, these traditional estimation methods; the algorithmic approaches of COCOMO, Function Point Analysis, and Use Case Points, as well as the non-algorithmic methods of expert judgement and analogy-based estimation, have been known to suffer from problems such as rigidity, prejudice, and range. Our findings show that these advanced technologies can enhance the precision of estimates by optimising input parameters, reducing bias, and using historical data appropriately.

The literature also reveals that these innovations can reduce estimation errors by 15-40%, which shows how applying fuzzy logic in COCOMO II, using ML-enhanced Function Point forecasts, and making context-aware adjustments for Use Case Points can enhance the accuracy and flexibility of the estimation. Moreover, the bias detection capability and automated analogy matching of AI can help non-algorithmic methods improve their purity while maintaining the accuracy of their outputs.

1. Introduction {{Abhijit Mahal}}

Software engineering estimation plays a crucial role in project planning, cost analysis and resource allocation. These parameters can help pave the way for effective strategic decision-making before starting a project or provide crucial estimates in maintaining existing software. For decades there have been majorly two types of methods that have defined the current software estimation techniques, these being algorithmic and non-algorithmic methods.

Algorithmic methods like FPA, COCOMO and UCP leverage the use of mathematical equations and historical data to present an estimate in terms of effort and cost. These methods with their well-structured framework require a lot of early project analysis, historical data and fine-tuning of the parameters, they still stand the test of time and are still being innovated to this date[1]. On the other hand, non-algorithmic data such as

Expert Judgement, Analogy based estimation and The Delphi methods require human knowledge and experience to estimate cost and effort. These methods allow flexibility but still manage to have biases, susceptible to inconsistencies and lack the use of invaluable historical data.

With advancements in recent years in fields such as GenAI and Machine learning, they have opened many paths to innovation in these existing fields. Machine learning can help analyse, categorize and humanize historical data while still providing invaluable insights towards the calibration of parameters, adjusting and fine tuning the models and also pointing out the misjudgements in project analysis. While these advancements are still recent, the world around is adapting to it very rapidly and integrating it with the existing technology to take it to the next level.

In this research paper we incorporate the use of GenAI and ML to further innovate and potentially evolve the existing methods and models by using the current advancements in technology and mitigate risks, reducing errors and making software engineering estimation more adaptive. This paper explores the current limitations of methods on both algorithmic and non-algorithmic methods, discussing the challenges, potential benefits and investigates how innovations in generative AI and machine learning can refine and improve these methods.

2. Research Methodology {{Zohaib}}

This paper investigates software estimating approaches, including their functionality, procedure, benefits, limitations, and contemporary developments. It includes both algorithmic methods like COCOMO, FPA, and UCP, as well as non-algorithmic methods like expert judgement and analogy-based estimation.

Relevant sources were gathered from IEEE, ScienceDirect, and Springer, with significant books and fundamental papers identified by snowballing. Along with new developments like machine learning and generative artificial intelligence, the advantages, disadvantages, and workings of each approach are examined.

A comparison of traditional and AI-enhanced models utilising historical data reveals increases in accuracy and adaptability, emphasising the impact of modern technology on software estimation.

3. Analysis

3.1 COCOMO {{Abhijit Mahal}}

3.1.1 Introduction

The Constructive Cost Algorithm or commonly known as COCOMO was founded in 1981 by Barry W. Boehm which was established as a software engineering estimation method. COCOMO algorithm would provide time and effort estimates for software development using inputs like KLOC(thousand lines of code), 15 cost driver attributes and calibration constants that would be initialised depending on the state of the project[1][3]. These states were initialised depending on the understanding of the project, amount of experience and the openness to further refinements and changes in regards to the functional specifications was taken into consideration. These 3 modes were organic, semi detached and embedded modes respectively where organic being low experience, low understanding and open to changes in the future and embedded being high understanding, moderate experience and fixed functionality[1].

3.1.2 Comparing COCOMO I & II

COCOMO II introduced in the 1990's solved many issues regarding its predecessor such as non-sequential development, reuse and object-oriented approach[1] [2]. It also provided different modes depending on the project requirements where it allowed users to estimate effort and time either for a project in its initial stages of development called the early design model or a project in it's development or maintenance phase thus simplifying and improving the pre-existing COCOMO model where only estimation were only available in the initial stages of software development[1] [2]. COCOMO II also increased the number of cost drivers, sub models and effort equation exponents as compared to COCOMO I[4].

COCOMO II uses SLOC(source line of code) and UFP(unadjusted functional points) which is a measure of the size of a software project based on its functionality as inputs

[2]. Along with the two input metrics, scale factors which represent the overall scale of the project which is based on the project's characteristics, such as how familiar the team is with similar projects and cost factors representing the factors influencing the overall cost of the project [2][4]. These factors have a rating level which represents the impact of these factors on the overall development of the project are also known as effort multipliers [2].

3.1.3 COCOMO II With A Neuro-Fuzzy Approach

Now that we know how COCOMO II improved over COCOMO I, let's talk about how we can further innovate the idea of software development estimation to make sure that we are getting the most out of the software estimation models and especially COCOMO II. One idea that was proposed was the use of Fuzzy-logic and NN to refine the input values for the COCOMO II model based on the software requirements[3]. This in turn increases the accuracy of the overall model[3]. The Fuzzy-Logic COCOMO II method enhances the traditional COCOMO II method by incorporating fuzzy logic to handle uncertainty in the input values. The architecture consists of 17 EM's, 5 SF's and 1 SS where the output was made in person-months [2][3].

These attributes go through the process of Fuzzification that converts these software attributes into fuzzy variables using membership functions, ratings like 'Low', 'very low', 'high', etc. are assigned to each attribute[2][3]. Membership function allows the distribution of absolute values like 0's and 1's to a more distributed format like values between 0's and 1's [3]. These can be distributed in a triangular structure which is a more linear format or a trapezoidal format that adds weights to certain ranges of values [3]. Then Fuzzy rules are applied to the Fuzzy variables where the variables are used in IF-THEN statements that allow to describe the relationship between the input and the output variables. This is done to describe the quality or property in a variable in a linguistic format rather than in numerical values [2]. An example of this for easier explanation can be "IF DATA is Very High, THEN effort is Very High". Then the final process of Defuzzification is applied where the fuzzy output is converted into a crisp value using techniques like Mean of Maximum, Center of Area and First of Maximum. These final crisp values obtained from defuzzification are used to calculate the final effort multipliers and scale factors using the COCOMO II equations[2][3].

3.2 Function Point Analysis {{Faizan}}

3.2.1 Introduction

In software engineering, accurately estimating the size, cost, and effort required for software development is significant for project success. Functional Point Analysis (FPA) is a way to measure how big and complex a software system is based on what it does for users [8]. It was first created by Allan J. Albrecht at IBM in 1979 and later improved by the International Function Point User's Group (IFPUG) [8][9]. FPA breaks down a software system into different parts, like inputs, outputs, files, and user interactions, and gives each part a weight based on how difficult it is. This helps estimate the effort needed to build software, track progress, compare projects, manage costs, and ensure the software meets business goals [8].

3.2.2 Understanding Function Point Analysis: Key Components and Calculation Process

FPA focuses on five main types of functional components, which are essential for understanding how the system interacts with users and other systems [7][9]. External Inputs (EIs) for user data entry (e.g: login forms), External Outputs (EOs) for system-generated outputs (e.g: reports), External Inquiries (EQs) for retrieving information (e.g: database searches), Internal Logical Files (ILFs) for internal data storage (e.g: customer databases), and External Interface Files (EIFs) for interacting with external systems (e.g: payment gateways) [7][9]. Each of these components is assigned a weight based on its complexity, ensuring that more complex functionalities receive greater importance in the overall calculation [7].

The process of performing Function Point Analysis follows a structured, step-by-step approach. The first step is identifying functional components by listing all the EIs, EOs, EQs, ILFs, and EIFs present in the system. For example, if the system includes a login form, a report generation feature, and a search function, these would be classified as EIs, EOs, and EQs, respectively [7]. Next, each component is assigned a weight depending on its complexity. A simple input (EI) might have a weight of 3, a complex output (EO) might have a weight of 7, and a basic data file (ILF) might have a weight of 7, while a more complex one could have a weight of 15 [7]. Once weights are assigned, the Unadjusted Function Points (UFP) are calculated by multiplying the number of each component by its corresponding weight and summing the results. After computing the UFP, an adjustment is made to account for project-specific factors such as performance requirements, security needs, and user experience. These factors are rated on a scale from 0 to 5, and the total sum is used to calculate an adjustment factor. The formula for this adjustment is: Adjusted Function Points (AFP) = UFP × VAF [5]. This final calculation provides a more accurate measure of the software's size, which can then be used to estimate effort, cost, and time [5].

3.2.3 Advantages of Function Point Analysis

One of the key advantages of FPA is its user-centered approach, as it measures software based on the functionality it provides to users rather than focusing on the technical details. This makes it easier for non-technical users to understand and evaluate the software's complexity [7]. Additionally, FPA can be used in all project phases, including early development stages, where it helps estimate effort, duration, and costs. This makes it a useful tool for project managers to plan and control software development effectively [7]. FPA also improves communication between teams, as it provides a standardized way for developers, analysts, and managers to discuss project complexity [8].

3.2.4 Limitations of Function Point Analysis

Despite its advantages, FPA has several limitations. One of the biggest drawbacks is its subjectivity, as different analysts may classify functions differently, leading to inconsistent results. According to paper [7], the measurements can vary by up to 30% within the same organization [7]. Additionally, FPA is a time-consuming process, requiring significant effort to collect function point data. It can take a certified analyst up to “five days” to complete an evaluation [7]. FPA still relies on outdated weight values that have remained unchanged for decades, making them less relevant for modern software development [7].

3.2.5 Enhancing Function Point Analysis

One major improvement comes from using Machine Learning models to estimate function points more efficiently. ML techniques like Support Vector Regression (SVR) and Random Forests can predict function points with high accuracy [9]. The study found that ML models could achieve similar results to traditional methods like High-Level FPA and Simple Function Points (SFP) while needing fewer details as input [9]. For example, instead of requiring five different data points, some ML models performed just as well using only two, such as total transactions and data files [9]. This means that project managers and developers can get quick and reliable function point estimates without needing a deep technical analysis, saving time and effort [9].

Taking automation even further, FPA-EX, a system that uses Generative AI to automatically extract function points from software requirement documents [10]. Unlike traditional methods, which require manual work by FPA experts, FPA-EX analyzes text using Large Language Models and can identify function points with high precision

[10]. It also uses a technique called ConceptAct Prompting (CAP), which helps the AI understand technical terms and apply domain knowledge accurately [10]. This reduces human effort significantly, shifting the role of FPA analysts from manual calculations to reviewing and verifying results [10].

3.3 Use Case Points {{Zohaib}}

3.3.1 Introduction

Software effort estimation is a critical component of project planning that ensures development teams spend time and resources efficiently. Gustav Karner's Use Case Points (UCP) estimating technique for object-orientated software development was introduced in 1993 [11]. It is based on Function Point Analysis (FPA), but it is modified for systems in which use case modelling is the primary technique of defining requirements. In order to generate various mathematical equations that take into consideration sizing, effort, and cost estimation. UCP is calculated using four main variables:[12]

1) Unadjusted Actor Weights (UAW)

2) Unadjusted Use Case Weight (UUCW)

3) Technical Complexity Factor (TCF)

4) Environment Factor (EF)

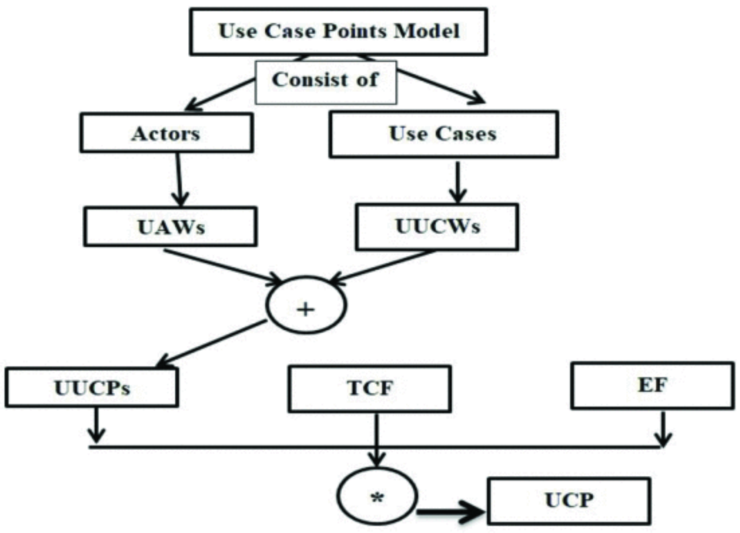

Figure 1. Traditional Use Case Points model. [32]

Unadjusted Actor Weight (UAW)

UAW measures the complexity of actor's in a project. Actor describes any individual who is participating in the use of a system based on their activity. Every actor involved in software development is weighted according to their categories. The human is the most complex category with a total weight of three, while the simplest is the external system that defines the API, and the average or intermediate weight is the external system that uses standard protocols. The table below defines the actor's complexity category:[13][12][11]

Category	**Description**	**Weight**
Simple	External system using an API	1
Average	External system communicating through a standard protocol	2
Human	Interaction involving a human user	3

Unadjusted Use Case Weight (UUCW)

Each use case is weighted to determine the level of complexity in software development. The amount of transactions or activities in a use case indicates its complexity. The category is comparable to UAW complexity, which includes simple, average, and complex. The weights of each category vary, with the complex category having the largest weight. The table below illustrates the details:[13][16]

Category	Description	Weight
Simple	Contains 3 or fewer transactions	1
Average	Includes 4 – 7 transactions	2
Human	Contains more than 7 transactions	3

Unadjusted Use Case Point (UUCP)

To estimate the UCP project's score, the estimation needs to be modified based on a set of use cases and actor weight. The UUCP should incorporate all of the project's functional needs. As a result, the non-functional requirements and environmental situation are crucial. The following formula is used to count between UUCW and UAW: [15][12][11]

$$UUCP = UAW + UUCW$$

Technical Complexity Factor (TCF)

The system size takes into account non-functionality as a factor influencing software development. Each factor's scale is weighted from 0 to 5, with the lowest being insignificant and the highest being crucial. All of this scale must be multiplied by the weight assigned to each factor. The Tfactor is calculated by multiplying the weight with the examiner's assessment. The table below displays each factor's weight in detail. [15]

Tfactor	Description	Weight
T1	Distributed systems	2
T2	Specific performance objectives	2
T3	End-user efficiency	1
T4	Complex internal processing	1
T5	Code reusability	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portability	2
T9	Easy to change	1

T10	Concurrency	1
T11	Security features	1
T12	Access for third parties	1
T13	Requires special user training facilities	10

The TFactor of a software size is calculated by 0.6 and 0.01 as stated below:

$$TCF = 0.6+(0.01xTFactor)$$

Environmental Complexity Factor (ECF)

The environmental factor in software development has a considerable impact on size measurement, particularly the productivity value. This is made up of eight EFactors of different weights. These range from development experience (1.5 weight) to programming language (-1 weight). The table below provides further information.[12]

Efactor	Description	Weight
E1	Experienced with a software development process	1.5
E2	Application experience	0.5
E3	Object-orientated experience	1
E4	Lead analyst capability	0.5
E5	Motivation	1
E6	Stable requirements	2
E7	Part-time workers	-1
E8	Difficulty of programming language	-1

The EFactor of this estimation software is shown below:

$$ECF = 1.4+(-0.03xEFactors)$$

Use Case Points (UCP)

The next stage in software effort estimation is to determine the UCP by adding UUCP, TCF, and ECF, which are shown in the equation below:

$$UCP = UUCP \times TCF \times ECF$$

Karner (1993) recommended using a fixed productivity rate of 20 person-hours per UCP to estimate the required effort, which results in the effort calculation shown in the equation below:

$$Effort=UCP \times 20Effort$$

Productivity is then determined as the ratio of effort to the total UCP, as shown in the equation below:

$$\text{Productivity}= \frac{effort}{UCPproductivity}$$

As a result, UCP and project person-hours are used to estimate effort. Karner (1993) estimates total development effort with a fixed productivity rate of 20 person-hours per UCP.[11][14][15]

3.3.2 Advantages of Use Case Points

Use cases provide multiple advantages in software development. They are commonly used for effort estimating, supporting teams in predicting workload, cost, and project scope [16]. They also enhance requirement analysis, which makes it easier to figure out customer expectations and ensure consistency [16]. Use cases help with automation by generating tests, analysing security, and modelling systems [11]. They improve software testing by assisting with the definition of test cases, estimating testing effort, and ensuring traceability of requirements and tests [16]. Furthermore, use cases enhance reuse in Software Product Lines (SPL), enabling better organisation of common and variable features [16]. They also help with formal verification, error detection, and review processes [16]. They aid in quality enhancement by refining requirements and assuring early testing [16]. Additionally, UCP can automate estimation, minimising manual work [17]. They can also help anticipate implementation time based on organisational averages, but this is dependent on use case documentation consistency [17]. Additionally, UCPs provide a pure measure of software size that is independent of team capability or experience, making them useful for objective estimates [17]. However, the success of use case estimation is determined by the quality and consistency with which they are written.

3.3.3 Limitations of Use Case Points

Despite their benefits, use cases have disadvantages. All use cases must be written first for estimation, which causes planning to be delayed. They are not suitable for iteration-based development because imprecise transaction rules result in inconsistent estimations. Certain technical issues interfere with estimation accuracy [17].

Use cases may not be suitable for all systems, particularly those without external actors. Lack of standardisation leads to variations in detail and terminology, complicating analysis. Costs for training and documentation rise with each iteration. Estimation is impacted by format and quality inconsistencies, making predictions inaccurate [16].

Testing Use Cases needs strict constraints, which adds significant cost. Reuse in Software Product Lines (SPL) is problematic due to increased modelling effort and improper uncertainty management [16].

Researchers have noted inconsistent formatting, granularity challenges, and high adoption costs since 1997, and these challenges continue to this day. In recent years, modelling effort has become a significant disadvantage [16].

3.3.4 Contemporary Innovations in UCP

Recent research has proposed innovative methods for improving the accuracy and suitability of the UCP approach to estimating software effort. Ngafidin (2023) did a thorough performance evaluation on a website builder project and discovered that including improved adjustment elements, such as technical difficulty and team experience, considerably enhances estimation precision. According to their findings, accuracy is improved, and the chance of underestimating or overestimating effort is decreased when UCP models are customised to particular project variables. [15]

Similarly, Azzeh (2020) conducted a systematic assessment of UCP techniques, highlighting the significance of contextual calibration. They emphasised that typical UCP models frequently fail to account for different project settings, resulting in unreliable predictions. To solve this, they recommended incorporating machine learning techniques to dynamically alter UCP settings based on previous project data. This adaptive technique improves prediction accuracy by using previous estimation patterns and contextual variables.[18]
These studies indicate a growing trend towards more exact and adaptable UCP models, which are becoming increasingly important in modern software development settings that require accurate resource and time management. The emphasis on adaptive algorithms and contextual calibration shows a trend away from static UCP models and towards dynamic, data-driven techniques, opening the path for more accurate software effort estimation practices.

3.4 Non-Algorithmic Methods and Generative AI {{Oluwaseun}}

3.4.1 Introduction

Software estimating directly affects budget control, timetable planning, and resource allocation. Precise assessment of the work needed for software projects offers many difficulties. As such, the focus in software estimation is not only on reaching perfect accuracy. Rather, the goal is to provide a structure that supports well-informed judgement on the viability and extent of a project.

These factors make it challenging to accurately gauge the effort required for the project! All of this makes nailing down an accurate estimate quite the challenge. As noted "Estimation techniques are used to calculate approximate results; made uncertain or incomplete data usable."[20]

This paper will investigate important non-algorithmic estimate methods, their benefits and drawbacks, and possible directions of improvement through Generative AI (GenAI) solutions. The aim is to consider the changing scene of software project estimation and propose means of enhancement for the current estimate systems.

3.4.2 Contextual Effort Estimation

This is one of the most prevalent non-algorithmic approaches. This technique assesses a project’s distinctive traits to gauge its effort requirements, drawing heavily on expert experience and historical context instead of rigid mathematical models. Boehm (1984) suggested that "expert-driven estimations tend to be reliable for small projects but lack scalability when applied to large, complex systems."[21].

Non-algorithmic effort estimation methods often require significant human judgment and domain expertise, making them more suited for projects where past data may not be directly applicable. By leveraging AI analytics, teams can enhance expert knowledge with data-driven insights, better-identifying inconsistencies and improving estimation accuracy.

3.4.3 Using Generative-AI

Artificial intelligence analytics in the contextual effort and estimation helps to improve expert knowledge through data-driven insights. By examining an abundance of past project data, GenAI can find trends and relationships that human estimators might miss. This skill provides more complex knowledge about possible difficulties and resource requirements, therefore changing the contextual estimating process.

Generative artificial intelligence can also enable improved and more efficient communication among project participants. Acting as a cooperative tool helps to guarantee that, when using non-algorithmic approaches, every team member is in line. An AI-driven platform might, for instance, let team members provide their opinions on project needs, timeframes, and hazards, so compiling a complete knowledge bank from which future projections can draw reference.

3.4.4 Price-to-Win Estimation

Organisations often apply price-to-win estimation in competitive contracting contexts. This estimation approach emphasises what customers are ready to spend rather than only computing the real expenses required to finish a project. Consider it as a strategic game; the goal is to find the best balance between staying competitive and paying running costs [22].

In this scenario, the complexity of the correct estimate results from the several demands at hand, which might make simple estimation techniques more difficult. This complex nature calls for careful balancing to guarantee project bids are competitive without compromising project feasibility or profitability.

Including strategies like function point analysis (FPA) helps to reduce the risk by improving the accuracy of price-to-win estimates. This method helps create size-based forecasts using historical data and expert judgement in concert. Based on user needs, function point analysis offers a methodical methodology for estimating the usability of a software project, therefore enabling companies to develop better-informed pricing policies[23].

While non-algorithmic methods concentrate on using human intuition and experience, algorithmic methods depend mostly on past data and established models to guide project projections. Combining the two points of view provides a complete estimation process that helps project teams create pricing recommendations reflecting both internal project capability and market dynamics.

3.4.5 Parkinson's Law Estimation

Parkinson's Law says, "Work expands to fill the time available for its completion"[23][24]. In software development contexts, where teams sometimes use the entire allocated time for projects regardless of the actual effort needed, this aphorism is especially pertinent. Strict deadlines have particular difficulties even if they can inspire higher production. Tight deadlines can put too much pressure on teams, which might cause possible compromises in work quality and high team member stress. Furthermore, applying arbitrary time limits could force teams to use shortcuts or cut back on thoroughness, therefore unintentionally generating technical debt that needs to be corrected after delivery.

3.4.6 Improving non-algorithmic methods using integration

Although non-algorithmic estimate methods have different advantages and drawbacks, their importance becomes especially important in situations marked by inadequate historical data or a greater demand for human understanding. Although these techniques usually lack the accuracy provided by algorithmic models, their natural adaptability can be used to create a good software estimating framework.

GenAI into these non-algorithmic estimate methods enables radical enhancements:

1. Predictive Views: By mining vast past project data, artificial intelligence can predict possible difficulties and provide more precise work estimates. This data-centric viewpoint strengthens the human components of contextual assessments, therefore improving the basis of knowledge at hand for making decisions. [25]

2. Communication: GenAI provides a collaborative environment for many team members involved in estimating activities, therefore ensuring that every opinion is heard and valued. This method creates a whole picture of project requirements by combining ideas from several experts.[26]

3. Ongoing Education: After a project ends, AI tools can generate a feedback loop capturing results and lessons discovered. This feedback would over time hone the estimations by providing particular suggestions for development.[26]

3.4.7 Conclusion

Non-algorithmic estimating methods offer flexible structures for organising software projects. Each contextual effort estimation, price-to-win estimation, and Parkinson's Law estimation have advantages and drawbacks. Although their accuracy is less than that of computational techniques, their importance in software estimation becomes clear, particularly in situations marked by little historical data or a great demand for human judgment.

Organisations should give consistency in application top priority if they want these non-algorithmic approaches to be as effective as they ought. Generative AI's capabilities will help to boost insights, simplify teamwork, and promote a culture of ongoing development, thus producing more accurate project estimates.

Adopting a hybrid strategy that combines the benefits of human knowledge with AI-driven analytics will become ever more crucial as the terrain of software development changes. Organisations that do this can improve their estimation accuracy, get feasible projects, and provide clients with premium output.

3.5 Non-Algorithmic Methods {{Noah}}

3.5.1 Introduction

Software estimation is a critical aspect of project management, influencing budgeting, resource allocation, and scheduling. Non-algorithmic estimation methods, such as expert judgment, analogy-based estimation, the Delphi method, and top-down or bottom-up approaches, are widely used due to their adaptability. However, these methods have notable limitations. This paper explores the effectiveness of non-algorithmic estimation methods and their potential improvements through AI-driven innovations.

3.5.2 Expert Judgment

Expert judgment relies on experienced professionals who estimate effort based on prior projects. It is well-suited for small projects and unique situations but is prone to bias and inconsistency. Boehm (1984) noted that expert judgment is reliable for small-scale projects but struggles with larger ones.[27] Cross-validation with historical data and peer reviews can enhance accuracy.

3.5.3 Analogy-Based Estimation

Analogy-based estimation compares a project to similar past projects. This method is effective when reliable historical data is available. Sophatsathit (2014) found that analogy-based methods can outperform algorithmic models under the right conditions.[28] Maintaining structured project histories and employing statistical techniques can improve reliability.

3.5.4 Delphi Method

The Delphi method involves multiple rounds of expert input to refine estimates. This method reduces bias but is time-consuming. Boehm (1984) recommended Delphi for projects with high uncertainty. [27] Anonymizing inputs and limiting rounds can mitigate its weaknesses.

3.5.5 Top-Down and Bottom-Up Estimation

Top-down estimation starts with an overall effort estimate, broken down into smaller tasks, while bottom-up estimation aggregates component estimates. Kitchenham & Linkman (1997) found hybrid approaches to be the most reliable.[29] Initial top-down estimates should be refined with bottom-up methods for better accuracy.

3.5.6 Enhancing Non-Algorithmic Estimation with AI

Generative AI (GenAI) can enhance non-algorithmic estimation methods by reducing bias and improving knowledge retrieval.

AI-Augmented Expert Panels

- AI can analyze historical data to provide preliminary estimates for expert refinement.
- Bias detection models can flag consistent over- or underestimation trends.

Kocaguneli et al. (2012) demonstrated that ensemble learning improves software effort estimation by aggregating diverse models, reducing uncertainty and bias. [30] This approach supports AI-powered Delphi Optimization, where AI can analyze expert responses and suggest refined midpoint estimates for enhanced consensus-building.

AI-Driven Analogy-Based Estimation

- AI can identify comparable projects based on technology stack and project scope.
- Automated post-project analysis can improve future estimates by learning from past discrepancies

Wen et al. (2012) suggest that AI-driven methods, such as regression and ensemble learning, significantly improve estimation accuracy by learning from past projects. This aligns with AI-driven analogy-based estimation, where automated analysis enhances project comparisons for better predictions.[31]

AI-Powered Delphi Optimization

- AI can analyze expert responses to detect outliers and suggest midpoint estimates.
- Automated summarization of expert discussions can speed up the process.

AI-Enhanced Top-Down and Bottom-Up Estimation

- AI can dynamically adjust estimates based on real-time project progress.

3.5.7 Drawbacks of Non-Algorithmic Methods

Despite their advantages, non-algorithmic methods have limitations:

- **Scalability:** Struggles with large projects.
- **Consistency:** Estimates vary between different experts.
- **Data Utilization:** Lacks historical data integration compared to algorithmic models.
- **Flexibility vs. Precision:** Adaptable but may lack accuracy.

3.5.8 Conclusion

Non-algorithmic estimation methods remain valuable, especially in cases requiring expert intuition. However, integrating AI-driven innovations can mitigate biases and enhance reliability. A hybrid approach that combines non-algorithmic methods with algorithmic techniques is recommended for optimal software estimation accuracy.

4. Limitations of Research {{Faizan}}

Approaching and writing a research paper for the first time, our group faced several challenges. Since we were new to research, we had little experience, which made it harder in the beginning to know the best way to organize our paper and find relevant sources. We also had time constraints due to miscommunication early on in the research, which made it difficult to conduct thorough research. Another challenge was finding our direction, we weren't sure exactly which aspects of the topic to focus on, and at times, the group members were not on the same page about where the research should go.

As we were researching, we realized that finding papers on gen AI and machine learning was particularly difficult, as most available research was very recent. This limited our access to more in-depth or foundational studies. We had to rely on free online content, which further restricted our access to more detailed studies. Analyzing data was another challenge, as some of the information was complicated and hard to understand. We came across studies with different methods and terms that were new to us. It took us extra time to figure out what information was important and how to explain it clearly in our paper. Overall, it was a new and valuable experience that helped us learn a lot about the research process.

5. Directions for Future Research {{Abhijit}}

The development of hybrid models to combine the strengths of algorithmic and non-algorithmic methods with artificial intelligence and machine learning. Algorithmic methods that we discussed like COCOMO and FPA could further be developed to investigate the effectiveness of hybrid models for large scale projects and also more in depth research for comparing the traditional models with the hybrid models.

AI tools can also be developed to mitigate biases in non-algorithmic methods like analogy-based estimation and expert judgement. This can possibly be achieved by identifying patterns of over or underestimation through historic data and also real time bias correction during the estimation process.

6. Conclusion {{Noah}}

Software engineering estimation is a central aspect of successful project planning that bridges budgeting to the deployment of resources. Both the traditional algorithmic approaches like COCOMO and Function Point Analysis and the traditional non-algorithmic approaches like expert judgment and analogy-based estimation have served to supply the necessary frameworks to estimate effort and cost. However, while the techniques have well established their presence and had time to mature, they are also plagued by their unique sets of weaknesses. Algorithmic models are rigid and require a lot of historical information, while the traditional approaches struggle with the problem of subjectivity, variability, and scalability.

With the arrival of AI and machine learning, the landscape of software estimation is revolutionizing at a very rapid speed. AI-driven techniques can complement current models by optimizing the input parameters, detecting biases, and drawing upon vast amounts of historical data to deliver more accurate predictions. Generative AI can potentially automate and improve non-algorithmic approaches of estimation to the degree of making techniques driven by experts more efficient and evidence-based. Although the technologies are being built out, they are a very promising stepping stone to adaptive and reliable techniques of estimation.

This paper contrasted traditional AI-powered estimation methodologies with their strengths, weaknesses, and areas of potential enhancement. With the growing incorporation of AI into software development, research must look to the next generation of models that reconcile the inflexibility of the algorithmic approach with the flexibility of the expert approach. In this sense, software estimation can overcome the inflexibility of models and become a smart, dynamic, and precise way of dealing with complex software projects.

6. References {{Faizan}}

[1] Chirra, S.M.R., Reza, H.: A Survey on Software Cost Estimation Techniques. Journal of Software Engineering and Applications. 12, 226–248 (2019). https://doi.org/10.4236/jsea.2019.126014.

[2] Malik, A., Pandey, V., Kaushik, A.: An Analysis of Fuzzy Approaches for COCOMO II. 68–75 (2013). https://doi.org/10.5815/ijisa.2013.05.08.

[3] Huang, X., Ho, D., Ren, J., Capretz, L.F.: Improving the COCOMO model using a neuro-fuzzy approach. Applied Soft Computing. 7, 29–40 (2007). https://doi.org/10.1016/j.asoc.2005.06.007.

[4] Difference between COCOMO 1 and COCOMO 2 Model - Tpoint Tech, https://www.tpointtech.com/difference-between-cocomo-1-and-cocomo-2-model, last accessed 2025/02/21.

[5]Kumar, B., Tiwari, U.K., Dobhal, D.C., Ram, M.: Function point analysis based effort estimation and prediction using Lagrange's interpolation in Agile software development. | EBSCOhost, https://openurl.ebsco.com/contentitem/gcd:164270741?sid=ebsco:plink:crawler&id=ebsco:gcd:164270741, last accessed 2025/02/21.

[6] Software Engineering: Function Point Analysis: Counting FPs, https://www.cs.rutgers.edu/courses/111/classes/fall_2011_tjang/texts/notes-java/principles_and_practices/fpa/fpa-counting.html, last accessed 2025/02/21.

[7] Hillman, M.F., Subriadi, A.P.: 40 Years Journey of Function Point Analysis: Against Real-time and Multimedia Applications. Procedia Computer Science. 161, 266–274 (2019). https://doi.org/10.1016/j.procs.2019.11.123.

[8] Functional Point (FP) Analysis - Software Engineering, https://www.geeksforgeeks.org/software-engineering-functional-point-fp-analysis/, last accessed 2025/02/21.

[9]Lavazza, L., Locoro, A., Liu, G., Meli, R.: Estimating Software Functional Size via Machine Learning. ACM Trans. Softw. Eng. Methodol. 32, 114:1-114:27 (2023). https://doi.org/10.1145/3582575.

[10] Zhao, Z., Jiang, H., Zhao, R., He, B.: Emergence of A Novel Domain Expert: A Generative AI-based Framework for Software Function Point Analysis. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. pp. 2245–2250. Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3691620.3695293.

[11] Karner, G.: Resource Estimation for Objectory Projects, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=17b5f04743cd13f6077fbdec227719e5d83dba10, (1993).

[12] Clemmons, R.: Project Estimation With Use Case Points. CrossTalk, The Journal of Defense Software Engineering. 19, (2006). https://www.researchgate.net/publication/200036324_Project_Estimation_With_Use_Case_Points

[13] Ochodek, M., Nawrocki, J., Kwarciak, K.: Simplifying effort estimation based on Use Case Points. Information and Software Technology. 53, 200–213 (2011). https://doi.org/10.1016/j.infsof.2010.10.005.

[14] Azzeh, M., Nassif, A.B., Martín, C.L.: Empirical analysis on productivity prediction and locality for use case points method. Software Qual J. 29, 309–336 (2021). https://doi.org/10.1007/s11219-021-09547-0.

[15] Ngafidin, K.N.M., Saintika, Y., Wijayanto, S.: Software Effort Estimation with Use Case Point Approach on Website Builder Project: A Systematic Performance Evaluation. In: 2023 Eighth International Conference on Informatics and Computing (ICIC). pp. 1–6 (2023). https://doi.org/10.1109/ICIC60109.2023.10381948.

[16] Barros-Justo, J.L., Benitti, F.B.V., Tiwari, S.: The impact of Use Cases in real-world software development projects: A systematic mapping study. Computer Standards & Interfaces. 66, 103362 (2019). https://doi.org/10.1016/j.csi.2019.103362.

[17] Estimating With Use Case Points, https://www.mountaingoatsoftware.com/articles/estimating-with-use-case-points#heading-13, last accessed 2025/02/21.

[18] Azzeh, M., Bou Nassif, A., Attili, I.B.: Predicting software effort from use case points: A systematic review. Science of Computer Programming. 204, 102596 (2021). https://doi.org/10.1016/j.scico.2020.102596.

[19] **[Tables]** **https://ieeexplore.ieee.org/document/9837735**

[20] S. B. Zahra and M. Nazir, "A Review of Comparison among Software Estimation Techniques". "https://www.researchgate.net/publication/270393939_A_Review_of_Comparison_among_Software_Estimation_Techniques" Bahria University Journal of Information & Communication Technology, vol. 5, pp. 39–45, 2012.

[21] B. W. Boehm, "Software Engineering Economics", NJ, USA: Prentice-Hall, 1981.

[22] P.M. Yawalkar, and L.R. Nerkar, "Software Cost Estimation using Algorithmic Model and Non-Algorithmic Model a Review". "https://research.ijcaonline.org/itcce/number2/ITCCE2010.pdf" Innovations and Trends in Computer and Communication Engineering (ITCCE-2014)

[23] N. Balaji, N. Shivakumar, and V. Vignaraj Ananth, "Software Cost Estimation using Function Point with Non-Algorithmic Approach, "https://www.researchgate.net/profile/Mohamed_Mourad_Lafifi/post/Need_latest_articles_of_Software_Cost_Estimation_of_FPA_using_Fuzzy_Logic/attachment/5a3610fcb53d2f0bba44e4ee/AS%3A572443716141056%401513492625215/download/Software+Cost+Estimation+using+Function+Point+with+Non+Algorithmic+Approach.pdf" Global Journal of Computer Science and Technology, vol. 13, no. 8, pp. 1-7, 2013.

[24] C. N. Parkinson, "Parkinson's Law: The Pursuit of Progress", London, UK 1958. "https://wrap.warwick.ac.uk/id/eprint/137491/7/WRAP-Incentive-schemes-resolving-Parkinsons-Law-project-management-Chen-2020.pdf"

[25] K.Singh, S.Chatterjee, and Marcello Mariani: "Applications of generative AI and future organizational performance": "https://www.sciencedirect.com/science/article/pii/S0166497224000713#abs0015"

[26] Hsu, YC, and Ching, YH: "Generative Artificial Intelligence in Education", Part One: the Dynamic Frontier. *TechTrends* **67**, 603–607. https://doi.org/10.1007/s11528-023-00863-9

[27] Boehm, B.W. (1984). Software Engineering Economics. IEEE Transactions on Software Engineering, [online] SE-10(1), pp.4–21. doi:https://doi.org/10.1109/tse.1984.5010193.

[28] Sophatsathit, P. (2014). Fine-Grained Work Element Standardization for Project Effort Estimation. Journal of Software Engineering and Applications, 07(08), pp.655–669. doi:https://doi.org/10.4236/jsea.2014.78060.

[29] Kitchenham, B. and Charters, S.M. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering | Request PDF. [online] ResearchGate. Available at: https://www.researchgate.net/publication/302924724_Guidelines_for_performing_Systematic_Literature_Rev

[30] Kocaguneli, E., Menzies, T. and Keung, J.W. (2012). On the Value of Ensemble Effort Estimation. IEEE Transactions on Software Engineering, 38(6), pp.1403–1416. Doi: https://doi.org/10.1109/tse.2011.111.

[31] Xia, C., Wang, Z., Shi, T. and He, X. (2013). An Improved Control Strategy of Triple Line-Voltage Cascaded Voltage Source Converter Based on Proportional–Resonant Controller. IEEE Transactions on Industrial Electronics, 60(7), pp.2894–2908. doi:https://doi.org/10.1109/tie.2012.2222854.

[32]https://ieeexplore.ieee.org/document/9837735