Software Project Effort Estimation Using Formal Method and Model Checker

Abdulaziz Alhumam
Dept. of Computer Science-College of Computer Science and Information Technology, King Faisal University, Saudi Arabia

***Abstract*—Software project effort estimation is a critical component of software development, as it determines the time and financial resources required to complete a project. Existing estimation techniques—ranging from empirical models and algorithmic methods to heuristic and expert-based approaches—struggle with inconsistent accuracy due to the inherent complexity, subjectivity, and contextual variability across software projects. Although earlier formal methods aimed to reduce confusion by being very precise, they usually don't allow for automated logical analysis or check if the assumptions used for estimation are consistent with one another. To address these limitations, this study introduces a novel formal modeling framework that integrates Z-Specification with the Z3 SMT solver to both formalize and computationally verify effort estimation models. The use of Z notation guarantees the meaning is precise and unambiguous. Furthermore, SMT (Satisfiability Modulo Theories) reasoning adds powerful new abilities that older methods lacked. These new capabilities include automatically finding constraint violations, confirming how parameters depend on one another, and determining feasible estimation ranges under clearly defined conditions. This integration not only reduces ambiguity but also provides a verifiable, machine-checkable basis for evaluating, refining, and comparing diverse effort estimation methods, thereby offering a more robust foundation than traditional or solely formalized models.**

***Keywords*—Software effort estimation; cost estimation; formal methods; SMT Solver; automated verification; Z-specifications**

I. Introduction

One of the crucial concerns for a Software house or company is to deliver the software product to its client or end-users within the agreed-upon timeframe. The software delivered must meet the requirements and satisfy all the quality parameters while remaining within financial budget constraints. Therefore, accurate assessments of the Software Project Effort estimation serve as key factors that may also guide the achievement of important goals. In other words, it is incredibly important to understand and manage the cost through accurate estimation for proper management, improved quality, and enhanced comprehension of the software project. Estimating software costs is an ongoing process that begins at the planning stage and continues throughout the entire duration of a project. Ongoing cost estimation is necessary to ensure that expenditures align with the budget. In the last few decades, many models have been proposed and developed to calculate the software costs. These models are based on various parameters, one of them is based on the use of information from previous projects to assess the current project cost. The other method is to develop essential formulas by analyzing the specific database available. Many of the proposed cost models depend on size measurements, like lines of code (LoC) and Function Points. Precisely determining the size rightly impacts the exactness of cost estimation. However, none of the previously mentioned methods yield an accurate estimation [1].

The widely used Effort estimation models, such as COCOMO and Function Point Analysis, suffer from subjectivity, data-dependency, and a lack of formal verification [2]. Various Formal methods (FM) are proposed, which provide a robust mathematical foundation model and verify system components, and one of the FM is Z [3]. The current study introduces a novel formal framework for software effort estimation that integrates the semantic clarity of Z notation with the automated reasoning capabilities of an SMT solver.

This Z-based formalization ensures semantic clarity, while SMT-backed reasoning introduces capabilities not available in earlier approaches—such as automated detection of constraint violations, verification of parameter dependencies, and exploration of feasible estimation ranges under formally defined conditions. We hypothesize that this rigorous, formally verifiable approach will yield a measurable improvement in estimation accuracy and consistency compared to existing empirical models. We formalized the estimation process, defined its components rigorously, and verified relationships using Z-Schema constructs, then applied the schemas to the Z3 SMT Solver [4] to estimate the software project effort. The results of this validation, presented in Section VII, demonstrate the framework's ability to provide more accurate and transparent estimation ranges.

II. Related Work

In today's software industry, the most important thing for any company is to deliver a project successfully and on time. From a management point of view, predicting efforts is a challenging task. According to the report, 65 to 80 % of projects have delays in their delivery dates. The higher the effort, the higher the cost. Therefore, precise prediction is essential. There are many models for estimating software development effort and costs. COCOMO (Constructive Cost Model) [5] is the most used model. Machine learning methods are more apt for software effort prediction since they can modify more easily. Machine learning can manage the variations better that are used to measure the effort needed to develop software, which depends on many factors — including the size of the problem, the experience of the users or developers, and other comparable inputs.

The effort estimation typically involves input of the project size based on the lines of code (LoC) or Functional points,

complexity levels, risk factors, and historical productivity data. These inputs are often combined using empirical models but are rarely subjected to rigorous formal analysis.

Formal Methods approach is used for writing formal description, analysis, and for the purpose of producing refinements [6]. A formal specification description is abstract, precise and, in a way, comprehensive. Precision drives ambiguities to be questioned and eliminated, the abstraction makes it possible for a human reader to grasp the larger picture, and the completeness ensures that every facet of behavior is described. Second, a thorough analysis can be conducted because of the description's formality. One can ascertain practical characteristics like deadlock-freedom or consistency by examining a single description [7]. Important characteristics like meeting high-level requirements or the accuracy of a suggested design can be ascertained by writing several descriptions from various angles [7]. Formal methods have been studied in the computing community since at least the 1960s, with seminal work by Floyd [9], Hoare [9], and Dijkstra [10] defining techniques for proving programs correct.

One of the model-based notations, which is a strongly typed mathematical specification language, is Z notation, which is not an executable notation that can be interpreted or compiled into running programs. Z-Specification Language provides: Schema-based notation, State and operation modeling, and support for invariants and preconditions. There are special tools for testing Z texts, which are like a mathematical description for software systems, and these tools look for two types of errors. This includes syntax errors like grammatical mistakes, checking if the Z test is written using the correct rules and symbols to be followed for Z notation, and type errors are being used consistently and correctly. These are similar to how a compiler checks the regular computer code before it can run. The characteristic that sets Z apart from other formal notations is the schema, which is used in Z to characterize a system's dynamic and static elements. The Z standard makes it possible to create a clear, verifiable, and traceable model. Z has an ISO standard and is further developed [11].

III. METHODOLOGY

This research study proposes and explores a framework for robust software effort estimation, leveraging an integrated model encompassing key variables, verification schemas, and equation schemas. The framework explicitly defines interdependencies between critical factors such as Effort, Complexity, Productivity, Risk, and Size. It introduces dedicated "Verification Schemas" for validating initial estimations of Effort and Complexity, and "Equation Schemas" that detail the computational relationships for deriving Productivity, Effort, and Complexity. By emphasizing both calculation and explicit validation of estimates, this work aims to enhance the accuracy, reliability, and transparency of software project predictions, addressing the usual challenges in traditional estimation methodologies. Fig. 1 represents a framework for a robust software effort estimation model related to cost, effort, and complexity estimation, for calculating and verifying project metrics, based on initial factor size and risk.

Fig. 2 represents a conceptual model or dependency, which is structured into three main areas: Input Factors, which are size and risk, which directly influences effort and core metrics. The effort calculated from size and risk is a key factor in the equation schemas, the other factors are productivity and complexity. Productivity is influenced by effort which leads to complexity, and which is calculated using formulas derived from effort. Complexity is the final calculated metric, which is influenced by productivity and is the focus of the verification schemas.

A. Variables (Input/State)

This defines the core quantitative and qualitative measures being tracked. These variables serve as inputs to the schemes for Effort: The amount of work needed (e.g., person-hours or person-months), Complexity: A measure of the difficulty of the system, Productivity: The rate at which output is generated (e.g., code per unit of effort), Risk: A measure of potential problems or uncertainties and Size: The magnitude of the system (e.g., lines of code, function points).

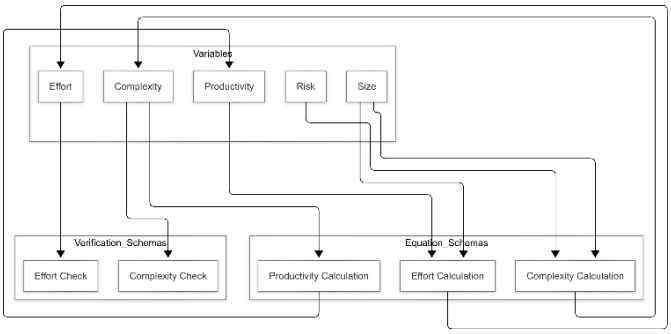

Fig. 1. Framework for robust software effort estimation.

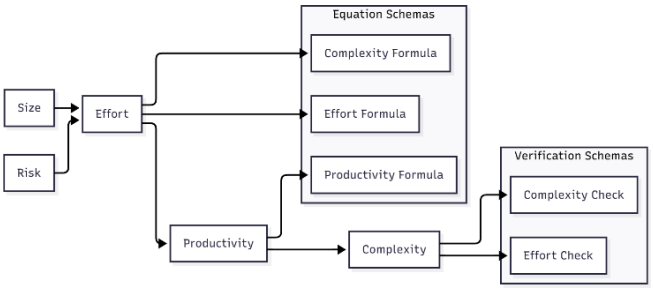

Fig. 2. Conceptual model

B. Verification Schemas

Represents validation or constraint checks on the system's properties. These schemas take variables as input and output as a Boolean value or a report on whether a condition is met. EfforCheck: Depends on effort variable. It checks if the effort falls within an acceptable range or budget and ComplexityCheck: Depend on the complexity variable. Likely checks if the system's complexity is manageable or below a defined threshold. The Verification Schemas checks to validate the calculated metrics: Effort Check: Effort <100 which sets an upper limit or a constraint on the calculated Effort. Complexity1 < Complexity2 suggests a comparative check.

C. Equation Schemas

Represents computations or functions that determine value of one variable based on others. Productivity calculation depends on effort and complexity variables. Effort calculation depends on the size and productivity variables. Complexity

calculation depends on the risk and size variables. Equation Schemas details the mathematical relationships used to calculate the core metrics: `Effort=Size /Productivity`. Effort is inversely proportional to Productivity for a given Size. `Complexity=Size*(1-Risk/10)`. Complexity increases with size but decreases as risk increases. `Productivity=100/Complexity` which defines productivity as inversely proportional to Complexity.

IV. Formal Description

Formal methods are mathematically based techniques and tools for the specification, design, and verification of software and hardware systems. They provide a rigorous framework for describing system properties and behaviors, allowing for precise reasoning and analysis. Fig. 3 given below illustrates a simplified model for calculating the effort required for a project, p, incorporating factors typically considered in project management, and can be interpreted through the lens of formal methods, particularly in the context of system modeling and analysis.

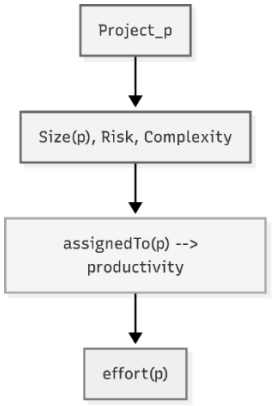

Fig. 3. Sequence of transformations.

The schemas described, which relate Size, Risk, Effort, Productivity, and Complexity through explicit mathematical formulas and constraints, can be formally specified using the Z Notation. The specification below defines the state space of the system and the operations that modify or check the state, based on the provided schemas.

A. Z Specification for Project Metrics Model

1) Basic types: We start by defining the basic sets for the variables in the model. Since all variables represent numerical quantities (size, effort, percentage/ratio), ℕ (natural numbers) and ℝ (real numbers).

```
[ProjectID]
size, Effort, Productivity, Complexity: ℝ
Risk: ℝ
```

2) State schema: The state schema ProjectMetrics defines the variables that constitute the state of the project estimation model.

```
ProjectMetrics
size, Effort, Productivity, Complexity: ℝ
Risk: ℝ
size ≥ 0
0 ≤ risk ≤ 10
effort ≥ 0
productivity ≥ 0
complexity ≥ 0
```

3) Initialization schema: The Init schema sets the initial state of the system, typically with the input variables defined and the calculated metrics initialized to zero or undefined.

```
Init
Δ ProjectMetrics
size’ ∈ ℝ
risk’ ∈ ℝ
effort’ = 0
productivity’= 0
complexity’ = 0
size’ ≥ 0
0 ≤ risk’ ≤ 10
effort’ = 0
productivity’ = 0
complexity’ = 0
```

4) Operation schema: CalculateMetrics: The CalculateMetrics schema represents the core function of the model, applying the "Equation Schemas" to calculate the output metrics based on the current inputs.

```
CalculateMEtrics
Δ ProjectMetrics
risk < 10
complexity’=size x (1-risk/10)
productivty’=100/complexity’
effort’= size/productivity’
size’ = size
risk’ = risk
```

5) Verification Schemas (Constraint Checks): The "Verification Schemas" [8] define checks that must be satisfied. These are modeled as query operations that report success or failure.

a) Effort verification: The CheckEffort checks if the calculated effort is below 100.

```
CheckEffort
Ξ ProjectMetrics
report!: Boolean
```

```
(effort < 100) ⇒ (report!= true)
(effort ≥ 100) ⇒ (report!= false)
```

b) Complexity verification: The CompareComplexity schema checks the constraint $Complexity_1 < Complexity_2$. This requires comparing the current state ($Complexity_1$) to an external value ($Complexity_2$) which is taken as input.

```
CompareComplexity
Ξ ProjectMetrics
Comparison_complexity? : ℝ
report!: Boolean
(complexity < comparision-complexity?) ⇒
(report!= true)
(complexity ≥ comparision-complexity?) ⇒
(report!= false)
```

V. Tools for Automation of Formal Models

Formal methods can help turn a conceptual diagram into a formal model that can be processed by automated tools. There are various tools and approaches that support the automation of Formal models, and these tools help in three ways in specifying the model, which allows users to write the system's properties and behaviors clearly, following rigorous and unambiguous language. These formal methods help in the analysis/verification of the model to check whether it is correct, complete, and satisfies the desired condition, and generate results/code directly from the specification. For effort estimation models, this could mean generating an executable version of the estimation function.

A. Model Checkers / Theorem Provers

These are core formal tools which are crucial for transforming a formal specification into something that can be analyzed automatically.

1) Z3 (SMT Solver) [4]: A popular and powerful tool named Satisfiability Modulo Theories (SMT) solver developed by Microsoft Research. Using the tool, one can encode relationships as logical formulas used to check for consistency and to find specific values for given certain inputs (Size, Risk, Complexity, Productivity) to calculate the effort.

```
(e.g.,Effort(p)=f(Size(p),
Risk(p),Complexity(p),
Productivity(assignedTo(p),p)))
```

2) Alloy (Relational Model Finder): Alloy is a language and tool for modeling structural properties of software systems [12]. It's excellent for modeling relationships and constraints. Alloy could then analyze these relationships for consistency or discover instances that satisfy certain conditions. While it doesn't directly handle numerical calculations as easily as Z3, it is strong for the qualitative aspects and relationships.

3) PVS (Prototype Verification System) / Isabelle/HOL / Coq (Interactive Theorem Provers) [13-15]: These are more heavyweight tools used for rigorous mathematical proofs.

B. Domain-Specific Modeling Languages (DSMLs) / Low-Code/No-Code Platforms [16].

Though these tools are not strictly used in "formal methods" in the highly rigorous sense of theorem provers, these can bridge the gap between conceptual diagrams and executable models, often with some level of formality.

1) MATLAB/Simulink or LabVIEW [17, 18]: These tools are excellent where one can define functions and simulate scenarios involving mathematical equations and simulations. One can implement Effort(p) function in these environments and then run simulations with varying inputs for Size, Risk, Complexity, and Productivity to see the impact on Effort. They aren't "formal verification" tools, but they allow quantitative modeling and analysis.

2) Business Process Model and Notation (BPMN) [19] Using these tools to model the process of effort estimation. While BPMN isn't a formal method for mathematical models few of the tools allow one to embed executable logic at each step, making the process formally specified and potentially automated. Tools. While primarily for business processes, the "flow" aspect of the diagram could be modeled.

3) General-Purpose Programming Languages: With Formal Libraries Languages, like Python with SymPy, SciPy, or Z3 bindings, one can implement a formal model directly in code.

4) Data Modeling Tools (for structured data and relationships) Tools like erwin Data Modeler [20], PowerDesigner [21], or Sparx Enterprise Architect can create conceptual, logical, and physical data models. While they don't directly "calculate effort", they can formally define the structure of your inputs and outputs. The entities for attributes size, Risk, Complexity, and Team can be defined, which impact the productivity. Using these tools, one can ensure data consistency and define relationships, which is one of the foundational steps for formal models.

VI. Formal Verification using Z3 SMT Solver

To apply the Z3 SMT solver for checking equations related to software project factors like Size, Risk, Complexity, and Productivity, a formal model was created, where the variables were defined and the rules that connect them, and how these variable equations and verification rules were created using Z3 Python API. Define each of the components: *P, Size, Risk, Complexity, assignedTo, Productivity, and Effort, w*hich are proposed specific mathematical forms for these functions (e.g., Size as function points, Risk as a probability of failure, Complexity using cyclomatic complexity). Fig. 3 represents a simple function or a sequence of transformations.

To support formal reasoning, the relationships among the project parameters were encoded as a constraint system amenable to SMT-based verification in Z3. The independent variables Size $\in \mathbb{N}^+$ and Risk $\in \mathbb{Q} \cap [0, 1]$ were instantiated for each case as fixed inputs, while Complexity, Productivity, and Effort were declared as dependent variables with explicit constraint schemas. Complexity was modeled as a monotonically increasing function of both Size and Risk, i.e., $\forall s,r \cdot (s\uparrow \vee r\uparrow) \Rightarrow$ Complexity$\uparrow$, capturing the empirical effect

that large or high-risk projects induce structural and managerial overhead. Productivity was constrained as a decreasing function of Complexity and Risk, i.e., ∀ c,r · (c↑ ∨ r↑) ⇒ Productivity↓, to reflect degradation of throughput under cognitive load, rework, and verification pressure. Effort was then encoded as an outcome variable derived via an effort-equation schema Effort = f (Size, Productivity), under the assumption that lower productivity under fixed size produces strictly higher effort. The "Project Cases" table enumerates three instantiated assignments (Size,Risk) with their Z3-verified consequences (Complexity, Productivity, Effort) satisfying a11 declared axioms and constraints, thereby exhibiting the expected monotonic progression Size,Risk ⇒

Complexity ⇒ Productivity ⇒ Effort.

A. Project_p (Initial State/Input)

This can be viewed as the initial "system" or "object" under consideration. In formal methods, we would define the set of all projects, perhaps as *P*, where $p \in P$.

Formally, `Project_p` could be represented as an initial state S_0 in a state-transition system, or simply as an input parameter to a function.

B. Size(p), Risk, Complexity (Parameters/Attributes)

These are attributes or properties associated with *Project_p*. In formal methods, these would be formally defined metrics or functions that map a project to a numerical value or a categorical description.

- *Size(p):P→R*$^+$ (e.g., lines of code, function points)
- *Risk(p):P→ [0,1]* (e.g., a probability or a risk score)
- *Complexity(p):P→R*$^+$ (e.g., cyclomatic complexity, number of interdependencies)

These could be seen as a tuple of characteristics: *(Size(p), Risk(p), Complexity(p)).* The transition from "*Project_p*" to "*Size(p), Risk, Complexity*" implies an evaluation function or a feature extraction process that characterizes the project.

C. assignedTo(p) → productivity (Function/Relation and Impact Factor)

This is the most critical part from a formal methods perspective, as it introduces a relationship and a derived property.

1) assignedTo(p): This represents the resources (e.g., team members, skill sets) allocated to project p. Formally, this could be a function A:P→R where R is the set of all resource allocations.

2) Productivity: This is an attribute of the assigned resources, or a property derived from the interaction of the assigned resources with the project's characteristics. Productivity could be defined as a function Prod: R×P→R$^+$. The arrow "→" signifies that assignedTo(p) directly influences productivity. The assignedTo(p) → productivity highlights that this step involves a mapping or a transformation where the characteristics of the assigned team (e.g., their skill levels, experience, team size) influence how efficiently the project attributes (size, risk, complexity) are managed. This could be a complex function incorporating human factors.

D. Effort(p) (Output/Result)

This is the final calculated output of the model.

Effort(p) would be a function that considers the project's characteristics (Size(p), Risk, Complexity) and the derived productivity.

Formally, *Effort(p)=f(Size(p), Risk(p), Complexity(p),*

E. Define Variable

```
from z3 import *

# Define variables as real numbers (can
also be Ints if discrete)
Size = Real('Size')
# Size of the software project (e.g., LOC,
Function Points)
Risk = Real('Risk')
# Risk factor (e.g., scale 0–1 or 0–10)
Complexity = Real('Complexity')
# Complexity of the system (scale 0–10)
Productivity=Real('Productivity')
# Productivity (e.g., LOC/person-month)
```

F. Equation Schemas

These are mathematical models or assumptions defining how variables relate.

1) Effort Estimation Equation

We define effort (E) based on the size and productivity.

```
Effort = Real('Effort')
equation_schema_1 = Effort == Size
                    / Productivity
```

2) Risk-adjusted Complexity Equation

Assume complexity increases with risk:

```
equation_schema_2 = Complexity == Size *
(1 + Risk / 10)
```

3) Productivity Impact by Complexity

Assume productivity decreases as complexity increases:

```
equation_schema_3 = Implies (Complexity > 0,
Productivity == 100 / Complexity)
```

G. Verification Schemas

These are assertions or properties we want to verify using Z3. Verify effort is less than a threshold (say 100 person-months)

```
verification_schema_1 = Effort < 100
```

Verify that increasing risk leads to increased complexity.

```
# Suppose we model two risk levels
Risk1 = Real('Risk1')
Risk2 = Real('Risk2')
Complexity1 = Real('Complexity1')
Complexity2 = Real('Complexity2')
schema_risk1 = Complexity1 == Size * (1 +
```

```
Risk1 / 10)
schema_risk2 = Complexity2 == Size * (1 +
Risk2 / 10)
schema_assumption = And (Risk1 < Risk2)
verification_schema_2 = Implies
(schema_assumption, Complexity1 <
Complexity2)
```

H. Putting it all into Z3 Solver

Constraint Setup for this Data: Below is a minimal Z3 SMT model that encodes the relationships used. This Z3 model computes Complexity, Productivity, and Effort.

```
s = Solver ()
# Add the equation schemas
s.add(equation_schema_1)
s.add(equation_schema_2)
s.add(equation_schema_3)
# Add the verification schema
s.add(verification_schema_1)
# Add additional bounds or assumptions
s.add(Size > 0, Productivity > 0, Risk >= 0,
Risk <= 10)
# Check
satisfiability if
s.check() == sat:
    print ("Verification successful!")
    print(s.model())
else:
    print ("Verification failed or
unsatisfiable.")
```

Fig. 4, Fig. 5, Fig. 6, Fig. 7, and Fig. 8 demonstrate the Z3 (SMT Solver) output when the property of Effort is changed.

```
(.venv) C:\z3_demo>python model.py
SAT result: sat
--- Example satisfying assignment ---
Size = 51
Risk = 490/51
Complexity = 100
Productivity = 1
Effort = 51

(.venv) C:\z3_demo>
```

Fig. 4. Output when the property of Effort <100.

```
(.venv) C:\z3_demo>python model.py
SAT result: sat
--- Example satisfying assignment ---
Size = 17
Risk = 490/51
Complexity = 100/3
Productivity = 3
Effort = 17/3

(.venv) C:\z3_demo>
```

Fig. 5. Output when the property of Effort <10

When the property of Effort is changed to less than 50 "Effort < 50":

```
(.venv) C:\z3_demo>python model.py
SAT result: sat
--- Example satisfying assignment ---
Size = 26
Risk = 120/13
Complexity = 50
Productivity = 2
Effort = 13
```

Fig. 6. Output when the property of Effort <50.

```
(.venv) C:\z3_demo>python model.py
Small & low-risk: sat
SAT result: sat
--- Example satisfying assignment ---
Size = 0
Risk = 0
Complexity = 0
Productivity = 0
Effort = 6

(.venv) C:\z3_demo>
```

Fig. 7. Output when small project <= 15 KLOC with low risk.

```
(.venv) C:\z3_demo>python model.py
Small & high-risk: sat
SAT result: sat
--- Example satisfying assignment ---
Size = 0
Risk = 8
Complexity = 0
Productivity = 0
Effort = 6

(.venv) C:\z3_demo>
```

Fig. 8. Small project <= 15 KLOC with high risk.

VII. RESULTS

Five different project scenarios with varying size and different risk test cases were performed with the exact values to measure the Complexity, Productivity, and Effort for the various computed values that were used for the test cases as shown in Table I.

TABLE I. VARIOUS COMPUTED VALUES

Case	Size	Risk	Complexity	Productivity	Effort
A	20	1	22.0	4.54	4.4
B	40	3	52.0	1.92	20.8
C	60	5	90.0	1.11	54.0
D	80	7	136.0	0.73	108.8
E	100	9	190.0	0.52	190.0

The graphs illustrated in Fig. 9, Fig. 10 and Fig. 11 present a realistic engineering insight, where risk and size amplify complexity, and reduced productivity rapidly inflates total effort. Fig. 9 and Fig. 10 illustrate a critical, non-linear relationship in software projects: as Size and Risk increase complexity across cases increases non-linearly as project size grows and as risk becomes higher as in Case A to Case E. Complexity value more than doubles from Case A to Case B, while the Size only doubles and Risk increases by a small linear step (1 to 3). This means larger and high-risk projects quickly become harder to manage. Complexity amplifies, leading to a sharp drop in Productivity, which ultimately results in a significantly inflated total Effort. Fig. 10 shows a rapid, steep decline in Productivity as the project cases advance. The largest drop occurs early, between Case A and Case B, representing a 57% reduction in productivity.

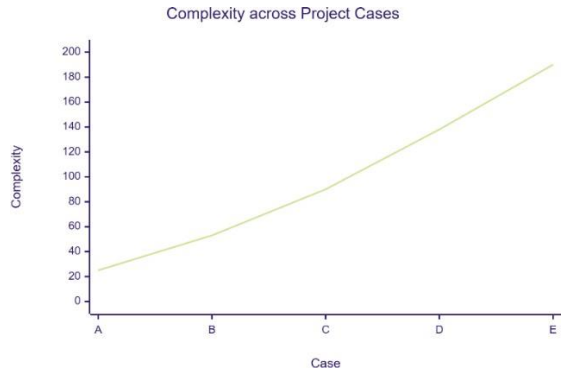

Fig. 9. Complexity across Cases.

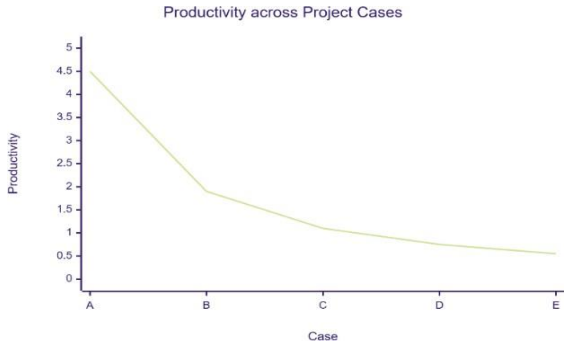

Fig. 10. Productivity across Cases.

Fig. 10 demonstrates the productivity across cases. As complexity increases, productivity drops, which reflects that the real projects' high uncertainty and size reduce the effective delivery rate, and the productivity versus Cases strictly decreases as complexity rises. While Size increases linearly (20, 40, 60, 80, 100), the calculated Effort increases at a highly accelerated rate: Case A to B: Effort increases 4.7 times (4.4 to 20.8). Case D to E: Effort increases by 1.7 times (108.8 to 190.0). The rapid drop in Productivity (due to amplified Complexity) is the primary driver of total project Effort. This phenomenon is a realistic depiction of large, complex, and high-risk projects where the team spends exponentially more time dealing with integration issues, debugging, and coordination (i.e., low productivity) than on core development, resulting in project costs that balloon far beyond the initial linear expectation based on size alone.

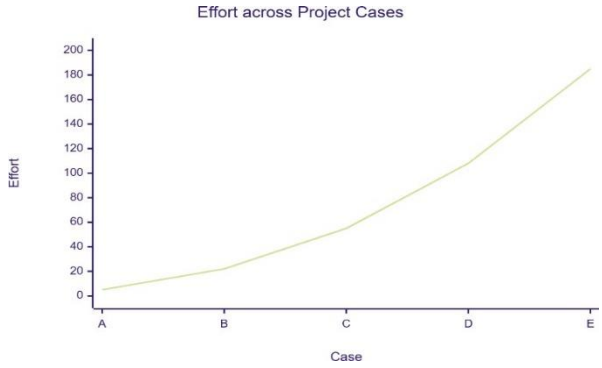

Fig. 11. Effort across Cases.

Fig. 11 demonstrates Effort across cases. Effort increases dramatically when productivity drops and for moderate rises in risk or size have a disproportionate impact on total effort. Effort versus Cases grow sharply as productivity drops.

VIII. Conclusion and Future Work

Software effort estimation has progressed from early heuristic and cost–based models such as COCOMO to more data-driven and logic-based frameworks. This work contributes to that evolution by demonstrating how relationships among Size, Risk, Complexity, Productivity, and Effort can be formalized using Z-notation and verified through an SMT solver such as Z3. Instead of relying on heuristic intuition alone, the encoded schemas enforced logically consistent relationships, ensuring that any computed effort values were entailed by the declared constraints. The illustrative "Project Cases" and the generated graphs confirmed that the expected behavior increases in size and risk propagated to higher complexity, reduced productivity, and elevated total effort. This coherence between historical theory, formal reasoning, and empirical visualization supports the feasibility of combining classical estimation knowledge with modern verification machinery to produce estimations that are interpretable, auditable, and computationally checkable.

In the future, we plan to improve the model by adding more real-world factors such as changing requirements, developer skill levels, number of defects, and outside system dependencies. We also plan to move from fixed (static) values to dynamic ones, so that factors like risk and productivity can change over time as the project progresses instead of staying constant. Hybrid or probabilistic SMT approaches may further enable uncertainty-aware verification and confidence-bounded outputs. Validation with industrial datasets will help assess whether real-world trajectories satisfy or refine the formal schemas. Ultimately, integrating the verified specification into automated tools or dashboards may provide project managers with rigorously justified, real-time decision support, aligning formal verification research with practical software engineering governance.

References

[1] Alturki, F., Alshahrani, A., Alhazmi, A., and Albogami, M. "Comprehensive Analysis of Software Effort Estimation Techniques: Evolving Trends, Key Challenges, and Prospective Directions". International Journal of Computer Applications, vol 186, no. 68, pp. 41–48, 2025.

[2] Lavazza, L.; Locoro, A.; Meli, R. “Software Development and Maintenance Effort Estimation Using Function Points and Simpler Functional Measures”. Software 3, no. 4, pp. 442-472, 2024.

[3] Ghassemi, B., and Ghassemi, R. “Software Cost Estimation: A Comparative Analysis Of Traditional And Machine Learning Approaches”. International Journal of Environmental Sciences, vol 11(7s), pp. 710–720, 2025.

[4] de Moura, L. and Bjørner, N. “Z3: An efficient smt solver’, *Lecture Notes in Computer Science”,* pp. 337–340, 2008, doi:10.1007/978-3-540-78800-3_24.

[5] Pahariya, J. S., V. Ravi, et al. "Software cost estimation using computational intelligence techniques". Nature & Biologically Inspired Computing, NaBIC 2009. World Congress on,2009.

[6] Clarke, E. M., Henzinger, T. A., & Veith, H. (Eds.). *Handbook of Model Checking*. Springer, 2020.

[7] P. W. H. Woodcock and J. Davies. “Using Z: Specification, Refinement, and Proof”. Prentice Hall, 1996.

[8] Song Gao, Bohua Zhan, Depeng Liu, Xuechao Sun, Yanan Zhi, David N. Jansen, and Lijun Zhang. Formal verification of consensus in the Taurus distributed database. In *Proceedings of the 24th International Symposium on Formal Methods (FM'21)* (LNCS, Vol. 13047), Marieke Huisman, Corina S. Pasareanu, and Naijun Zhan (Eds.). Springer, Germany, 741–751., 2021.

[9] C. A. R. (Tony) Hoare. 1969. An axiomatic basis for computer programming. Commun. ACM 12, 10 (1969), 576–580.

[10] Edsger W. Dijkstra. 1968. A constructive approach to the problem of program correctness. BIT Numer. Math. 8, 3 (1968), 174–186.

[11] B. A. Liskov and J. V. Guttag. “Program Development in Java: Modeling Specification, and Refinement”. Addison Wesley, 2000.

[12] Daniel Jackson, “Alloy: A Lightweight Object Modelling Notation”, ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 11, No. 3.pp 256–298, 2002.

[13] Hähnle, R., Miller, G., and Schmidt, R. "The KeY System: Integrating Deductive and Model Checking for Software Verification." In International Conference on Formal Engineering Methods (ICFEM). Lecture Notes in Computer Science, vol 3785. Springer, Berlin, Heidelberg, 2005.

[14] Nipkow, T., Paulson, L., Wenzel, M., "Isabelle/HOL: A Proof Assistant for Higher-Order Logic". Lecture Notes in Computer Science, vol 2283. Springer, Berlin, Heidelberg, 2002.

[15] Bertot, Y., Castéran, P. "Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions". Springer, 2004.

[16] France, R., & Rumpe, B. “Domain specific modeling”. ACM Computing Surveys (CSUR), Vol 37 No., 167-205, 2005

[17] Palm, W. J. “Introduction to MATLAB for Engineers (5th ed.)”. McGraw Hill, 2019.

[18] MathWorks. (n.d.). “Simulink Documentation: Simulink Environment Fundamentals”. Retrieved October 26, 2025, from https://www.mathworks.com/help/simulink/simulink-environment-fundamentals.html

[19] Camunda. (n.d.). BPMN 2.0 Symbols - A complete guide with examples. Retrieved October 26, 2025, from https://camunda.com/bpmn/reference/

[20] Quest Software, Inc. erwin Data Modeler Navigator Edition User Guide (2021 R1). Retrieved from Quest Supportwebsite: https://support.quest.com/erwin-data-modeler/2021%20r1/technical-documents,2021

[21] SAP SE. SAP PowerDesigner, version 16.7. Accessed October 26, 2025. https://www.sap.com/products/powerdesigner-data-modeling-tools.html