Artificial Estimators

A Study in Training Expert Systems for Estimating Software Development Task Efforts

Andrea Jans

Information & Computer Sciences, University of Hawai'i at Mānoa

Mentor: Dr. Daniel Port

HON 495: Thesis Proposal

Dr. Angela Sy

1st May 2024

Abstract

Since the dawn of programming, engineers have struggled to estimate the effort required to develop software. Precision, accuracy, and confidence level have remained elusive targets, often leading to over/underestimation of project timelines and costs. Bias is frequently the culprit, leading software engineers to miss the mark.

Halkjelsvik and Jorgensen (2012) shed light on the prevailing issues within this domain, revealing that deviations exceeding one-third of the initial estimate were frequent occurrences. Many estimation models have been developed over the years, such as COCOMO, ONSET, and FPA, with the primary aim of aiding project teams in making more accurate estimates for software development projects. Other models have provided frameworks and guidelines for estimating effort and cost at the software development task level, which will be the focus of this study.

This study delves into the possibility of training a machine learning (ML) system using a large language model (LLM) to generate accurate and reliable estimates within a desired confidence level, thereby addressing one of the long-standing challenges in software engineering. This study will source historical data from companies and organizations that keep records of their estimations and resulting actuals. Historical data consisting of functional specifications with estimated and actual time train the expert system and data not used in training tests the expert system. This study will compare the resulting estimates against the original data and analyze for precision, accuracy, and confidence level.

Table of Contents

Introduction

Chess has a storied history, starting in India in the 6th century CE, spreading to Persia, adopted by the conquering Muslims, and eventually arriving in Europe via Spain and Sicily around 1500 CE ("History of Chess," 2023). Though many play the game of chess, few master the game. In 1957, Alex Bernstein, an IBM engineer and mathematician, wrote the first known complete computer chess program (*Deep Blue*, n.d.). Over the following decades, computer chess programs began rivaling the amateur players, but the best players easily defeated them. In 1987, ChipTest, an advanced chess-playing machine created at Carnegie Mellon University, won the North American Computer Chess Championship (*Deep Blue*, n.d.). People started to wonder if a computer could beat the best-of-the-best chess players, Grandmasters. IBM decided to hire Carnegie Mellon University researchers and let them develop the hardware and software necessary to accomplish the task (*Deep Blue*, n.d.). In just a decade, Deep Blue became the first computer to defeat a Grandmaster, a chess expert, in tournament play (*Deep Blue*, n.d.).

In this age of advancements in Machine Learning, Large Language Models and Generative Pre-Trained Transformer systems, like Chat-GPT, can do marvelous things thanks to the developments pioneered by Deep Blue.

In computer science, one area often left to the experts is estimating the effort required to fix a programming defect or engineer a new feature. Based on rules of thumb and gut feelings, efforts are made to study the models used by experts and the confidence level that the estimate will be "probably approximately correct" (PAC) (Valiant, 1984).

Could an expert system based on a specially trained Large Language Model fronted by a Generative Pre-trained Transformer be trained using historical estimation data to estimate the efforts of software engineering projects with a satisfactory confidence level?

Literature Review

Challenges

Effort estimation in software development has been a persistent challenge since the 1960s, prompting extensive research and exploration into potential solutions. Halkjelsvik and Jorgensen (2012) shed light on the prevailing issues within this domain, revealing that deviations exceeding one-third of the initial estimate were frequent occurrences. Moreover, these deviations predominantly resulted in increases in the actual effort compared to the initially estimated effort.

Jørgensen (2004) surveyed software engineering estimation methods. Finding a standard set of 'best practices' for software engineering estimation by experts, Jørgensen (2004) laid out 12 guidelines:

1. Evaluate estimation accuracy but avoid high evaluation pressure
2. Avoid conflicting estimation goals
3. Ask the estimators to justify and criticize their estimates
4. Avoid irrelevant and unreliable estimation information
5. Use documented data from previous development tasks
6. Find estimation experts with relevant domain background and good estimation records
7. Estimate top-down and bottom-up, independently of each other
8. Use estimation checklists
9. Combine estimates from different experts and estimation strategies
10. Assess the uncertainty of the estimate
11. Provide feedback on estimation accuracy and development task relations
12. Provide estimation training opportunities.

The empirical research findings of Halkjelsvik and Jorgensen (2012) underline a pervasive trend in software project management: estimates often fall short of accurately predicting the required effort. These discrepancies can lead to various challenges, including budget overruns, missed deadlines, and compromised project quality.

Past Solutions

The COCOMO model proposed by Boehm (1981) emphasizes the intricacies of estimating software development effort and highlights the qualitative aspects of the process. The COCOMO Model provides a systematic way to calculate the cost and effort, helps identify the factors that have the most significant impact on the price and effort, and can determine the feasibility of a software project ("COCOMO Model - Software Engineering," 2018). However, the COCOMO Model assumes that the size of the software is the main factor, does not consider the specific characteristics of the software engineering team, and is not as precise as it could be ("COCOMO Model - Software Engineering," 2018).

According to Shepperd et al. (2018), one significant challenge that can impede the accuracy of expert estimates is biases in the estimator's judgment. Biases can skew the estimation process, leading to flawed decisions and outcomes.

One common bias observed in expert estimates is the anchor bias. Initial information, known as an anchor, may not necessarily be accurate or relevant provided to the estimator initiates the bias. Despite subsequent corrections or additional information, the estimator tends to cling to the original anchor, even if incorrect. This results in a distortion of judgment and estimation (Shepperd et al., 2018).

Dagnino (2013) introduced the ONSET (ONe Software Estimation Toolkit), a method for estimation in situations where historical data is lacking. ONSET amalgamates various concepts, principles, and techniques from the software estimation domain, usually treated as separate entities in literature. This toolkit becomes particularly valuable for organizations facing the challenge of unreliable historical data. The paper illustrates the application of these principles

and methods within ONSET, demonstrating how estimates can be derived effectively without relying on intricate or costly tools.

Machine Learning

In a current survey of software engineering effort estimation techniques, Saeed et al. (2018) reiterated that effort estimation in software development is a crucial aspect that significantly impacts project planning, resource allocation, and overall project success. One key takeaway emphasized by the research is that different effort estimation models have advantages and limitations. These models are not one-size-fits-all solutions but tools tailored to specific project requirements and conditions. The choice of estimation model depends on factors such as project size, complexity, team composition, and available historical data. The survey did find that supervised learning algorithms are the most popular for effort estimation.

Lavazza et al. (2023) applied machine learning techniques to the early estimation of software functional size. Their objective was to investigate whether machine learning methods could produce more precise estimates of Functional Point Analysis (FPA) measures compared to traditional approaches like High-level FPA. They conducted an empirical study using a substantial dataset of functional size predictors to train and assess three widely recognized and robust machine learning algorithms: Random Forests, Support Vector Regression, and Neural Networks. The study involved a systematic experimental process encompassing dataset filtering, splitting, parameter optimization, and model training and validation cycles.

Following this methodology, Lavazza et al. (2023) evaluated the estimation accuracy of the machine learning models and compared them against fixed-weight models (e.g., High-level FPA) and linear regression models. Additionally, the performance of these models employed a separate dataset as a test set to validate. The findings suggest that Support Vector Regression

produces relatively accurate estimation models. However, the level of accuracy achieved is similar to that of High-level FPA or models constructed using ordinary least squares regression. Lavazza et al. (2023) observed that models capable of generating accurate estimations only sometimes require differentiation among transaction and data types.

The groundwork has been laid for the next generation of machine learning algorithms, including large language models, to perform software engineering effort estimation. The large language model finds similar tasks across the training data using multiple methodologies and historical data. This historical data based on functional specifications, estimates, and actual effort should deliver estimations that will be accurate and have a high confidence.

Research Methods

Design

This mixed-method research project will qualitatively train a large language model with functional specifications and estimation vs. actual software engineering task efforts. Quantitatively, past projects not in the training data will test the system's output and compare it against the estimation vs. actual software engineering task efforts.

As this research project is not about building a machine learning system using a large language model, a free/open source software (FOSS) package will be identified and trained using the data acquired from past projects, with the appropriate data transformed into a proper format. Prior research on machine learning models has yet to use large language models. Therefore, this research is believed to be among the first to do so. As an exploratory research project, the goal is to increase the knowledge base regarding using machine learning and large language models in software engineering effort estimation.

The historical estimates and actual efforts recorded by the original project teams compare to the estimations returned by the large language model. Historical project data will be run multiple times to generate enough data to make statistical analysis reasonable. The estimates from the LLM will also be compared against prior system runs. The research aims to determine if the estimates produced by the LLM are accurate and fall into a confidence level.

Data Collection

The primary data will be collected from past software projects that have collected and saved the functional specifications, estimates, and actual effort expended. The software will likely need development to transform the disparate data formats into a standard format for digestion by the training or estimation algorithms.

The research project will ask the software development community to volunteer their historical data. These teams will be interested in the findings and whether a tool that is as accurate as a human expert can be created. This primary data has been collected in the field and will be analyzed in the laboratory. It is the only data that can be used to train and test the LLM, and as many sources as possible will be sought.

The strengths of the data sources are that they are actual data from the field collected by organizations interested in improving the software engineering effort estimates used to model the cost and schedule estimates of future software projects. However, a set format for storing and organizing the historical data is needed.

Measurement and Instrumentation

The primary measurement will be the estimation of low-level task-based software engineering efforts produced by the LLM. The low-level estimates are rolled up into higher-level

task estimates until the forecast for the entire project can be derived. This data is then compared against the actual historical results to determine accuracy.

The research project will use cloud computing to virtualize computing power and not invest in physical hardware. The LLM, developed utility software, training data, and test sets will be stored as source files in a GitHub repository owned by the research project. Putting the project in the cloud will allow focusing efforts on the functionality needed instead of being side-tracked by problems with physical hardware.

Data Organization and Management

Training and test run data will be stored in the GitHub repository using a naming convention to easily differentiate the test run and build of the LLM that produced the file. Research project computers or cloud instances store the data consumed and produced.

Timeline

Task	Due	Status
GitHub Organization Created	May 2024	Not started
GitHub Repositories Created	May 2024	Not started
LLM Engine Cloned	June 2024	Not started
Cloud Computing Environment Chosen	June 2024	Not started
LLM Engine Installed	June 2024	Not started
First Historical Training Data Received	July 2024	Not started
First LLM Engine Training Completed	August 2024	Not started
Preliminary Tests Completed	September 2024	Not started
Final Historical Training Data Received	October 2024	Not started
Final LLM Engine Training Completed	November 2024	Not started

Task	Due	Status
Final Tests Completed	December 2024	Not started
Results Analyzed	January 2025	Not started
Report Written	February 2025	Not started
Report Peer Reviewed	March 2025	Not started
Results Presented	April 2025	Not started

References

Boehm, B. W. (1981). *Software engineering economics*. Prentice-Hall.

COCOMO Model—Software Engineering. (2018, April 13). *GeeksforGeeks*. https://www.geeksforgeeks.org/software-engineering-cocomo-model/

Dagnino, A. (2013). Estimating software-intensive projects in the absence of historical data. *Proceedings of the 2013 International Conference on Software Engineering*, pp. 941–950.

Deep Blue | IBM. (n.d.). IBM Heritage. Retrieved February 18, 2024, from https://www.ibm.com/history/deep-blue

Halkjelsvik, T., & Jørgensen, M. (2012). From origami to software development: A review of studies on judgment-based predictions of performance time. *Psychological Bulletin*, *138*(2), 238–271. https://doi.org/10.1037/a0025996

History of chess. (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=History_of_chess&oldid=1188427997

Jørgensen, M. (2004). A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, *70*(1), 37–60. https://doi.org/10.1016/S0164-1212(02)00156-5

Lavazza, L., Locoro, A., Liu, G., & Meli, R. (2023). Estimating Software Functional Size via Machine Learning. *ACM Transactions on Software Engineering and Methodology*, *32*(5), 114:1-114:27. https://doi.org/10.1145/3582575

Saeed, A., Butt, W. H., Kazmi, F., & Arif, M. (2018). Survey of Software Development Effort Estimation Techniques. *Proceedings of the 2018 7th International Conference on Software and Computer Applications*, 82–86. https://doi.org/10.1145/3185089.3185140

Shepperd, M., Mair, C., & Jørgensen, M. (2018). An experimental evaluation of a de-biasing intervention for professional software developers. *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 1510–1517. https://doi.org/10.1145/3167132.3167293

Valiant, L. G. (1984). A theory of the learnable. *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, 436–445. https://doi.org/10.1145/800057.808710