Leveraging AI To Enhance the Software Development Lifecycle

by Joan R. Gentuba

B.S. in Computer Science, March 2001, University of Cebu
M.S. in Information Systems Engineering, December 2021, Johns Hopkins University

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Engineering

May 17, 2026

Praxis directed by

Karen Heine
Professorial Lecturer in Engineering and Applied Science

The School of Engineering and Applied Science of The George Washington University certifies that Joan Roncesvalles Gentuba has passed the Final Examination for the degree of Doctor of Engineering as of May 17, 2026. This is the final and approved form of the Praxis.

Leveraging AI To Enhance the Software Development Lifecycle

Joan Roncesvalles Gentuba

Praxis Research Committee:

Karen Heine, Professorial Lecturer in Engineering and Applied Science, Praxis Director

Thomas Mazzuchi, Professor of Engineering Management and Systems Engineering & of Decision Sciences, Committee Chair

Joseph Blackford, Adjunct Professor of Engineering and Applied Science, Committee Member

© Copyright 2026 by Joan R. Gentuba
All rights reserved

Dedication

This Praxis is dedicated to Francesca and Robert. Through the long days, you are the collective force behind my why.

Acknowledgements

I would like to thank the great Creator, who has been directing my steps since Day One. Thank you for the gift of dreams, for all the doors that have closed and those that have opened and led me here.

I am grateful for the support of family and friends, whose constant encouragement sustained me through every challenge. To my children, Francesca Preston and Robert Preston, thank you for the joy you bring into my life. I want you both to know that I love you very much and am very proud of you. I am especially grateful to Eugene Preston, who is always there for us and is the one person I can rely on for bouncing around ideas and all things tech, and whose shared conversations about AI motivated me to bring this research to fruition. To the two constants in my life, Jeaneth Sagales and Pablita Niebres, thank you for your prayers and words of support each time I come across a wall that seems unscalable.

A special thank you also goes to my advisor, Dr. Heine, for her valuable insights and for working with me as my research was taking shape. To the members of my cohort, thank you for the encouragement and collaboration as we all embarked on this journey together.

Finally, I acknowledge those that have come before me. This praxis would not be possible without the visionaries in Computer Science and Artificial Intelligence who continually drive the technology forward.

Abstract of Praxis

Leveraging AI to Enhance the Software Development Life Cycle

The Software Development Life Cycle (SDLC) has many inefficiencies such as manual tasks, that often lead to errors, costly rework, delays, and project overruns. While Artificial Intelligence (AI) tools have demonstrated improvements in development workflows, AI integration and adoption remain fragmented and limited to individual phases. This research examines the impact of end-to-end AI integration across all SDLC phases, utilizing a Discrete Event Simulation (DES) model with Monte Carlo sampling. The simulation was run 10,000 times with 100 tasks per run producing task-level observations. The results demonstrated an overall cycle time reduction of 58%, with phase-specific improvements ranging from 12.7% to 68.2%, primarily driven by the adoption rate per phase. The simulation results are compared against published benchmarks from literature to provide context through independently conducted studies. The simulation model serves as a decision tool for software engineering managers for prioritizing and planning AI implementation in their workflows and provides a mechanism for estimating cycle time improvements.

Table of Contents

Dedication iv

Acknowledgements v

Abstract of Praxis vi

List of Figures xi

List of Tables xii

List of Equations xiii

List of Symbols xiv

List of Acronyms xv

Chapter 1—Introduction 1

1.1 Background 1

1.2 Research Motivation 2

1.3 Problem Statement 2

1.4 Thesis Statement 3

1.5 Research Objectives 3

1.6 Research Questions and Hypotheses 4

1.7 Scope of Research 4

1.8 Research Limitations 5

1.9 Organization of Praxis 6

Chapter 2—Literature Review 7

2.1 Introduction 7

2.2 Artificial Intelligence Technologies 8
2.3 SDLC Planning Phase - KA9 - Software Engineering Management 13
2.4 SDLC Requirements Engineering Phase - KA1 - Software Requirements 16
2.5 SDLC Design/Prototyping Phase - KA3 - Software Design 19
2.6 SDLC Development/Implementation Phase - KA4 - Software Construction 20
2.7 SDLC Testing Phase - KA5 - Software Testing 21
2.8 SDLC Deployment Phase - KA6 - Software Engineering Operations 23
2.9 SDLC Maintenance Phase - KA7 - Software Maintenance 24
2.10 Discrete Event Simulation in Software Engineering 27
2.11 AI Productivity Studies and Benchmarks 28
2.12 Summary and Conclusion 30
Chapter 3—Methodology 32
3.1 Introduction 32
3.2 Dataset Description and Preparation 33
3.3 SDLC Phase Mapping 37
3.4 Simulation Model Design and Architecture 38
3.5 AI Effect Calculation 42
3.6 Time Allocation 44
3.7 Task Time Calculations 45
3.8 Sensitivity Analysis 48
3.9 Normality Test 49

3.10 Paired t-Test 54
3.11 Effect Size 56
3.12 Confidence Interval 57
3.13 Aggregation Across Simulation Runs 57
3.14 Simulation Flowcharts 58
Chapter 4—Results 64
4.1 Introduction 64
4.2 H1 Results 65
4.2.1 H1 Hypothesis Statement 65
4.2.2 H1 Statistical Analysis 66
4.2.3 H1 Summary 67
4.3 H2 Results 70
4.3.1 H2 Hypothesis Statement 70
4.3.2 H2 Statistical Analysis 70
4.3.3 H2 Summary 71
4.4 H3 Results 72
4.4.1 H3 Hypothesis Statement 72
4.4.2 H3 Statistical Analysis 73
4.4.3 H3 Summary 75
4.5 Sensitivity Analysis Results 76
4.6 Summary and Conclusion 79
Chapter 5—Discussion and Conclusions 81
5.1 Discussion 81

5.2 Conclusions 82

5.3 Contributions to Body of Knowledge 84

5.4 Recommendations for Future Research 85

References 87

List of Figures

Figure 3-1a: Normality Test – Mean Difference Probability Plot 51

Figure 3-1b: Normality Test – Mean Baseline Probability Plot 52

Figure 3-1c: Normality Test – Mean AI-Enabled Probability Plot..... 52

Figure 3-1d: Normality Test – Percent Improvement Probability Plot..... 53

Figure 3-2. Parameter Derivation Flowchart 60

Figure 3-3. Main Simulation Flowchart..... 61

Figure 3-4. Simulation Engine Flowchart..... 62

Figure 3-5. Sensitivity Analysis Flowchart 63

Figure 4-1. Simulation Distribution Histogram..... 68

Figure 4-2. Baseline vs. AI-Enabled Cycle Time Hours 69

Figure 4-3. Phase Improvements vs. H3 Thresholds 74

Figure 4-4. Sensitivity Results Tornado Diagram 79

List of Tables

Table 3- 1: Summary of Dataset Characteristics 35

Table 3- 2: Workflows and SDLC Phase Mapping 38

Table 3- 3: Simulation Parameters..... 42

Table 3- 4: AI Effect Values by Phase..... 44

Table 3- 5: Time Allocation by SDLC Phase 45

Table 3- 6.a: Excel Worksheet Example with Task 1 Cycle Time..... 47

Table 3- 6.b: Breakdown of AI Effect Calculation in Worksheet Example 48

Table 3- 7: Sensitivity Analysis Parameters and Variations..... 49

Table 4- 1: Simulation Control Parameters 64

Table 4- 2: Paired t-test Results..... 67

Table 4- 3: SDLC Phase Ranking by AI Effect and Hours Saved..... 71

Table 4- 4: H2 Statistical Summary..... 72

Table 4- 5: H3 Statistical Summary..... 75

Table 4- 6: Sensitivity Analysis Results 78

List of Equations

Equation 3- 1: Integration Level 36
Equation 3- 2: Impact Modifier 36
Equation 3- 3: Log-normal Distribution Equation 39
Equation 3- 4: Log-space Variance Equation 40
Equation 3- 5: Log-space Mean Equation 40
Equation 3- 6: Log-normal Mean Equation 40
Equation 3- 7: Log-normal Variance Equation 40
Equation 3- 8: Log-normal Coefficient of Variation Equation 40
Equation 3- 9: Box-Muller Transform Equation 41
Equation 3- 10: Log-normal Transformation Equation 41
Equation 3- 11: Expected Value (EV) Equation 43
Equation 3- 12.a: AI Effect Equation 43
Equation 3- 12.b: AI Effect (Simplified Form) Equation 43
Equation 3- 13: Baseline Phase Time Equation 45
Equation 3- 14: AI-Enabled Phase Time Equation 46
Equation 3- 15: Total Cycle Time Equation 47
Equation 3- 16: Paired T-Statistic Equation 54
Equation 3- 17: Paired Difference Equation 55
Equation 3- 18: Mean Paired Difference Equation 55
Equation 3- 19: Standard Deviation of Paired Differences Equation 55
Equation 3- 20: Standard Error Equation 55
Equation 3- 21: Cohen's d 56
Equation 3- 22: Confidence Interval Equation 57

List of Symbols

μ	Log-space mean of the log-normal distribution
σ	Log-space standard deviation of the log-normal distribution
σ^2	Log-space variance
$\hat{\sigma}$	Standard deviation of paired differences
$\bar{d}$	Mean paired difference
z	Standard normal variate
n	Number of tasks per simulation run
i	Task index
j	SDLC phase index
t	Paired t-statistic
df	Degrees of freedom
α	Significance Level
d	Cohen's d Effect Size
SE	Standard Error
$E[X]$	Expected value
Var[X]	Variance
$\mu_{Baseline}$	Mean baseline cycle time
$\mu_{AI-Enabled}$	Mean AI-enabled cycle time

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Networks
CI	Confidence Interval
COCOMO	Constructive Cost Model
CPDP	Cross-Project Defect Prediction
CRN	Common Random Numbers
CV	Coefficient of Variation
DES	Discrete Event Simulation
DL	Deep Learning
EV	Expected Value
GAI	Generative Artificial Intelligence
GPT	Generative Pre-trained Transformer
HA	Heuristic Algorithms
IT	Information Technology
KA	Knowledge Areas
KBS	Knowledge-Based Systems
ML	Machine Learning
MNB	Multinomial Naive Bayes
NER	Named Entity Recognition
NLP	Natural Language Processing
OAT	One-at-a-Time

RQ	Research Question
SDD	Software Design Documents
SDLC	Software Development Lifecycle
SFP	Software Fault Prediction
SQL	Structured Query Language
SUT	System Under Test
SVM	Support Vector Machines
SWEBOK	Software Engineering Body of Knowledge
TCNN	Transfer Convolutional Neural Network
UML	Unified Modeling Language

Chapter 1—Introduction

1.1 Background

The Software Development Life Cycle (SDLC) has been continuously evolving as software development teams work to improve inefficiencies while facing challenges to meet increasingly complex computer software requirements. Traditional methods such as the Waterfall model made way for Agile approaches that placed emphasis on iterative development and continuous feedback. Despite the advances in software development methodologies, software development teams continue to struggle with inefficiencies in software project management such as scope creep, resource allocation, manual processes, and bottlenecks that often result in project delays and costly rework. These in turn lead to budget overruns and missed deadlines.

The advances in Artificial Intelligence (AI) technologies have the potential to address inefficiencies in the SDLC. As AI tools become more widely available, integrating AI throughout all the SDLC phases is now becoming increasingly feasible.

This praxis examines the impact of AI integration across the SDLC on software development productivity and efficiency improvements. The study is based on survey data from the 2025 Stack Overflow Developer survey, which captures responses from developers who have utilized AI in their workflows, and project time allocation data from COCOMO II. A Discrete Event Simulation (DES) model is used to calculate the projected cycle time improvements from AI integration throughout the SDLC.

1.2 Research Motivation

There is a need to address the inefficiencies that are prevalent in the current SDLC. While organizations have started adopting AI for specific workflows, the implementations are sporadic and isolated to one or two SDLC phases, with prevalence focused towards coding assistance. The literature review indicates that AI integration into specific phases of the SDLC has demonstrated improvements for those phases. However, the current research is fragmented, with individual studies focusing on one phase. The aim of this study is to evaluate the impact of AI integration across all phases of the SDLC and to enable calculation of productivity and efficiency improvements using simulation.

1.3 Problem Statement

The current processes in the SDLC without AI integration are inefficient. These involve manual tasks that frequently lead to errors and rework, resulting in budget overruns and high project failure rates.

The software development life cycle processes have been continuously improving throughout the years, however "certain challenges, such as error-prone manual coding, delayed feedback loops, and resource allocation, have persisted, necessitating further innovation" (Alenezi & Akour, 2025). These inefficiencies result in high failure rates. Conger (2013) reported that "53% of projects overran original schedules by an average of 222%," with 31% canceled, only 9% of large company projects completed on time and within budget, and only 42% delivered planned benefits. More recent research confirms that high failure rates remain a persistent challenge in the IT sector, with poor communication, inadequate organizational support, and team-related factors identified as

the most frequently cited contributors to project failure (Gupta et al., 2019). Artificial intelligence tools show promise for addressing software development issues, but their implementation is isolated in specific SDLC workflows. An evaluation of the end-to-end integration of AI is necessary to understand how implementation across all SDLC phases can transform software development in terms of increased productivity and improved efficiency.

1.4 Thesis Statement

An analysis of the end-to-end integration of artificial intelligence in the software development life cycle is required to evaluate its impact on productivity and efficiency.

A Discrete Event Simulation (DES) model with Monte Carlo sampling is used to calculate project cycle time improvements from AI integration across the SDLC phases. The model utilizes parameters derived from survey data from the 2025 Stack Overflow Developer Survey, and time allocation data from COCOMO II.

1.5 Research Objectives

This Praxis aims to provide software engineering managers with a modeling tool that enables the calculation of cycle time improvements from AI integration across the SDLC phases. The research objectives are as follows:

1. Examine current studies on the impact of AI integration in software development
2. Develop a Discrete Event Simulation model that projects cycle time improvements for AI integration in the SDLC
3. Validate simulation projections through statistical analysis and comparison against published AI improvement benchmarks from the literature review

1.6 Research Questions and Hypotheses

The following research questions (RQ) and hypotheses (H) were formulated to address the praxis research objectives.

RQ1: Does AI integration in the SDLC improve software development effectiveness?

RQ2: How does AI integration affect performance across the different SDLC phases?

RQ3: Can a Discrete Event Simulation model accurately predict phase-specific cycle time improvements from AI integration across individual SDLC phases?

H1: AI integration improves overall software development effectiveness by 20%.

H2: AI integration demonstrates the highest improvements in the Development phase with a 25% reduction in phase cycle time.

H3: AI integration cycle time improvements vary by SDLC phase, phases with low adoption achieve less than 30% reduction while higher-adoption phases achieve greater than 50% reduction in phase cycle time.

1.7 Scope of Research

The scope of this research focuses on projecting the impact of AI integration across the SDLC phases using Discrete Event Simulation, with parameters derived from the 2025 Stack Overflow Developer Survey and time allocation data from COCOMO II. The study has the following scope boundaries:

- The data source for AI effect calculation is filtered for professional developers who are currently using AI in their development workflows and who provided responses to impact statements from AI Agent use.

- The time-allocation parameter in the model is adapted from COCOMO II for the six SDLC Phases: Planning, Requirements, Design, Development, Testing, and Deployment. Maintenance was excluded due to lack of a citable COCOMO II time allocation.
- The outcome variable is set to projected cycle time reduction in hours.
- The simulation type uses Monte Carlo sampling and does not include other variability contexts such as resource constraints, organizational size or dynamics and culture, and inter-task dependencies.
- The results from the model represent projections where developers actively use AI in their workflows.

1.8 Research Limitations

The research is limited to perception-based survey data on developer responses to usage of AI tools in their development workflows, and impact statements on AI agent use. The parameters are derived from single-year survey data from 2025 and do not track the progression of AI adoption patterns across multiple years. The simulation models the project tasks independently and does not account for resource constraints, inter-task dependencies, organizational size or dynamics, or queueing effects. The time allocations from COCOMO II are adapted to map to the SWEBOK v4.0 Knowledge Areas (KA). The simulation model does not include the Maintenance phase as there is no citable time allocation for this phase in COCOMO II.

The simulation is calibrated to routine, well-defined development tasks and may not generalize to highly complex or research-intensive projects, as discussed in Section 5.2.

1.9 Organization of Praxis

The Praxis follows the recommended template provided by The George Washington University Graduate School of Engineering and Applied Science. Chapter 1 covers the introduction to the research topic and provides the background and the research motivation. It presents the problem and thesis statements, providing the research objectives and poses the research questions and hypotheses. Chapter 1 also includes the praxis scope and its limitations. This is followed by the literature review of existing research in Chapter 2. Chapter 3 discusses the methodology that was developed for the simulation model and describes the statistical analysis process conducted for the research. Chapter 4 provides an in-depth analysis of the simulation results and statistical validation of the research hypotheses. Finally, Chapter 5 provides a summary and conclusion for the praxis, outlining its contributions to the body of knowledge and offers recommendations for future research.

Chapter 2—Literature Review

2.1 Introduction

This chapter reviews current research on the impact of artificial intelligence integration in the software development life cycle. The review examines AI technologies and their applications across the individual SDLC phases. It also covers the DES as a methodology for modeling software processes, the theoretical basis for the AI Effect calculation, and the published AI productivity benchmarks that the praxis uses for external validation.

This chapter is organized as follows. Section 2.2 provides a review of AI technologies and their relevant subfields in software development. The SDLC phases are categorized according to the "Guide to the Software Engineering Body of Knowledge" (SWEBOK) Knowledge Areas (KA). The review focuses on the following KAs, as these are relevant to the research: "KA1 – Software Requirements", "KA3 – Software Design", "KA4 – Software Construction", "KA5 – Software Testing", "KA6 – Software Engineering Operations", "KA7 – Software Maintenance", and "KA9 – Software Engineering Management" (SWEBOK Guide v4.0). Sections 2.3 through 2.9 examine each phase of the SDLC in detail, mapping each phase to its corresponding SWEBOK Knowledge and exploring specific AI applications and technologies relevant to that phase. Section 2.10 covers Discrete Event Simulation as a methodology for modeling software processes. Section 2.11 reviews published AI productivity studies and benchmarks used for external validation of the simulation results. Section 2.12 provides a summary of the findings and identifies gaps in the current literature.

2.2 Artificial Intelligence Technologies

Artificial intelligence has multiple definitions. One definition from the European Commission as cited by Sheikh et al. (2023) defines AI as "Systems that display intelligent behaviour by analysing their environment and taking actions—with some degree of autonomy—to achieve specific goals". Shahkarami et al. (2014, as cited in Shehab et al., 2020) described AI as "the capability of [a] digital computer to perform tasks in ways that are smarter than human ability". Singh & Sambhav (2023) put emphasis on AI's role in making machines perform traditionally human tasks and "engage in actions that humans see as intelligent". These capabilities and the recent advances in AI have impacted the software development industry with tools like coding and test automation assistants. This praxis examines the AI technologies that are most relevant to the software development life cycle. These are also the underlying technologies that are integrated into most contemporary AI tools. The AI technologies are as follows: Machine Learning, Natural Language Processing, Deep Learning, Neural Networks, Knowledge-Based Systems, Generative AI, and Multimodal AI.

Machine Learning (ML).

Mitchell (1997, as cited in Borges et al., 2020) defined Machine Learning as a subfield of artificial intelligence "composed of several algorithms for approximating patterns in data discovery". ML uses patterns to train models on datasets, enabling the models to recognize patterns and make predictions or decisions based on new data. Machine Learning encompasses several techniques; each is useful for specific types of problems and data availability.

Machine Learning Techniques.

Supervised Learning trains models based on labeled datasets where input-output pairs are known. Durelli et al. (2019) explains that "Supervised learning is used when for each input variable (i.e., X) there is a corresponding output variable (i.e., Y)". In this model, algorithms such as Neural Networks map inputs to outputs using the training data.

Unsupervised Learning differs from Supervised Learning in that it trains the models based on unlabeled data and identifies patterns and relationships within the data. Durelli et al. (2019) explains that "… unsupervised learning is when only input data are available, so the goal is to understand the underlying relationship between the inputs". This approach deals with clustering problems, where the objective is "to determine whether the inputs fall into distinct groups" (Durelli et al., 2019).

Reinforcement Learning uses reward-based mechanisms to train the models. Davis et al. (2016) defines this method as "an approach where general, broad-based negative and positive reinforcement is used to guide learning". Agents train for optimal decision-making through trial and error, where good results are rewarded and bad results are penalized, enabling the agents to learn based on cumulative rewards over time.

Semi-Supervised Learning is a hybrid approach that combines supervised and unsupervised learning techniques. This method utilizes a small set of labeled data to identify patterns and then analyzes the underlying structure and distribution of unlabeled data. Ahmad et al. (2020) described the method as a model that deduces a "function grounded on [a] lesser amount of labeled training dataset and [a] huge amount of unlabeled dataset".

Self-Supervised Learning enables models to predict parts of the input from other parts. The models train on pre-text tasks without human-annotated labels, learning to

generate their own labels from the data structure itself. Prakash (2024) describes how this method trains neural networks by hiding text segments and requiring the prediction of missing parts: “The neural network pays selective attention to the words comprising the surrounding context of the missing part. It then learns and represents the meaning of each word as its embedding” and uses these representations to generate new content.

Transfer Learning “aims to leverage knowledge from a related domain (called source domain) to improve the learning performance or minimize the number of labeled examples required in a target domain” (Zhuang et al., 2020). This approach is useful when the target model data are absent or limited. An example of this method is the Transfer Convolutional Neural Network (TCNN), which Qiu et al. (2019) proposed for “mining transferable semantic features” in cross-project defect prediction.

Data Mining identifies patterns in large datasets and analyzes the information to make predictions concerning new data. Witten et al. (2011) define data mining as:

> The process of discovering patterns in data. The process must be automatic or (more usually) semiautomatic. The patterns discovered must be meaningful in that they lead to some advantage, usually an economic one. The data [are] invariably present in substantial quantities. (p. 3).

In the context of software engineering, extensive amounts of data are generated from all phases of the software development life cycle (Sofian et al., 2022). Witten et al. (2011) further stated that “intelligently analyzed data is a valuable resource. It can lead to new insights, and, in commercial settings, to competitive advantages” (p. 2). Data mining techniques can extract patterns from the data to identify trends that support decision-making and performance optimization.

Heuristic Algorithms (HA) are "designed to solve a problem faster and more efficiently than the traditional methods by sacrificing optimality, accuracy, precision, or completeness for speed" (Sofian et al., 2022). In software engineering, HA can be used for complex decision-making problems that require expensive overhead. HA can be a standalone technique or combined with other AI techniques as a hybrid solution (Sofian et al., 2022).

Deep Learning (DL).

LeCun et al. (2015) described Deep Learning as a technique that "allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction". LeCun et al. (2015) noted that the key distinguishing feature of deep learning is that the "layers of features" are learned from data and not "designed by human engineers". For example, in image processing, each layer builds on the previously learned layer and progresses from simple to complex. The first layer is the basic features or the edges. The second layer is a combination of these edges that form patterns. The third layer combines these patterns into object parts, and the final layers complete the image from the parts it recognized.

Natural Language Processing (NLP).

Natural Language Processing enables computers to understand and generate responses in human-like language (Necula et al., 2024). Liu et al. (2022) defined NLP as applying "various levels of linguistic analysis to the target, such as tokenization, part-of-speech tagging, decomposition, [and] spelling/ grammar error recovery". NLP encompasses techniques that revolve around language translation, speech recognition, sentiment analysis, and text generation. In the context of the software development life

cycle, NLP can process large amounts of natural language data, such as those in user stories and requirements documents.

Generative AI (Gen AI).

Banh & Strobel (2023) describes Generative AI as having "the primary goal of generating new, probabilistically produced data (i.e., content) with varying outputs based on the same input" and having "capabilities to generate high-quality, contextually relevant content almost indistinguishable from human-created work". They frame generative AI as representing "new paradigms of machine processing, shifting from data-driven, discriminative AI tasks toward sophisticated, creative tasks", defining two main characteristics at its core: Generativity or the ability to create new and original content, and Variance, where valid probabilistic outputs can result from the same input (Banh & Strobel, 2023). The authors highlighted the use of "prompting", a process in which users employ natural language to interact with a GAI application to produce their intended output.

Knowledge-Based Systems (KBS).

Knowledge-Based Systems include "expert systems, corporate databases, and other organizational IT applications which help corporations manage their knowledge assets effectively" (Dutta, 1997). The purpose of building a KBS is to capture a company's "knowledge assets" into an organization's information system that can serve as a useful "tangible knowledge asset" (Dutta, 1997). Dutta (1997) characterized KBS using a three-level hierarchy: Data, comprising raw numbers such as sales figures; Information, which includes organized data such as "sales by region" in reports; and

Knowledge, representing how users understand the information. A KBS does more than just store information; it also captures how users interpret the information.

Multimodal AI.

Multimodal AI models are "deep learning models that can learn across more than one data modality" (Munikoti et al., 2024). These systems process various forms of input such as text, images, video, audio, and other data formats. Multimodal AI can perform both single-modality tasks, or tasks that involve a single data type, and multi-modality tasks, or tasks that require simultaneous processing across multiple modalities (Munikoti et al., 2024).

2.3 SDLC Planning Phase - KA9 - Software Engineering Management

The SWEBOK Guide v4.0 defines Software Engineering Management as "a collection of work activities involved with planning, estimating, measuring, controlling, coordinating, leading, and managing risk factors for a software project to help ensure that software products and software engineering services are delivered efficiently, effectively, and to the stakeholders' benefit" (Washizaki, 2024). The key activities in planning include project initiation, feasibility analysis, effort estimation, and risk management. The planning phase defines the project's scope, objectives, and procedures; therefore, it is considered critical to a project's success. However, manual planning becomes increasingly difficult as project complexity grows, since effective planning involves coordinating multiple variables, such as team capacity, changing requirements, scheduling, resource allocation, and risk management.

2.3.1 Key Activities in the Planning Phase

The key activities in the planning phase include Project Initiation, Feasibility Analysis, Project Plan Development, Resource Allocation, Risk Management, and Communication Planning. Project Initiation "focuses on reviewing the software requirements and determining the need, scope, feasibility, and authorization for a software project" (SWEBOK Guide v4.0). Feasibility Analysis determines project viability across financial, operational, scheduling, and legal aspects, informing the leadership whether a project is worth undertaking (SWEBOK Guide v4.0). Project Plan Development creates a roadmap that contains detailed information concerning the scope, timeline, resource availability, schedules, milestones, and deliverables. "Detailed plans based on a known architecture, requirements, and constraints are used to reduce risk and cost" (SWEBOK Guide v4.0). Resource Allocation is "based on and constrained by the availability of resources and their optimal use, and by issues relating to personnel" (SWEBOK Guide v4.0). Proper resource allocation ensures that resources are available to meet project timelines and target the goals outlined in the project. Risk Management ensures that a proper analysis is performed and that risks are mitigated. According to the SWEBOK Guide v4.0, "Risk management entails identifying risk factors, analyzing probability and potential impact of each risk factor, prioritizing risk factors, and developing risk mitigation strategies to reduce the probability of a negative event and to minimize the negative impact if a risk factor becomes a problem." Communication Planning ensures effective communication among stakeholders. This involves establishing effective communication channels and defining the frequency and modes of communication. The SWEBOK Guide v4.0 highlights the use of Communication Tools

and how they can “help provide timely and consistent information to relevant stakeholders involved in a project”.

2.3.2 AI in the Planning Phase

The praxis identified current research where artificial intelligence technologies can address challenges in the planning phase. According to Karim Zadeh et al. (2024), “AI can automate routine and repetitive tasks, freeing up team members to focus on higher-value activities.” Dam et al. (2018) explored how AI assists by “automating repetitive, high-volume tasks to enable project analytics for estimation and risk prediction”, and proposed a framework for descriptive, predictive, and prescriptive analytics for supporting agile project management.

Improving estimation accuracy is another area where AI can improve the planning phase. Karim Zadeh et al. (2024) noted that “AI algorithms can analyze historical data and identify patterns to generate more accurate estimates for future tasks.” Sofian et al. (2022) noted how ML has been widely studied for “building prediction models, estimating software development efforts, and improving the accuracy of other estimation techniques.”

AI also supports resource allocation by analyzing “team members’ skill sets, workloads, and past performance to match them with the most suitable tasks” (Karim Zadeh et al., 2024). Shamim (2024) reported that AI “can optimize resource allocation by analyzing project requirements, skill sets, and availability”, highlighting the use of chatbots and virtual assistants for facilitating communications among stakeholders “by providing real-time updates, answering queries, and facilitating collaboration among project team members.”

Shehab et al. (2020) reported on how Knowledge-Based Systems (KBS) "are used to manage requirements and decisions taken during the design process", noting how domain ontologies enable a "good understanding of a problem domain and the detection of incompleteness and ambiguities because they encompass the strengths of KBS and NLP-based systems."

2.4 SDLC Requirements Engineering Phase - KA1 - Software Requirements

The Requirements Engineering phase establishes what a software application must accomplish and prevents costly rework from poorly defined requirements, which often leads to project failure from scope creep, budget overruns, and delivery delays. The SWEBOK Guide v4.0 formally defines a Software Requirement as:

> "• a condition or capability needed by a user to solve a problem or achieve an objective;
>
> • a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document;
>
> • a documented representation or capability as in (1) or (2) above"

2.4.1 Key Activities in Requirements Engineering

The SWEBOK Guide v4.0 identifies key activities during requirements engineering phase including requirements elicitation, analysis, specification, validation, and management. Requirements Elicitation involves gathering requirements from stakeholders through meetings, interviews, surveys, and observation. Requirements Analysis follows elicitation. The gathered requirements are examined, with models and illustrations created to ensure that these are complete, clear, and feasible. The next

activity in this phase is Requirements Specification. During requirements specification, the requirements are used to construct a detailed Software Requirements Specification (SRS) document that categorizes the requirements into Functional Requirements (FRs) and Non-Functional Requirements (NFRs). The Requirements Validation activities then verify that the documented specifications accurately reflect stakeholder expectations and that the SRS contains no ambiguities or issues that could compromise software quality in subsequent SDLC phases. As requirements continue to evolve throughout the project life cycle, the Requirements Management activities track evolving stakeholder needs, ensuring that the project goals are aligned with updated requirements. The SWEBOK Guide differentiates Requirements Management from Requirements Development by noting that the latter focuses on "reaching an agreement on what software is to be constructed", while Requirements Management is concerned with "maintaining the agreement over time" (SWEBOK Guide v4.0).

2.4.2 AI in Requirements Engineering

The AI technologies reviewed for the requirements engineering phase include NLP, ML, and DL. Liu et al. (2022) reviewed research demonstrating that domain-specific knowledge bases can be integrated with NLP-powered chatbots to support requirements elicitation, allowing these systems to capture more targeted stakeholder needs. Trinh et al. (2024) conducted a systematic literature review and found that requirements elicitation was the most frequently addressed software development activity in AI integration research compared to other SDLC phases and that NLP was the second most common AI model type used. They noted AI's use in "translating human language

and sentiment into appraisals of what the software should be" when eliciting requirements.

In the requirements analysis process, research demonstrated how Support Vector Machines (SVM) and Multinomial Naïve Bayes (MNB) can classify requirements into functional and non-functional categories, with reported accuracies of 88% and 91%, respectively (Liu et al., 2022). Ahmad et al. (2020) reviewed ML techniques for software requirements identification and classification on Stack Overflow, describing their effectiveness compared to traditional NLP techniques. Binkhonain and Zhao (2019) conducted a systematic review of ML-based approaches for identifying and classifying non-functional requirements, highlighting the use of precision and recall matrices to measure the performance of ML algorithms.

Named Entity Recognition (NER) techniques can automatically identify use case elements, and NLP-based approaches enable the creation of UML diagrams from SRS documents (Liu et al., 2022) in requirements specification activities. Liu et al. (2022) also explored ML applications in requirements engineering, describing how data mining enables pattern recognition and outlining the distinction between supervised and unsupervised learning approaches.

Liu et al. (2022) reported that in requirements validation, Random Forest classifiers performed better than other models when identifying ambiguous requirements, while J48 Decision Tree algorithms also demonstrated strong performance in ambiguity classification.

Necula et al. (2024) examined the integration of NLP in software requirements engineering, tracking how advanced computational techniques developed from 1991 to

2023 have refined and automated software requirements engineering tasks. Necula et al. (2024) also reviewed the role of Deep Learning in requirements engineering task automation, finding that these techniques improved the accuracy of requirements analysis.

2.5 SDLC Design/Prototyping Phase - KA3 - Software Design

The Design phase translates requirements into system architecture and produces detailed Software Design Documents (SDD) and prototypes that stakeholders use to visualize the system prior to software development. The SWEBOK Guide v4.0 defines Software Design as a "practical process of transforming a problem statement into a solution statement".

2.5.1 Key Activities in Design/ Prototyping

The key activities in the Design/ Prototyping phase include Design and Prototyping. The Design process outlines the system architecture, encompassing all software and hardware components, data flow processes, and the user interface. The detailed design specification serves as the blueprint for software development. Prototyping creates working models that validate the design concepts before full development begins. This enables the early identification of potential design flaws and facilitates the necessary adjustments, thereby reducing costly rework later in the development phases. Typical prototyping activities include building wireframes and detailed user interface mockups.

2.5.2 AI in Design/ Prototyping

Applications of AI in the Design/ Prototyping phase include NLP-based generative AI tools and ML techniques. Prakash (2024) examined NLP-based Gen AI

tools, highlighting the use of GPT-powered diagramming tools such as DiagramGPT and ChartAI. Across these tools, diagram types generated from natural language prompts include flowcharts, architecture diagrams, sequence diagrams, Gantt charts, Entity-Relationship diagrams, and class diagrams. Borges et al. (2020) reported that ML techniques could predict design artifact quality and enable earlier quality assessment during the development process.

2.6 SDLC Development/Implementation Phase - KA4 - Software Construction

"Software construction is the detailed creation and maintenance of software through coding, verification, unit testing, integration testing, and debugging" (SWEBOK Guide-v4.0). This phase translates the design specifications that are defined in the design phase, into functional software using programming languages and development tools.

2.6.1 Key Aspects of the Development/Implementation Phase

The Development/Implementation phase involves transforming the software design into a fully functional application that meets the specified requirements. The Software Construction KA in the SWEBOK Guide v4.0 outlines a number of activities for the successful completion of the Development/Implementation phase. This review focuses on the following key activities considered relevant to the praxis.

Coding and Unit Testing. Developers write the code for the software components that comprise the application system and conduct unit testing. Individual modules or units of the software are created and tested in isolation to ensure that each unit functions as expected, and issues are identified and fixed at the component level.

Integration. Once the individual components and modules have been completed and tested, they are integrated into one complete system. During the integration process,

the various modules are combined, ensuring that individual modules work together seamlessly and in tandem. Integration testing is conducted to identify issues that occur from the interactions among the individual components or modules.

Documentation. This process produces detailed documentation that covers the software's functionality, design, and code structure. Software development teams reference this documentation when performing updates and maintenance on the system, making this a key activity of the development phase.

2.6.2 AI in Development/Implementation

Prakash (2024) examined how Generative AI Tools (GAITs) function in the development phase, demonstrating their applications in code and debugging assistance, as well as documentation. Chatterjee et al. (2024) conducted a controlled experiment at ANZ Bank to evaluate GitHub Copilot's effectiveness in a corporate development environment. The study showed that engineers using the AI tool completed tasks 42.36% faster on average, and static code analysis showed that the Copilot-assisted code had fewer code smells. Machine Learning (ML) is also increasingly adopted in system development phases for source code analysis and defect detection (Sofian et al., 2022). Natural Language Processing (NLP) can analyze "code comments, commit messages, and issue trackers" for technical debt identification, including documentation debt (Pandi et al., 2023).

2.7 SDLC Testing Phase - KA5 - Software Testing

The Testing phase evaluates software against the standards outlined in the requirements specifications to verify that it performs as expected before deployment. The SWEBOK Guide v4.0 defines software testing as "the dynamic validation that a

system under test (SUT) provides expected behaviors on a finite set of test cases suitably selected from the usually infinite execution domain".

2.7.1 Key Activities in Software Testing

The key activities in the Testing phase include Functional Testing and Structural Testing. Functional Testing validates software behavior against the requirements without examining the internal code structure. Test cases are derived directly from the documented specifications. This approach is commonly known as "Black-box" testing because testers work without knowing the underlying implementation (Durelli et al., 2019). Structural Testing takes the opposite approach, examining internal code components and logic paths. Often called "White-box" testing, this method requires knowledge of the code's internal structure to confirm that all code execution paths are validated (Durelli et al., 2019).

2.7.2 AI in Software Testing

In defect prediction, Vashisht et al. (2015, as cited in Shehab et al., 2020) proposed an ANN-based framework that was trained on historical defect data from 45 real-world projects to predict defects in new projects and improve software quality. When validated across 15 real-time projects, the framework showed that observed defects consistently fell within the predicted defect ranges, achieving up to 90% accuracy. Pandey et al. (2021), in a systematic literature review, demonstrated that machine learning techniques outperform traditional statistical models when classifying "fault-prone" or "non-fault-prone" class modules in Software Fault Prediction (SFP). Qiu et al. (2019) demonstrated how Cross-Project Defect Prediction (CPDP) effectively transfers defect prediction knowledge between projects by proposing a transfer convolutional neural network

(TCNN) approach that addressed classification error and cross-project distribution divergence.

In automated testing, Prakash (2024) evaluated AI tools, including GitHub Copilot, Tabnine, and Codeium, across systems of varying complexity (low, medium, and high), and showed how generative AI can improve the testing process through automated test creation. Trinh et al. (2024) found that AI techniques, including NLP models for test case generation, contribute to "reducing test-writing time and costs".

2.8 SDLC Deployment Phase - KA6 - Software Engineering Operations

The Deployment phase transitions the software application from the development and testing environments to the final live production environment. The SWEBOK Guide v4.0 defines software engineering operations as "the set of activities and tasks necessary to deploy, operate and support a software application or system while preserving its integrity and stability".

2.8.1 Key Operations Process Activities in Deployment

In the Deployment phase, the software application is promoted from the development and testing environments to the final live production environment. Software Engineering Operations activities are grouped into three separate processes: Operations Planning, Operations Delivery, and Operations Control.

Operations Planning defines service-level agreements (SLAs) and develops plans to manage service availability, continuity, and the ability to meet customer commitments. Key activities involve software backup, disaster recovery, and effective information security management through policies, training, and integrated security mechanisms. Operations planning also addresses the management of different environments (e.g.,

development, testing, pre-production, and production) with modern practices emphasizing automation to improve efficiency.

Operations Delivery involves operational testing, verification, and acceptance to ensure readiness for deployment. The deployment/ release engineering process is documented for a seamless transition to production. Rollback and data migration strategies, change management for controlled software updates, and problem management are combined to minimize disruptions.

Operations Control encompasses incident management for restoring normal service operations, as well as continuous monitoring using performance metrics and indicators to maintain system health. Proactive monitoring and problem management minimize disruptions, and operations service reports drive decision-making and continuous improvement. Automated systems, analysis, and reporting tools support and maintain operational health and system performance (SWEBOK Guide v4.0).

2.8.2 AI in Deployment

Sofian et al. (2022) found that Machine Learning (ML) is the predominant AI technique used across the different phases of the SDLC, including decision-making tasks and supporting predictive analysis. However, the deployment phase remains one of the least studied software engineering phases for AI applications (Sofian et al., 2022). Lei et al. (2021) examined Heuristic Algorithm (HA) approaches for cost-optimized deployment strategies, finding that these strategies reduce resource waste and lower operating costs through virtual machine deployment and server type selection.

2.9 SDLC Maintenance Phase - KA7 - Software Maintenance

The SWEBOK Guide v4.0 defines Software Maintenance as "the totality of activities required to provide cost-effective support for software in operation". The categories for maintenance include corrective, preventive, adaptive, additive, and perfective maintenance. Each of these categories address the different aspects of post-deployment software application support. The maintenance phase is included in this literature review as this phase is one of the phases mapped to the survey workflows. However, this phase is excluded from the simulation model due to the lack of citable time allocation source from COCOMO II.

2.9.1 Key Categories in the Maintenance Phase

The key categories or activities in the Maintenance Phase include Corrective Maintenance, Preventive Maintenance, Adaptive Maintenance, Additive Maintenance, and Perfective Maintenance.

Corrective Maintenance addresses defects that were not detected during testing. The SWEBOK Guide v4.0 defines this phase as "Reactive modification (or repairs) of a software product performed after delivery to correct discovered problems." These corrections are necessary to maintain system reliability and ensure continued functionality.

Preventative Maintenance takes a proactive approach. Rather than wait for faults to arise, this phase involves "modification of a software product after delivery to correct latent faults in the software product before they occur in the live system" (SWEBOK Guide v4.0). The aim is to identify and address potential issues before they become problems in the operational system, helping keep the software stable.

Adaptive Maintenance keeps software compatible with evolving environments. The SWEBOK Guide v4.0 describes this phase as the "modification of a software product performed after delivery to keep a software product usable in an evolving environment." Operating system upgrades, hardware modifications, and updates to software dependencies that could impact system compatibility are included in this process.

Additive Maintenance provides new features that enhance the software's usability, performance, and maintainability. In this type of maintenance, the focus is on adding new functionality rather than improving existing functionality.

Perfective Maintenance aims to improve rather than expand the software. As described in the SWEBOK Guide v4.0, this phase comprises "the modification of a software product after delivery to provide enhancements for users, improvement of program documentation, and recoding to improve software performance, maintainability, or other software attributes." This form of maintenance activity differs from Additive Maintenance in that the focus is on improving existing functionality by refining user interfaces and optimizing code.

2.9.2 AI in the Maintenance Phase

Pandey et al. (2021) examined Machine Learning (ML) approaches for software fault prediction and found that having the ability to predict defective modules early leads to better resource allocation, reducing testing effort and maintenance costs. Their research also found that ML approaches enable the test process to focus on problematic modules and identify candidates for refactoring.

Azeem et al. (2019) conducted a systematic review and meta-analysis of ML algorithms for detecting code smells, comparing the effectiveness of ML techniques such

as Decision Trees, SVMs, JRip, and Random Forest. They noted that code smells make code more change- and fault-prone and difficult to understand, increasing maintenance effort and costs.

Rahmaniar (2024) examined ChatGPT, finding that its Natural Language Processing (NLP) capabilities and NLP-based tools for automated generation of software documentation and code annotation facilitate better code comprehension and maintenance.

Pandey et al. (2021) also discussed Deep Learning (DL) for software fault prediction, noting that these approaches are useful in scenarios where datasets do not have "extracted features set and/or class labels". Their study observed that DL architectures improve accuracy with larger training datasets. However, they also noted that DL algorithms are inherently complex and "require hyperparameters tuning of many control parameters".

2.10 Discrete Event Simulation in Software Engineering

Kellner et al. (1999) established the classification framework for software process simulation modeling. Their work addresses three questions: Why simulate, What to include, How to implement. They identified six purposes for simulation, including technology adoption, which is relevant to this praxis' objective of evaluating AI integration in the SDLC. Their work identified multiple simulation approaches, including Discrete Event Simulation, and hybrid approaches. DES is the appropriate paradigm when detailed representation of activities is required (Kellner et al., 1999). This research maps to their framework as follows: research purpose is technology adoption, the scope is a single-project SDLC with task-level modeling, and the model utilized is DES with Monte Carlo sampling.

Madachy (2008) demonstrated that discrete event simulation has been applied to software engineering processes to analyze detailed process activities, which include resource utilization and phase-level modeling. Madachy (2008) identified the log-normal distribution as appropriate for modeling software task durations, which accounts for the right-skewed nature of software tasks.

Paired comparison designs using Common Random Numbers reduce variance and increase statistical power (Banks et al., 2010). This is the basis for the paired comparison design used in the research methodology to compare baseline versus AI-enabled cycle times. Banks et al. (2010) also established that sufficient replications are needed for replication stability. This informed the decision for the number of simulation runs used in this research.

2.11 AI Productivity Studies and Benchmarks

The research reviewed published AI productivity studies to provide external validation benchmarks for the simulation projections. The simulation results are compared against these benchmarks in Chapter 4 to assess whether the results are within the range of values from productivity studies on AI adoption. Given that AI adoption in the SDLC is an emerging area, the majority of the available sources for this benchmark review come from industry reports or preprints rather than peer-reviewed publications.

Peng et al. (2023) conducted a randomized controlled trial evaluating the impact of GitHub Copilot on developer productivity. The study recruited 95 professional programmers via Upwork, randomly assigned to a control group with no Copilot, and a treatment group with Copilot. Developers completed a standardized task under controlled conditions. The result from the study showed the developers in the treatment

group had 55.8% faster completion, with 95% CI [21-89%]. Of the 95 participants recruited, 35 completed the task and the productivity results come from this subset of participants. The experimental design in this study makes it an appropriate benchmark for validating the simulation results. However, this experiment is not peer-reviewed, and the single standardized task in the study does not capture real-world, multi-phase software development projects.

Cui et al. (2025) extended the work of Peng et al. (2023) by conducting three separate field randomized controlled trials at Microsoft, Accenture, and an anonymous Fortune 100 electronics company, although the Accenture experiment was excluded because of layoffs that affected 42% of study participants. The experiments ran from two to eight months, with a sample size of approximately 4,867 professional developers performing real development work. The result showed a 26.08% increase in completed pull requests per week, 13.55% more code commits, and 38.38% more builds, with no drop in code quality. The results also showed that short-tenure and junior or less experienced developers gained the most, with an improvement of 21-40%, while senior developers showed little to no statistically significant improvement. This study is a SSRN working paper, a working paper repository commonly used by researchers at institutions including MIT, Princeton, and Wharton, and is also not peer-reviewed. The study has a large sample size, with real production work at two independent organizations over an extended period. However, it measures task throughput and does not directly measure cycle time (Cui et al., 2025).

Chatterjee et al. (2024), as previously mentioned in Section 2.6 of this document, reported results from a controlled experiment with ANZ Bank. The experiment was an

A/B design involving over 100 software, cloud, and data engineers who were tasked to complete Python algorithmic coding challenges with and without Copilot. The experiment was conducted over a six-week period. The results showed that the Copilot group completed tasks 42.36% faster than the control group, with productivity improvements observed across all developer proficiency levels. The study also reported statistically significant improvements in code quality metrics, with fewer bugs and code smells. Although the study used self-reported task completion times and not software development workflows, it represents a controlled experiment conducted in a real enterprise environment, independent of the organizations that developed Copilot, which provides meaningful credibility when using it as a benchmark.

Ziegler et al. (2024) provided the only peer-reviewed study in the research benchmark review. The study confirmed through survey and telemetry data that productivity gains from coding tools are measurable and consistent across all developer skill levels. However, their study measures perceived productivity and does not report on actual task completion time.

2.12 Summary and Conclusion

The literature review has demonstrated the potential of artificial intelligence when integrated into individual SDLC phases, as described in Sections 2.3 – 2.9. However, these findings are phase specific, with each study focused on examining the impact of AI within a single phase. There is limited research on studies that measured the cumulative effect of AI integration throughout the SDLC phases. This limits the ability of software engineering managers to assess the total impact of AI adoption across the full development lifecycle process. The published benchmarks show improvements but are

varied in their scope and what they measured, with some measuring task completion times, and others measuring task throughput or perceived productivity gains. Most of these experiments focused on activities related to the development phase, with limited coverage of the other SDLC phases. The literature review did not identify a benchmark study that provided phase-specific projections demonstrating where AI has the highest or lowest returns. Reviewing DES as a methodology for modeling software processes showed how the model can be applied to project AI-driven improvements across the SDLC. The DES model developed in this praxis projects phase-specific and overall cycle time improvements from AI integration throughout the SDLC, from the planning phase all the way through to deployment. The methodology is presented in Chapter 3.

Chapter 3—Methodology

3.1 Introduction

This chapter describes the research methodology used to demonstrate the productivity improvement effects of AI integration across the SDLC. The research uses Discrete Event Simulation (DES) with Monte Carlo sampling. The goal of the simulation is to produce a paired baseline for cycle times with no AI integration and with AI-enabled integration for statistical comparison. The output from the model is validated against published productivity improvements in the literature review. The results from the simulation and the statistical analysis are used to provide empirical evidence needed for validating the praxis hypotheses outlined in Chapter 1.

This chapter is organized as follows. Section 3.2 describes the dataset source, the survey filters applied to select the qualifying respondent population, and how the survey responses are prepared for use in the simulation. Section 3.3 documents the mapping of survey workflows to the SDLC phases used in the simulation model. Section 3.4 details the simulation model design and architecture, including the DES framework, the Monte Carlo sampling technique, and the log-normal parameters used to generate stochastic task times. Section 3.5 describes the AI Effect calculation. Section 3.6 documents the time allocation parameters adapted from COCOMO II. Section 3.7 describes the task time calculations for both baseline and AI-enabled phase time computations. Section 3.8 describes the sensitivity analysis process, using One-at-a-Time (OAT) method for testing model robustness by parameter variation. Sections 3.9 through 3.13 document the statistical methods applied to the simulation outputs. Finally, Section 3.14 presents the

simulation flowcharts that provide a complete step-by-step visual representation of the simulation architecture for reproducibility.

3.2 Dataset Description and Preparation

The dataset was obtained from publicly available data from the 2025 Stack Overflow Developer Survey conducted by Stack Overflow (Stack Overflow, 2025). Stack Overflow is one of the largest global communities online who conduct annual surveys to collect data from developers across different countries and diverse industries. The dataset was chosen based on the survey questions about AI tool usage in specific development workflows that map to SDLC phases, and responses on developer perceptions on the impact of AI agent use in software development.

The dataset is available for download from the Stack Overflow website. The download comes in a zipped folder containing two .csv files, one for the survey responses, and another one for the survey schema. A sample survey file in .pdf format is included in the download. The 2025 survey is grouped into five sets of questions as follows:

- Learning and Career
- Tech and Tech Culture
- Community
- AI
- Final Thoughts

To prepare the dataset, the data and schema files were first loaded into MS SQL Database tables. The table schema and structures were analyzed to identify questions and responses that would be relevant for the study. The study used Structured Query

Language (SQL) queries to filter the data and produce views that will be used for the final dataset.

The raw data for the 2025 survey contained 49,128 records. The first criterion filtered for professional software developers only and was used to exclude students, hobbyists and non-professional respondents from the final dataset. The second criterion filtered for respondents who were currently employed or working as an independent contractor or a freelancer or self-employed. This was to ensure that the dataset only includes responses from those who have current active development roles. The next criterion filtered for AI tool users to include only respondents who are using AI in at least one development workflow. The query excluded respondents who had missing or "NA" responses on workflow usage and AI agent impact. These values are used to calculate the AI Effect, which is one of the parameters in the simulation model. The goal was to select only developer respondents who are actively using AI tools/ agents in at least one workflow and provided responses for their perceived impact on development work.

The number of records after applying the initial inclusion/ exclusion criteria totaled 15,589. These are the responses from professional developers who are actively using AI tools in their SDLC workflows. Additional filter was added to filter for respondents who used AI Agents and provided responses on impact statements for their AI Agent use. After the additional filter was applied, the remaining number of respondents totaled n = 5,902.

Table 3- 1: Summary of Dataset Characteristics

Characteristic	Dataset Value
Datasource	Stack Overflow Developer Survey
Survey Year	2025
Raw Record Count	49,128
Final Record Count	5,902
Respondent Type	Professional software developers
Employment Status	Employed, Independent Contractor, Freelancer, Self-Employed
AI Tool Usage	Currently using AI in at least one development workflow
AI Agent Usage	Confirmed AI Agent users with valid impact reponses
Impact Statements Evaluated	Productivity, Automation, Code Quality, Time Reduction
SDLC Phases Mapped	Planning, Requirements, Design, Development, Testing, Deployment, Maintenance
SDLC Phases Simulated	6 Phases, excluding Maintenance

The specific survey questions that were used to provide the simulation parameters are as follows:

1. "Which of the following workflows or activities do you currently use AI tools for?" This is a multi-select question where respondents check all the workflows where they use AI. The study maps these workflows to SDLC phases. The percentage of responses for each workflow is calculated to

represent the Integration Level for that phase. The Integration Level equation takes the form:

Equation 3- 1: Integration Level

$$Integration\ Level(j) = \frac{\sum PHSE_{(i,j)}}{n}$$

where j = the SDLC phase, i = the respondent index, $PHSE_{(i,j)}$ = 1 if the respondent i uses AI in phase j and 0 otherwise, n = total respondents = 5,902.

2. "To what extent do you agree with the following statements regarding the impact of AI agents on your work as a developer?" This is presented in the survey as a matrix question where the respondents indicate their agreement with statements related to the impact of AI agents in their work. The response options from Strongly Agree to Strongly Disagree are mapped to a 5 to 1 numeric scale for the analysis, where Strongly Agree = 5 and Strongly Disagree = 1. The four AI agent impact statements included in the study are as follows:
    a. "AI agents have increased my productivity"
    b. "AI agents have helped me automate repetitive tasks"
    c. "AI agents have improved the quality of my code"
    d. "AI agents have reduced the time spent on specific development tasks"

For each phase, each of the impact statement's mean score is normalized by dividing by 5. The four normalized means are then averaged to calculate the value for the Impact Modifier. The Impact Modifier equation is as follows:

Equation 3- 2: Impact Modifier

$$Impact\ Modifier(j) = \left(\frac{1}{4}\right) * \sum \left[\frac{MeanScore_{(s,j)}}{5}\right]$$

Where j = SDLC phase, s = impact statement index 1 to 4, $MeanScore_{(s,j)}$ = mean numeric response score for statement s among respondents where $PHSE_{(i,j)}$ = 1, divided by 5 to normalize to [0,1]

3.3 SDLC Phase Mapping

The workflow responses were mapped to the following SDLC phases to standardize the analysis and assigned a binary vector value. To be consistent with software development standards, the mapping is based on the definitions from the SWEBOK Guide v4.0. For each of the workflows selected, the corresponding mapped SDLC phase is assigned a value = 1, otherwise the value is set to 0. Some workflows map to multiple SDLC phases as these workflows are considered activities for these phases.

Table 3- 2: Workflows and SDLC Phase Mapping

SDLC Phase	Survey Workflow
Planning	Project planning
	Predictive analytics
Requirements	Creating or maintaining documentation
	Learning about a codebase
	Search for answers
Design	Creating or maintaining documentation
	Generating content or synthetic data
	Learning new concepts or technologies
Development	Documenting code
	Generating content or synthetic data
	Writing code
	Debugging or fixing code
	Committing and reviewing code
Testing	Testing code
	Generating content or synthetic data
	Debugging or fixing code
Deployment	Deployment and monitoring
Maintenance	Creating or maintaining documentation
	Learning about a codebase
	Debugging or fixing code

3.4 Simulation Model Design and Architecture

The research utilizes Discrete Event Simulation (DES) and Monte Carlo technique to model tasks completing each phase of the SDLC. "DES allows representing and studying the behavior of physical systems, i.e., collections of entities interacting over time, whose states evolve in time as is the case in computer networks" (Ngangue Ndih & Cherkaoui, 2015, p. 488). In the research model, each discrete event is a task completing a phase in the SDLC. These tasks have stochastic service times and are tracked as they flow through processing points represented as phases of the SDLC.

The Monte Carlo technique is "a technique for selecting numbers randomly from a probability distribution for use in a trial (computer run) of a simulation model"

(Blackford, 2024). The random numbers generated from the log-normal probability distribution are used to represent the tasks that go through the entire development life cycle. Using log-normal distribution ensures that task times are greater than zero and represents the variability of tasks, which by nature have different complexities and cycle time completions. Based on the literature review, log-normal distribution is appropriate for software metrics data as it closely represents software task durations, which are right-skewed and long tailed in nature (Madachy, 2008). The log-normal parameters are derived from the target mean = 1.0 and the coefficient of variation (CV) = 0.20. The goal is for each simulated task to have a random multiplier that centers around 1.0, which would be the representation for tasks that complete as planned. The CV of 20% takes into account the variability of task cycle times. This value is derived from variability observed in the Integration and Test phase distributions in COCOMO II, where effort ranges from 16% to 34% across project sizes and scale factors (Boehm et al., 2000, Table 54). Selecting 20% represents projects that are well-managed while also providing enough variability without modelling extremely chaotic projects. The resulting random factors fall along the range of 0.6 – 1.5, with some outliers outside of the range. This simulates real-world scenarios where tasks run into unexpected roadblocks or complexities. The log-normal distribution equation takes the form:

Equation 3- 3: Log-normal Distribution Equation

$$f(x) = \left(\frac{1}{x\sigma\sqrt{2\pi}}\right) \exp\left\{-\frac{(ln(x) - \mu)^2}{2\sigma^2}\right\}, x > 0$$

where x = the random factor, μ = the mean of the underlying distribution in log-space, and (σ) = the standard deviation of the underlying distribution in log-space.

To be able to use the equation, the log-space variance and log-space mean need to be derived from the target mean = 1.0 and the CV = 0.20. The log-space variance equation converts the CV = 0.20 into the variance parameter required by the log-normal distribution and takes the form:

Equation 3- 4: Log-space Variance Equation

$$\sigma^2 = \ln(1 + CV^2)\ therefore, \sigma = \sqrt{\sigma^2}$$

The log-space mean equation converts the target output mean = 1.0 into the mean parameter for the log-normal distribution.

Equation 3- 5: Log-space Mean Equation

$$\mu = \ln(target\ mean) - \frac{\sigma^2}{2}$$

The inverse of the above-given equations are used to validate the outputs of the log-normal distribution.

Equation 3- 6: Log-normal Mean Equation

$$E[X] = \exp\left(\mu + \frac{\sigma^2}{2}\right)$$

Equation 3- 7: Log-normal Variance Equation

$$Var[X] = [\exp(\sigma^2) - 1] * \exp(2\mu + \sigma^2)$$

Equation 3- 8: Log-normal Coefficient of Variation Equation

$$CV = \sqrt{\exp(\sigma^2) - 1}$$

where E[X] = the expected mean value of the output, Var[X] = the variance of the output, and CV = the coefficient of variation, which is the standard deviation divided by the mean.

The C# simulation code uses the Box-Muller transform with a fixed seed set to 42 for reproducibility across all simulation runs. The Box-Muller transform method converts two uniform random numbers into a standard normal variate z that is then transformed into a log-normal variate (Box & Muller, 1958). The equation for Box-Muller transform is as follows:

Equation 3- 9: Box-Muller Transform Equation

$$z = \sqrt{-2\ln(u_1)} * \cos(2\pi u_2)$$

The log-normal transformation equation takes the form:

Equation 3- 10: Log-normal Transformation Equation

$$Random\ Factor = \exp(\mu + \sigma * z),$$

Where z = the standard normal variate generated from two uniform random numbers u_1 and u_2 via the Box-Muller transform, μ = the log-space mean, and σ = the log-space standard deviation. In Excel, the LOGNORM.INV(RAND(), μ, σ) function can be used for the log-normal transformation.

The model processes n = 100 simulated tasks per run and tracks each individual task as they flow through a sequence of discrete points that corresponds to each SDLC phase. For each phase, the model looks up the base cycle time allocations assigned to that specific phase. The time allocations are derived from COCOMO II. The model then generates a random multiplier from a log-normal distribution. The simulation uses a

paired design approach, applying the same random multiplier to both the baseline phase time and the AI-enabled phase time, following the Common Random Numbers (CRN) technique. This approach controls for inherent task difficulty and enables paired t-tests for statistical analysis. The baseline and AI-enabled phase times are each accumulated across all the SDLC phases to get the total cycle times. The simulation is implemented using C# to generate the random tasks, with a seed set to 42 for reproducibility, so that each run produces the same random factors for task times. The code aggregates the results, calculates task-level details and sensitivity analysis results. An output file with sample task times is generated from running the code. This was used to recreate a similar process in Excel for validation using the random factors as frozen values, with 100 tasks, that represents a simplified single simulation run for interactive testing.

Table 3- 3: Simulation Parameters

Parameter	Value	Unit
Base Cycle Time	40	hours
Number of Tasks per Run	100	tasks
Number of Simulation Runs	10,000	runs
Coefficient of Variation	0.2	—
Random Seed	42	—
SDLC Phases	6	phases

3.5 AI Effect Calculation

The AI Effect calculation is used to calculate the productivity improvement value for tasks that go through each phase of the SDLC. It is a combination of the two survey-derived components, the Integration Level, which indicates AI adoption per phase, and the Impact Modifier, which is calculated from the responses to the impact statements on AI Agent use in developer workflows. The AI Effect is based on the expected value

theory (Raiffa, 1968, pp. 8-9), where the Expected Value (EV) is the sum of each outcome multiplied by the probability of the outcome and takes the form:

Equation 3- 11: Expected Value (EV) Equation

$$E(X) = \sum P(X_i)X_i$$

In the simulation, a task with AI has its probability set to the Integration Level and its improvement magnitude set to the Impact Modifier, and the task with no AI has its probability set to 1 minus the Integration Level and zero improvement.

Equation 3- 12.a: AI Effect Equation

$$AI\ Effect = (Integration\ Level * Impact\ Modifier) + \big((1 - Integration\ Level) * 0\big)$$

In its simplified form, the equation then becomes:

Equation 3- 12.b: AI Effect (Simplified Form) Equation

$$AI\ Effect = (Integration\ Level * Impact\ Modifier)$$

Based on the values derived from the survey data, the primary driver for the AI Effect variation is the Integration Level or the adoption rate in the workflows related to the SDLC. The Development phase shows the highest adoption rate at 93.9%, while Deployment is showing the lowest adoption rate at 17.0%. The Impact Modifier is relatively consistent across the SDLC phases and ranges from 72.5% for Requirements to 75.1% for Planning.

Table 3- 4: AI Effect Values by Phase

SDLC Phase	Integration Level	Impact Modifier	AI Effect
Planning	36.50%	75.10%	0.27
Requirements	91.80%	72.50%	0.67
Design	85.90%	73.00%	0.63
Development	93.90%	72.60%	0.68
Testing	84.80%	73.30%	0.62
Deployment	17.00%	74.80%	0.13

3.6 Time Allocation

The Time Allocation parameters are adapted from COCOMO II. These allocations represent the proportion of the total effort spent on each SDLC phase for a typical software development project. COCOMO II is the industry-standard software cost estimation model that is based on data from real software development projects. However, COCOMO II only provides effort distributions for a four-phase Waterfall model and does not have a one-to-one mapping with the six-phase SDLC model for this research. Specifically, Plans & Requirements are combined into one phase in COCOMO II and assigned a 7% effort distribution with a range of 2-15%, Product Design at 17%, Programming at 52-64%, Integration & Test are also combined at 19-31%, and Transition at 12% with a range of 0-20%. These distributions total more than 100% due to Planning & Requirements and Transition being treated as supplemental effort percentages that are added on top of the core activities. To resolve the mismatches, the six COCOMO II sub-phases were included in the distribution and normalized to 100%. These were adjusted to split the combined phases to match the SDLC phases used in the research. The basis for these adjustments are the COCOMO II Waterfall phase distribution tables (Boehm et al., 2000, Table 45), the Waterfall activity distributions (Boehm et al., Tables 51-55), and the

COCOMO II MBASE/RUP activity distributions (Boehm et al., 2000, Table 57). The allocations follow and preserve COCOMO II's empirical ranking, with Development having the largest percentage, followed by Testing, Design, Requirements, Planning, and Deployment.

Table 3- 5: Time Allocation by SDLC Phase

Phase	SWEBOK KA	Time Allocation	AI Effect
Planning	KA9	8.70%	27.4115%
Requirements	KA1	13.00%	66.5550%
Design	KA3	16.30%	62.7070%
Development	KA4	32.60%	68.1714%
Testing	KA5	21.70%	62.1584%
Deployment	KA6	7.60%	12.7160%
(Maintenance excluded - no citable time source for time allocation)	KA7		

3.7 Task Time Calculations

The three core equations that drive the simulation are the Baseline Phase Time, AI-Enabled Phase Time, and Total Cycle Time. A Base Cycle Time is set to 40 hours and is used to represent a one-week sprint in a typical 40-hour work week. This is distributed across the phases using the COCOMO II time allocation percentages in 3.6. The Random Factor discussed in 3.4 is applied to the equation to simulate realistic variability. The Baseline Phase Time equation calculates the time it takes for a single task to complete one SDLC phase with no AI assistance. It serves as the control condition which will be used for comparison against AI-enabled times. The equation for Baseline Phase Time is as follows:

Equation 3- 13: Baseline Phase Time Equation

$$PhaseTime_{Baseline_{ij}} = BaseCycleTime * TimeAllocation_j * RandomFactor_{ij}$$

Where, i = Task index, j = SDLC phase index, BaseCycleTime = 40 hours, TimeAllocation_j = COCOMO II Time Allocation percentage for SDLC phase, RandomFactor_{ij} = log-normal multiplier for Task i in Phase j from the RandomFactor equation in 3.4.

The AI-Enabled Phase Time equation calculates the length of time it takes for an AI-assisted task to complete one SDLC phase and takes the form:

Equation 3- 14: AI-Enabled Phase Time Equation

$$PhaseTime_{AI-Enabled_{ij}} = BaseCycleTime * TimeAllocation_j * (1 - AI\ Effect_j) * RandomFactor_{ij}$$

Where, i = Task index, j = SDLC phase index, BaseCycleTime = 40 hours, TimeAllocation_j = COCOMO II Time Allocation percentage for SDLC phase, $AI\ Effect_j$ = AI Effect value from AI Effect calculation per phase in 3.5, and RandomFactor_{ij} = log-normal multiplier for Task i in Phase j from the RandomFactor equation in 3.4. The equation is identical to the Baseline Phase Time except for an additional term, 1 –AI Effect_j. This term converts the proportion of time saved into the proportion of time remaining. As an example, for Development phase, the AI Effect = 0.68, then (1 – 0.68) = 0.32, which means that the AI-enabled task would take 32% of the baseline phase time. If a baseline task of 13.69 hours is multiplied by 68%, it would yield 9.31 hours, which is representative of the time saved. To calculate for the actual AI-enabled task time, the baseline phase time is multiplied by (1 – 0.68), which would result to 4.38 hours. The sum of the resulting time saved and AI-enabled task time equals that of the baseline phase time.

The Total Cycle Time aggregates the phase times across all the SDLC phases to produce the total end-to-end cycle time for each task. The equation for Total Cycle Time is as follows:

Equation 3- 15: Total Cycle Time Equation

$$TotalCycleTime_i = \sum PhaseTime_{ij}$$

Where, j = ranges over all SDLC phases, $PhaseTime_{ij}$ = from either Baseline Phase Time or AI-Enabled Phase Time. The total cycle times for the tasks are calculated separately for the baseline and the AI-enabled tasks. This produces a matched pair of cycle times for each of the 100 tasks per simulation run. These matched pairs are used as the raw data for the subsequent statistical analyses.

Table 3- 6.a: Excel Worksheet Example with Task 1 Cycle Time

Parameters:		
Base Cycle Time	40	hours
Random Factor	1.05	

SDLC Phase	Time Allocation	Base Hours	AI Effect	(1 - AI Effect)	Baseline Time (hrs)	AI-Enabled Time (hrs)	Time Saved (hrs)
Planning	8.70%	3.48	27.4115%	0.7259	3.65	2.65	1.00
Requirements	13.00%	5.20	66.5550%	0.3345	5.46	1.83	3.63
Design	16.30%	6.52	62.7070%	0.3729	6.85	2.55	4.29
Development	32.60%	13.04	68.1714%	0.3183	13.69	4.36	9.33
Testing	21.70%	8.68	62.1584%	0.3784	9.11	3.45	5.67
Deployment	7.60%	3.04	12.7160%	0.8728	3.19	2.79	0.41
Total	**100%**	**40**			**42**	**18**	**24**

Table 3- 6.b: Breakdown of AI Effect Calculation in Worksheet Example

AI Effect Calculated from 2025 Stack Overflow Developer Survey:	
Planning:	36.5% integration x 75.1% impact = 27.4115%
Requirements:	91.8% integration x 72.5% impact = 66.5550%
Design:	85.9% integration x 73.0% impact = 62.7070%
Development:	93.9% integration x 72.6% impact = 68.1714%
Testing:	84.8% integration x 73.3% impact = 62.1584%
Deployment:	17.0% integration x 74.8% impact = 12.7160%

3.8 Sensitivity Analysis

Sensitivity analysis is "often defined as a local measure of the effect of a given input on a given output" (Saltelli et al., 2004, p. 42). It is conducted to test the robustness of a model and involves varying the input parameters and observing the subsequent effects on the outputs. The research uses One-at-a-time (OAT) analysis. Global sensitivity methods offer broader coverage of parameter interactions (Saltelli et al., 2004). However, this study examines only three parameters with clearly established values. OAT analysis was therefore sufficient as it is the simplest approach for models with limited parameters (Hamby, 1994). For each parameter that is varied, the other parameters are held at their base values and the same master random seed = 42 is used to ensure the results are due to the parameter change and not to some other random variation. For each of the variations, the overall cycle-time improvement, the Cohen's d, the t-statistic, and the 95% confidence interval are output to a csv file. The parameters used in the sensitivity analysis are AI Effect, Coefficient of Variation (CV), and Base Cycle Time. To test the model's sensitivity to the overall impact of AI integration, the combined AI Effect is scaled. AI Effect is the product of the Integration Level and Impact Modifier, scaling the combined AI Effect is equivalent to individually scaling either of

the components while still holding the other constant. The AI Effect for each phase is scaled by the same factor so the ratios between phases remain the same.

The AI Effect scale and the Base Cycle Time parameters in the sensitivity analysis are uniform scaling factors and affect both the mean paired difference and standard deviation proportionally. These cancel each other out in the calculations for Cohen's d and t-statistic and the results of the analysis show these as unchanged. Since the CV parameter alters the underlying stochastic distribution, variation for CV also showed varied results for the corresponding Cohen's d and t-statistic calculation.

Table 3- 7: Sensitivity Analysis Parameters and Variations

Parameter	Variation	MeanImprovement_Pct	CI_Lower_Hrs	CI_Upper_Hrs	CohenD	TStatistic
CV	0.1	57.9958	22.943	23.4075	19.9052	199.0515
CV	0.15	57.9962	22.827	23.5237	13.2718	132.7182
CV	0.2	57.9965	22.711	23.6399	9.9562	99.5619
CV	0.25	57.9968	22.5949	23.7559	7.9677	79.677
CV	0.3	57.997	22.4788	23.8719	6.6429	66.4286
BaseCycleTime	20	57.9965	11.3555	11.8199	9.9562	99.5619
BaseCycleTime	30	57.9965	17.0332	17.7299	9.9562	99.5619
BaseCycleTime	40	57.9965	22.711	23.6399	9.9562	99.5619
BaseCycleTime	60	57.9965	34.0665	35.4598	9.9562	99.5619
BaseCycleTime	80	57.9965	45.422	47.2798	9.9562	99.5619
AIEffectScale	0.8	46.3972	18.1688	18.9119	9.9562	99.5619
AIEffectScale	0.9	52.1969	20.4399	21.2759	9.9562	99.5619
AIEffectScale	1	57.9965	22.711	23.6399	9.9562	99.5619
AIEffectScale	1.1	63.7962	24.9821	26.0039	9.9562	99.5619
AIEffectScale	1.2	69.5958	27.2532	28.3679	9.9562	99.5619

3.9 Normality Test

A normality assessment was conducted prior to conducting the paired t-test. The paired t-test assumes that the paired differences are approximately normally distributed. Violation of this assumption can produce unreliable t-statistics and p-values, which leads to incorrect conclusions about statistical significance. The normality assessment was

performed using the Ryan-Joiner normality test in Minitab with a significance level of $\alpha = 0.05$. The Ryan-Joiner test is Minitab's equivalent of the Shapiro-Wilk test and measures the correlation between the data and the expected normal quantiles for assessing normality. The test was performed on all four key output variables: MeanBaseline, MeanAIEnabled, MeanDifference, and PercentImprovement. These variables are each derived from the n = 10,000 simulation runs.

MeanDifference is the direct input variable to the paired t-test in Eq. 3-16 and is the primary variable of interest for the normality assessment. The results for the normality assessment conducted on the MeanDifference variable showed that it passed with RJ = 1.000 and p-value = 0.087, which is above the 0.05 threshold and confirmed that the normality assumption for the paired t-test is satisfied. The other remaining variables also formally passed their respective normality assessments. The MeanBaseline returned RJ = 1.000 and p-value = 0.085. The MeanAIEnabled test results showed RJ = 1.000 and p-value > 0.100, while the PercentImprovement test results showed RJ = 1.000 and p-value > 0.100.

Visual inspection of the probability plots supported the results, with the data points tracking closely along the fitted reference line without S-curves, bending, or outliers at the tails. Figures 3-1[a-d] present the probability plots for all the four variables.

The normality of the run-level means is further supported by the Central Limit Theorem, which states that the distribution of sample means converges to normality as the number of observations increases regardless of the shape of the underlying distribution. Each run-level value is derived from the average of 100 task-level

observations drawn from a log-normal distribution. The sample size of n = 10,000 simulation runs is sufficiently large enough for the CLT to apply.

The normality assumption for the paired t-test is satisfied based on these three grounds: all four variables passed the normality thresholds for the Ryan-Joiner test, visual probability plot inspection confirmed linearity across all the four variables, and the Central Limit Theorem theoretical application support for normality.

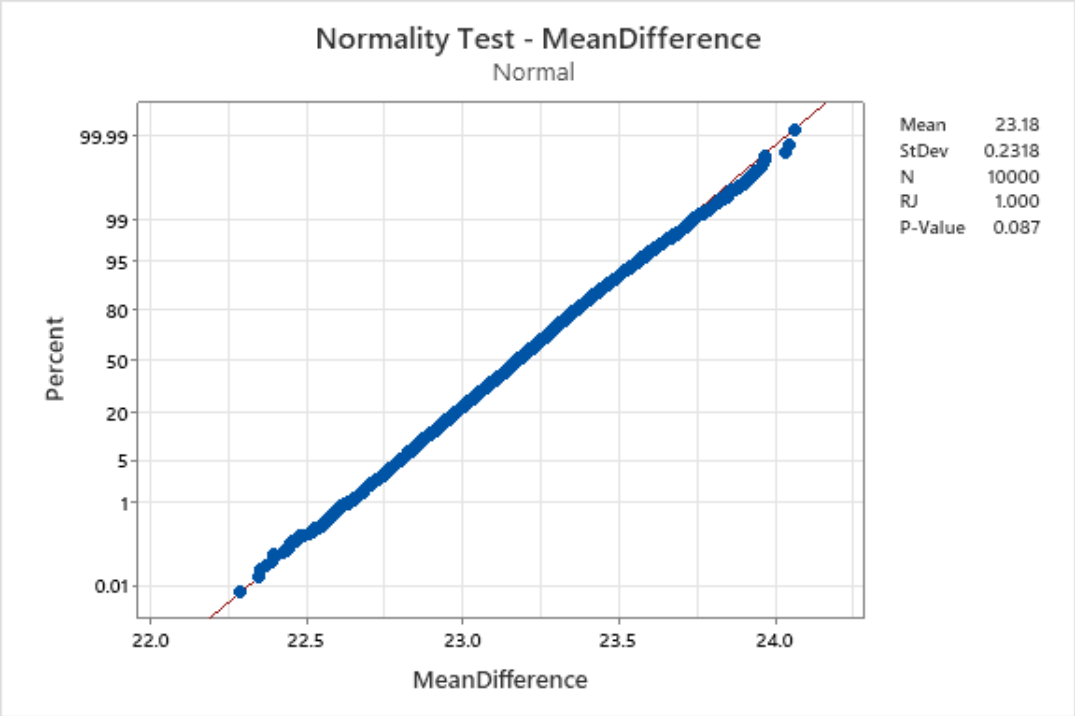


Figure 3-1a: Normality Test – Mean Difference Probability Plot

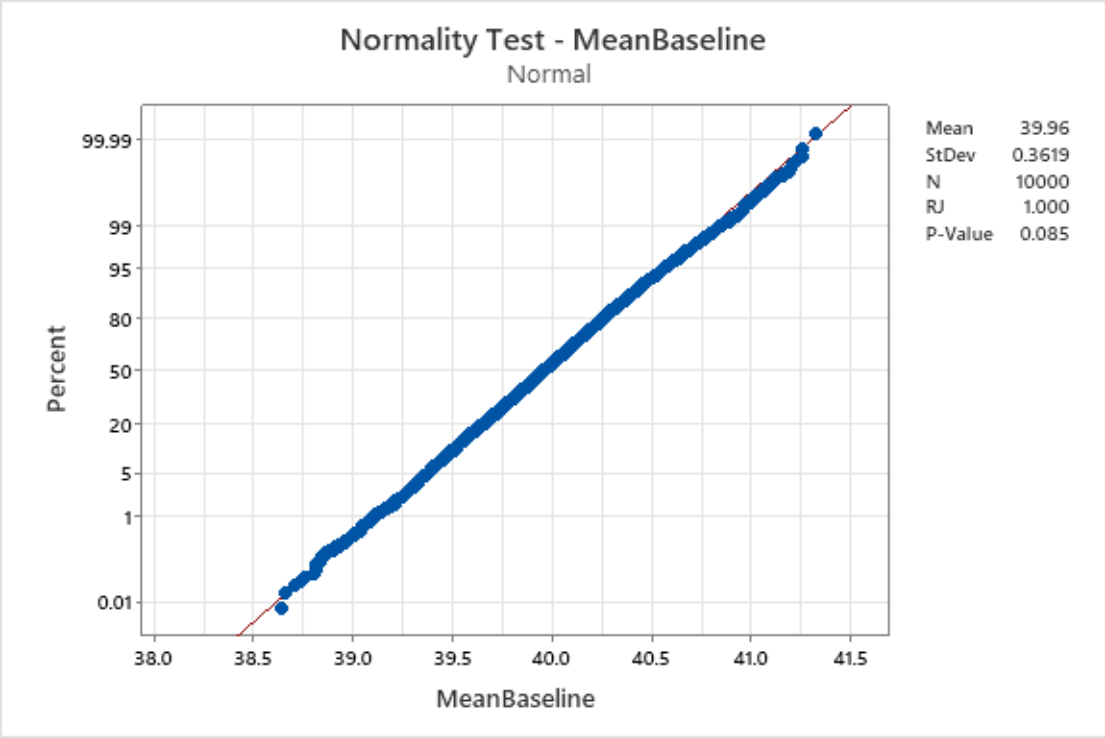


Figure 3-1b: Normality Test – Mean Baseline Probability Plot

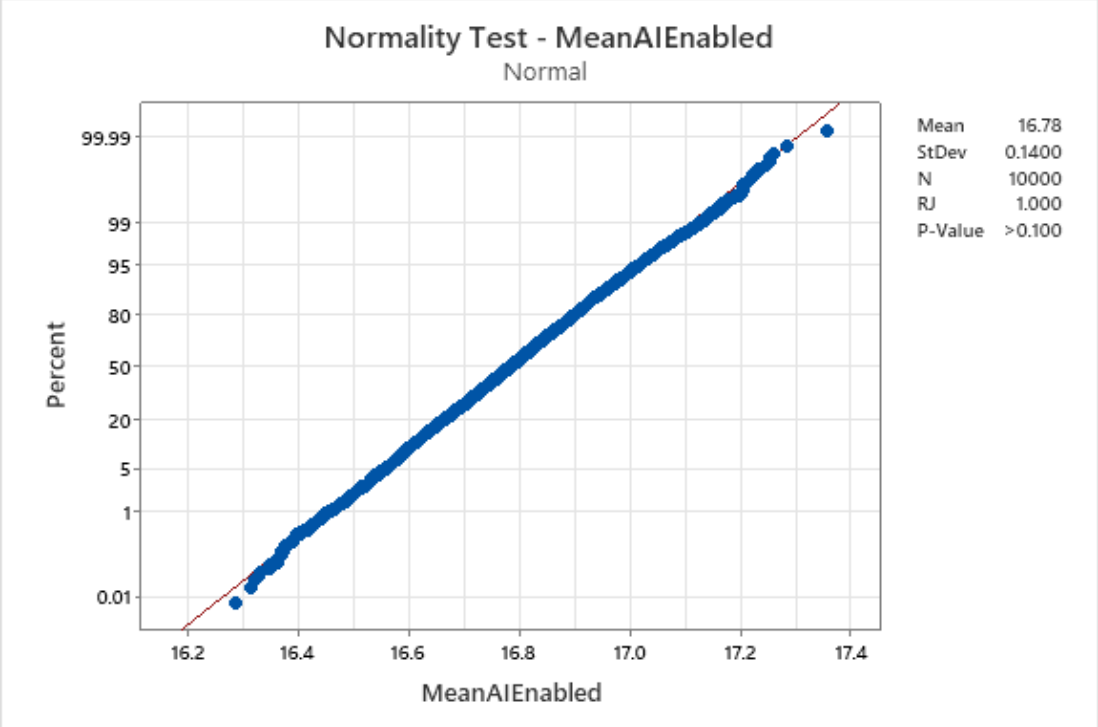


Figure 3-1c: Normality Test – Mean AI-Enabled Probability Plot

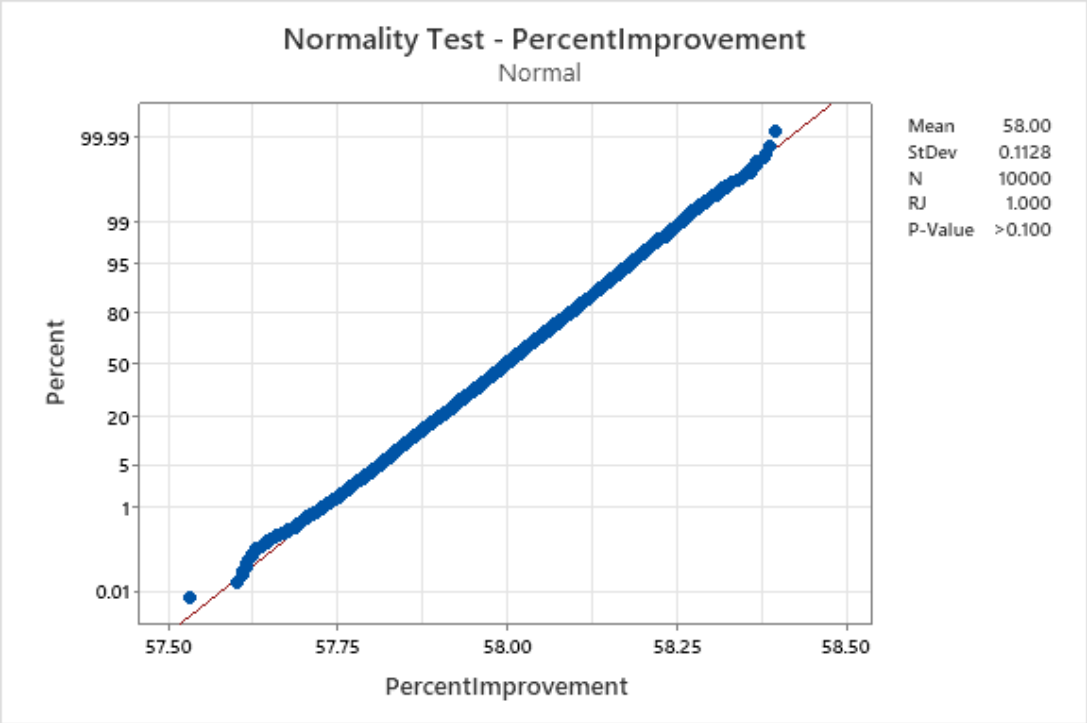


Figure 3-1d: Normality Test – Percent Improvement Probability Plot

3.10 Paired t-Test

To validate the research hypotheses H1 and H3, a paired t-test is conducted to test whether the mean difference between the baseline and AI-enabled cycle times is significantly greater than zero. For H1, this is applied to the predicted 20% overall cycle time reduction. For H3, the phase-level cycle time improvements are compared against the predicted thresholds of less than 30% for phases with lower adoption and greater than 50% for phases with higher adoption. The simulation uses a Common Random Number (CRN) design and each simulated task is processed under both the Baseline Phase Time and AI-Enabled Time using the same random numbers, which result in dependent observations. This design is chosen to eliminate variability between tasks. The hypotheses for the paired t-test are as follows:

H_0 (Null): There is no difference between the mean cycle times for baseline and AI-enabled tasks.

H_1 (Alternative): Baseline tasks take significantly longer to complete than AI-enabled tasks

The paired t-statistic equation takes the form:

Equation 3- 16: Paired T-Statistic Equation

$$t = \frac{\bar{d} - 0}{\frac{\hat{\sigma}}{\sqrt{n}}}$$

Where, $\bar{d}$ = the mean of paired differences between Baseline Phase Time$_i$ and AI-Enabled Phase Time$_i$, calculated across all the 100 tasks in the simulation run; $\hat{\sigma}$ = the standard deviation of the paired differences; and n = number of tasks per simulation run. For each of the simulation runs, the process follows several steps. The first step is to calculate the

paired difference for each task by subtracting the AI-Enabled Cycle Time from the Baseline Cycle Time. This isolates the AI Effect as both baseline and AI-enabled tasks share the same random numbers. The equation used is as follows:

Equation 3- 17: Paired Difference Equation

$$d_i = Baseline\ Cycle\ Time_i - AIEnabled\ Cycle\ Time_i$$

The second step calculates the mean of the paired differences for the 100 tasks. This represents the average hours saved for each task in the simulation run, and follows the equation:

Equation 3- 18: Mean Paired Difference Equation

$$\bar{d} = \left(\frac{l}{n}\right) * \sum d_i$$

The third step calculates the standard deviation of the paired differences. There is a high correlation between the baseline and AI-enabled task completion times due to the model's CRN paired design. This step uses the equation:

Equation 3- 19: Standard Deviation of Paired Differences Equation

$$\hat{\sigma} = \frac{\sqrt{\sum(d_i - \bar{d})^2}}{n-1}$$

The fourth step calculates the standard error and takes the form:

Equation 3- 20: Standard Error Equation

$$SE = \frac{\hat{\sigma}}{\sqrt{n}}$$

The final step calculates the t-statistic, with degrees of freedom df = n -1 = 99. This value is compared to the critical value from the t-distribution table for a one-tailed test at $a = 0.05$. The threshold t_critical is 1.66 for df = 99 for one-tailed test (NIST/SEMATECH e-Handbook of Statistical Methods, n.d.). If the t-statistic exceeds this threshold, then the null hypothesis is rejected with p-value < 0.05, which indicates that AI integration in the SDLC significantly reduces development cycle times.

3.11 Effect Size

The Effect Size is measured by Cohen's d. Cohen's d quantifies the magnitude of the difference between the baseline and AI-enabled cycle times regardless of the sample size. While the t-test can indicate that the difference is statistically significant, having a larger sample size results in a larger t-statistic and a smaller p-value, and can produce statistical significance even though the actual difference is small. Measuring the effect size provides insight on whether the difference is large enough to be meaningful. Cohen's guidelines are as follows: d = 0.2 (Small), d = 0.5 (Medium), d = 0.8 (Large) (Cohen, 1988). For paired designs, Cohen's d is calculated as:

Equation 3- 21: Cohen's d

$$d = \frac{\bar{d}}{\hat{\sigma}}$$

Where, $\bar{d}$ = the mean of the paired Baseline and AI-Enabled Cycle Time differences, and $\hat{\sigma}$ = the standard deviation of the paired difference, with both values calculated within each simulation run. A Cohen's d exceeding 0.8 indicates that the AI-enabled cycle time is more than one standard deviation below the baseline cycle time, which means the AI-

enabled tasks took less time to complete compared to the tasks without AI. The resulting value for Cohen's d = 9.96, which exceeds the "Large" threshold by 12.5 times.

3.12 Confidence Interval

The 95% Confidence Interval is calculated to determine the range wherein the true mean improvement is expected to fall and uses the following equation:

Equation 3- 22: Confidence Interval Equation

$$CI = \bar{d} \pm t_{critical} \frac{\hat{\sigma}}{\sqrt{n}}$$

Where, $t_{critical}$ = 1.984 (two-tailed) for df = 99 at $\alpha = 0.05$ (NIST/SEMATECH e-Handbook of Statistical Methods, n.d.). The two-tailed t_critical value is used to provide a symmetric bound for the estimate.

3.13 Aggregation Across Simulation Runs

The simulation is run 10,000 times in the C# code with 100 tasks per run. Each run uses random sampling and calculates the statistical methods outlined in this Chapter: Paired t-test, Cohen's d, and 95% Confidence Interval. This produces 10,000 corresponding values for each statistic. Consistent findings across all simulation runs indicate a stable pattern, not a result of one random configuration. This follows the recommendation from Banks et al. (2010) that sufficient replications are needed for replication stability. The aggregation allows reporting of worst-case and best-case bounds, where if the lower bound t-statistic across all runs exceeds the critical value for $p < 0.001$ and the lowest Cohen's d still classifies as a large effect, then the conclusion still holds even in the worst-case scenario.

3.14 Simulation Flowcharts

This section contains the series of flowcharts that provide a complete visual representation of the simulation architecture for reproducibility. Each process in the flowchart that contains calculations is mapped to the Equations in the list of equations table.

Figure 3-2: Parameter Derivation flowchart shows the derivation of simulation parameters from the 2025 Stack Overflow Developer Survey. It documents the SQL filtering, and SDLC mapping. It also documents how the Integration Level and Impact Modifier values are derived, and the AI Effect calculation that is based on these variables.

Figure 3-3: Main Simulation flowchart shows the main simulation execution flow, which includes initialization of parameters, calling the Simulation Engine, aggregating the statistical analysis outputs, and exporting the results to output files. The Main Simulation uses the parameters that are output in the parameter derivation flowchart to call on the Simulation Engine and process the main simulation. It then calls the Sensitivity Analysis method separately to process the simulation for recalibrated parameters.

Figure 3-4: Simulation Engine flowchart shows the core simulation execution, which includes the run loop, the task loop, and the phase loop. It takes the simulation parameters as inputs and outputs the RunResults list and the SingleRunTasks. This engine is designed for re-use by both the Main Simulation and Sensitivity Analysis methods for consistency between the base simulation results and the sensitivity analysis results.

Figure 3-5: Sensitivity Analysis flowchart shows how the parameters are varied one at a time and passed on to the Simulation Engine. The results are then aggregated across all variations and sent as output to the sensitivity analysis file.

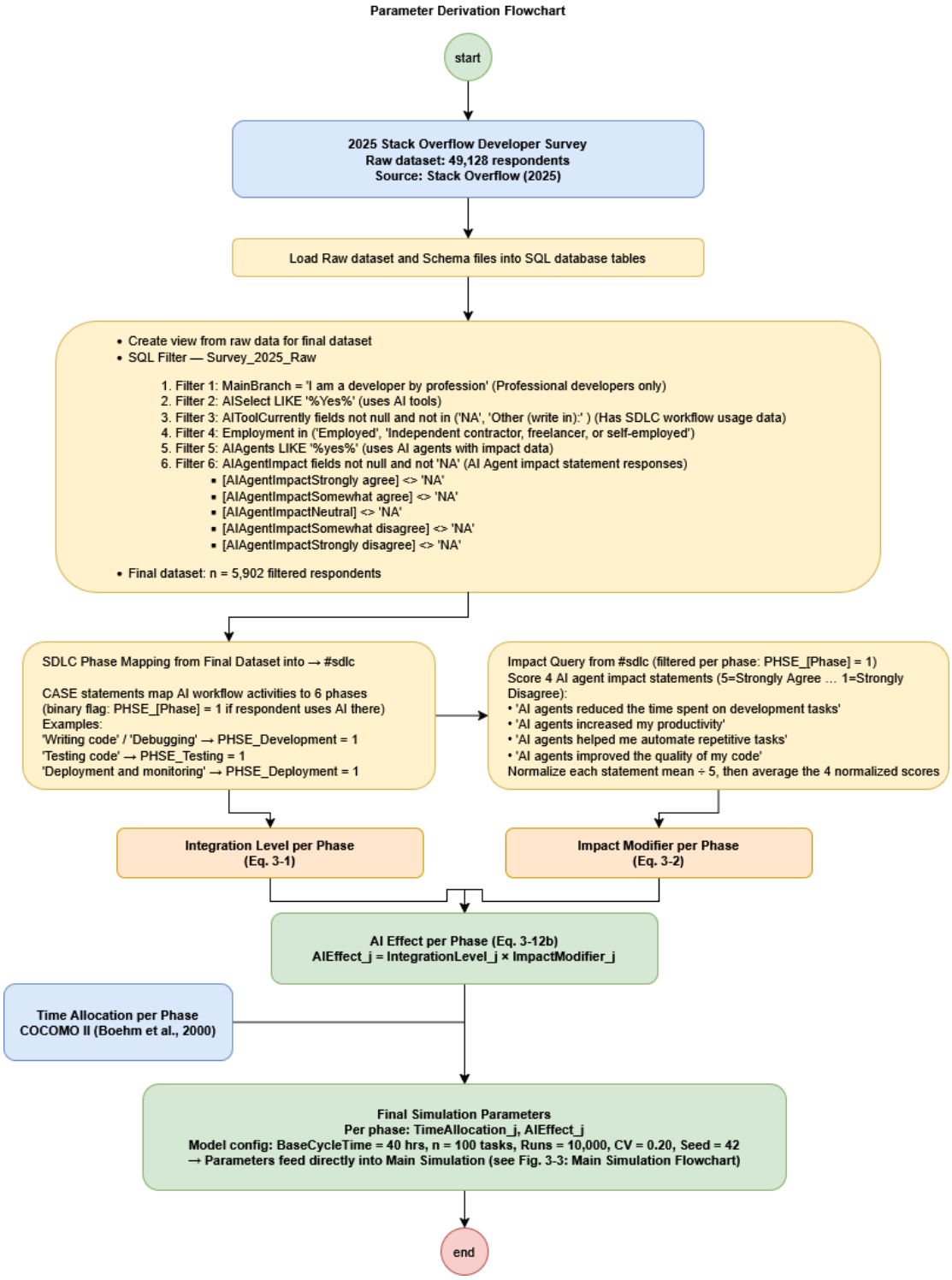


Figure 3-2. Parameter Derivation Flowchart

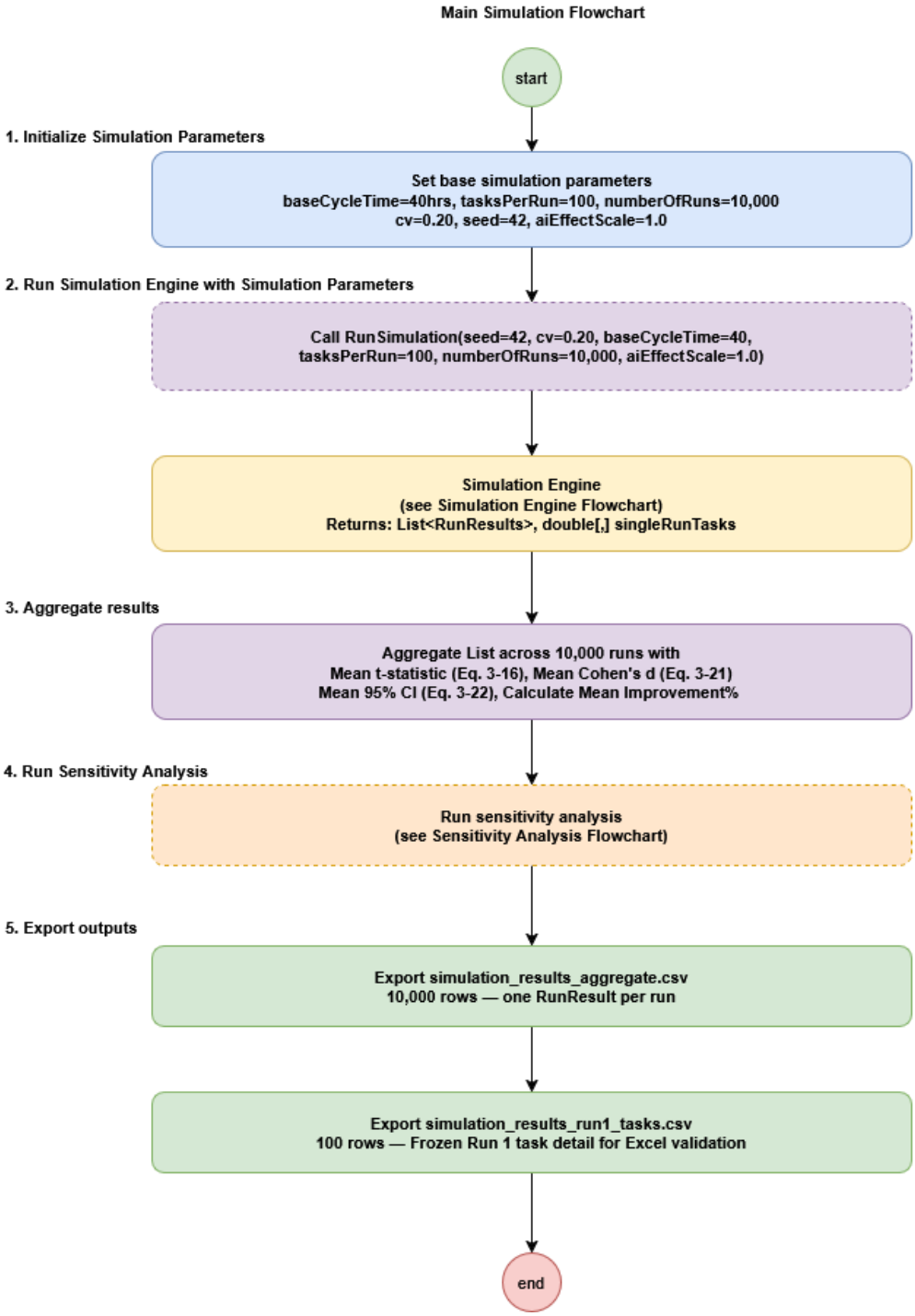


Figure 3-3. Main Simulation Flowchart

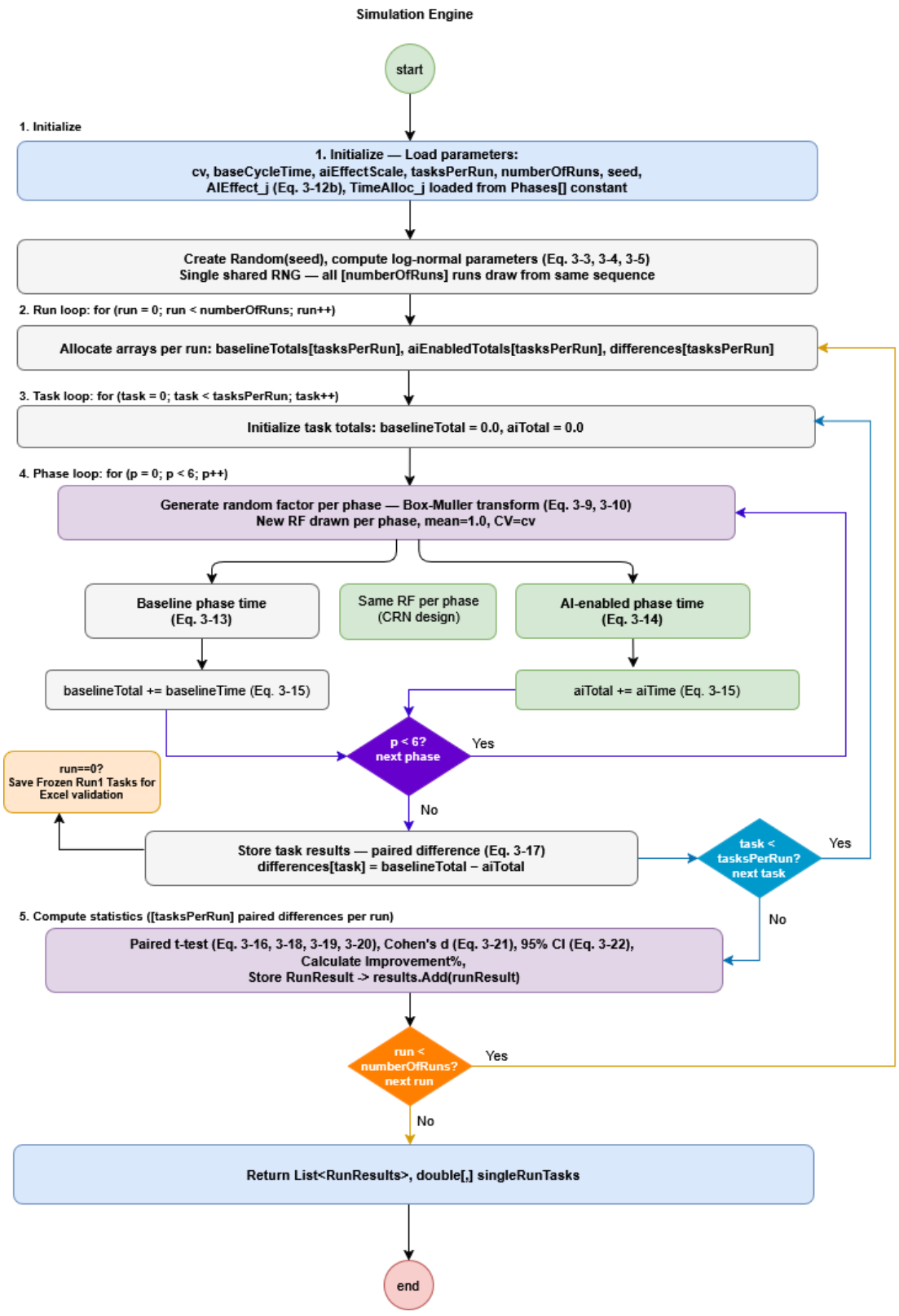


Figure 3-4. Simulation Engine Flowchart

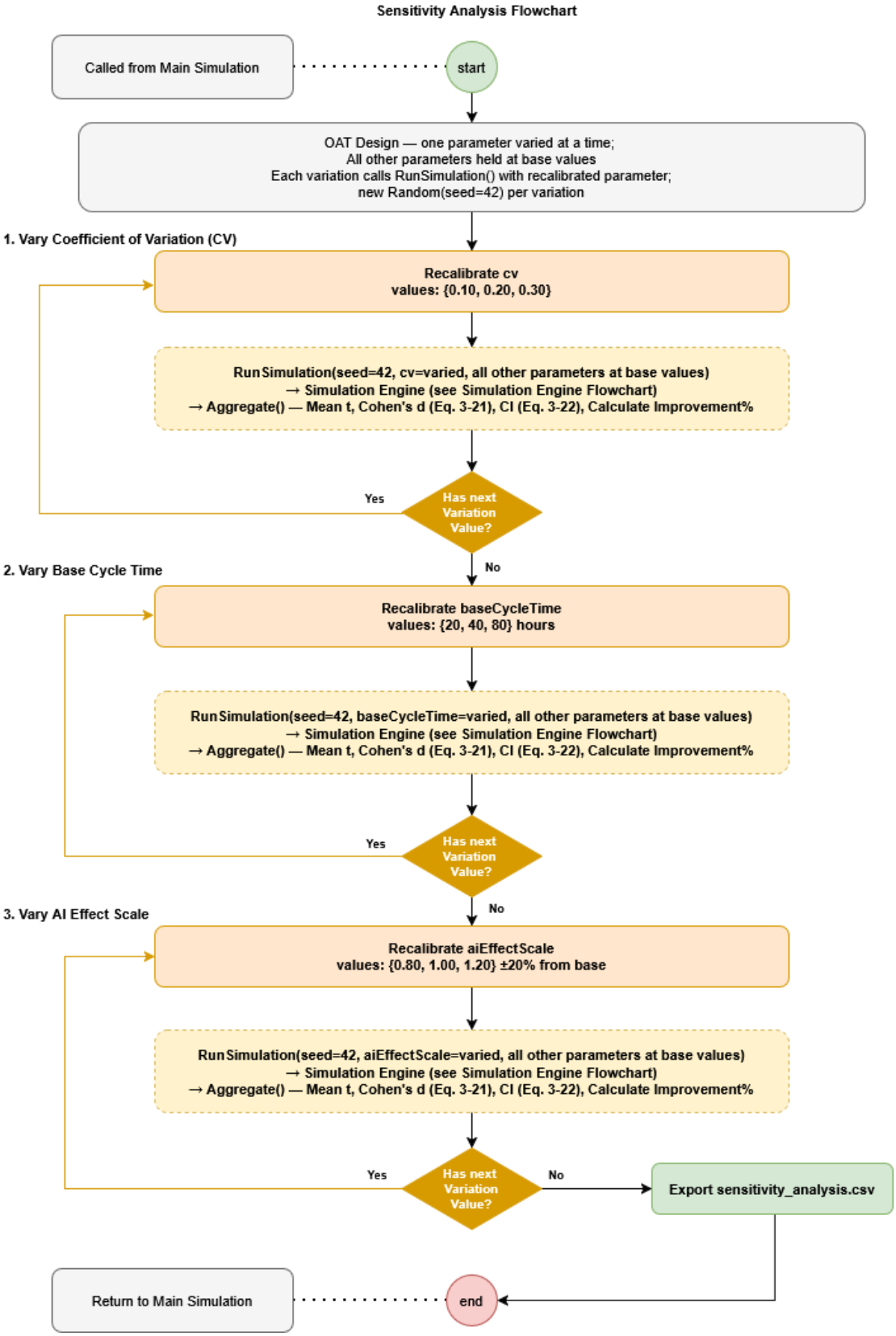


Figure 3-5. Sensitivity Analysis Flowchart

Chapter 4—Results

4.1 Introduction

This chapter presents the results from the DES methodology described in Chapter 3. The results are used to test the hypotheses established in Chapter 1. The methodology consisted of 1.) simulation model design as discussed in Section 3.4, 2.) AI Effect parameterization from the 2025 Stack Overflow survey data that was presented in Section 3.5, 3.) paired comparison design with Common Random Numbers (CRN) as described in Section 3.4, and 4.) statistical analysis for each of the research hypotheses. The simulation model was configured based on the control parameters listed in Table 4- 1. These parameters were established in the methodology in Chapter 3 and were used across all simulation runs, except for the sensitivity analysis where the parameters were varied.

Table 4- 1: Simulation Control Parameters

Parameter	Value	Unit
Base Cycle Time	40	hours
Number of Tasks per Run	100	tasks
Number of Simulation Runs	10,000	runs
Total Task-Level Observations	1,000,000	observations
Coefficient of Variation	0.2	—
Random Seed	42	—
SDLC Phases	6	phases

The simulation generated 1,000,000 task-level observations from 10,000 runs with 100 tasks per run. Each task was processed through six SDLC phases in both baseline and AI-enabled conditions using the same random factors from the CRN design described in Section 3.4. The simulation only considered SDLC phases from Planning all through Deployment. The Maintenance phase was excluded from the simulation as this phase did not have a citable time source for time allocation.

The results of the simulation produced matched-pairs of baseline and AI-enabled cycle times that are then used in the paired t-test analysis. The C# simulation code generated three output files: an aggregate .csv file with the summary statistics across all 10,000 runs, a task-level .csv with the 100 paired cycle times for Run 1, which was used for verification against the Excel-based simulation, and a sensitivity analysis .csv file with the results for the sensitivity analysis tests. The mean baseline cycle time was 39.96 hours, and the mean AI-enabled cycle time was 16.78 hours. The overall reduction in cycle time was 58%, which is 23.18 hours in cycle time savings per task.

Sections 4.2 – 4.4 present the results of the research hypotheses. Each section discusses the simulation results and provides the statistical analysis validating the hypotheses. Section 4.5 presents the sensitivity analysis results. Finally, Section 4.6 provides a summary and the conclusions based on the results.

4.2 H1 Results

4.2.1 H1 Hypothesis Statement

The research hypothesis H1 states: AI integration improves overall developer effectiveness by 20%

The paired t-test was used to validate H1. The hypotheses for the paired t-test are as follows:

- H_0 (Null): There is no difference between the mean cycle times for baseline and AI-enabled tasks.

  H_0 (Null Hypothesis): $\mu_{Baseline} - \mu_{AI-Enabled} = 0$
- H_1 (Alternative): Baseline tasks take significantly longer to complete than AI-enabled tasks

H_1 (Alternative Hypothesis): $\mu_{Baseline} - \mu_{AI-Enabled} > 0$

4.2.2 H1 Statistical Analysis

The simulation was run 10,000 times with 100 tasks per run following the process described in Chapter 3. Across all runs, the mean baseline cycle time was 39.96 hours and the mean AI-enabled cycle time was 16.78 hours. The mean paired difference was 23.18 hours, demonstrating an overall 58% reduction in cycle time. This observed value exceeds the 20% improvement predicted for H1.

The paired t-test was conducted with df = 99 and the t-statistic for each run was compared against the t-critical value. As described in Section 3.9, the threshold for t-critical = 1.66 for df = 99, for a one-tailed test at α = 0.05. The mean t-statistic was 99.56, which also exceeds the critical value of 1.66 at α = 0.05. This confirms that the observed values are consistent and not dependent on any specific random configuration.

The mean Cohen's d was 9.96, exceeding the threshold of 0.8 for "Large" effect size by 12.5 times. This indicates that the baseline vs. AI-enabled distributions are approximately 10 standard deviations apart and not overlapping. The high Cohen's d reflects the controlled simulation environment where the AI Effect was isolated. In real-world scenarios, there will most likely be additional variance coming from different contexts and the Cohen's d will likely still show a large effect but not as high as 12.5 times the "Large" threshold.

The 95% Confidence Interval hours were calculated as described in Section 3.11 in Chapter 3, with t_critical = 1.984, two-tailed for df = 99 at α = 0.05. The results showed [22.71, 23.64] hours, approximately $\pm$ 0.47 hours from the Mean Cycle Time difference of 23.18 hours. The narrow interval indicates simulation precision.

4.2.3 H1 Summary

The t-statistic of 99.56 far exceeded the t-critical threshold of 1.66, with p-value < 0.001 across all 10,000 simulation runs, with an improvement range of 57.53% - 58.40%. This rejects the null hypothesis H_0: $\mu_{Baseline} - \mu_{AI-Enabled} = 0$. The simulation results confirmed that AI integration improves overall software development effectiveness, with an overall improvement of 58%, which exceeded the predicted 20% improvement stated in H1.

Table 4- 2: Paired t-test Results

Statistic	Results	Interpretation
Mean Baseline Cycle Time	39.96 hrs	—
Mean AI-Enabled Cycle Time	16.78 hrs	—
Mean Paired Difference	23.18 hrs	—
Overall Improvement	58%	Exceeds H1 threshold of 20%
Improvement Range (10,000 runs)	57.53% – 58.40%	—
Mean t-statistic	99.56	t-critical = 1.66, df = 99, α = 0.05
p-value	< 0.001	All 10,000 runs
Mean Cohen's d	9.96	Exceeds 'Large' threshold = 0.8 by 12.5x
95% Confidence Interval	[22.71, 23.64] hrs	± 0.47 hrs

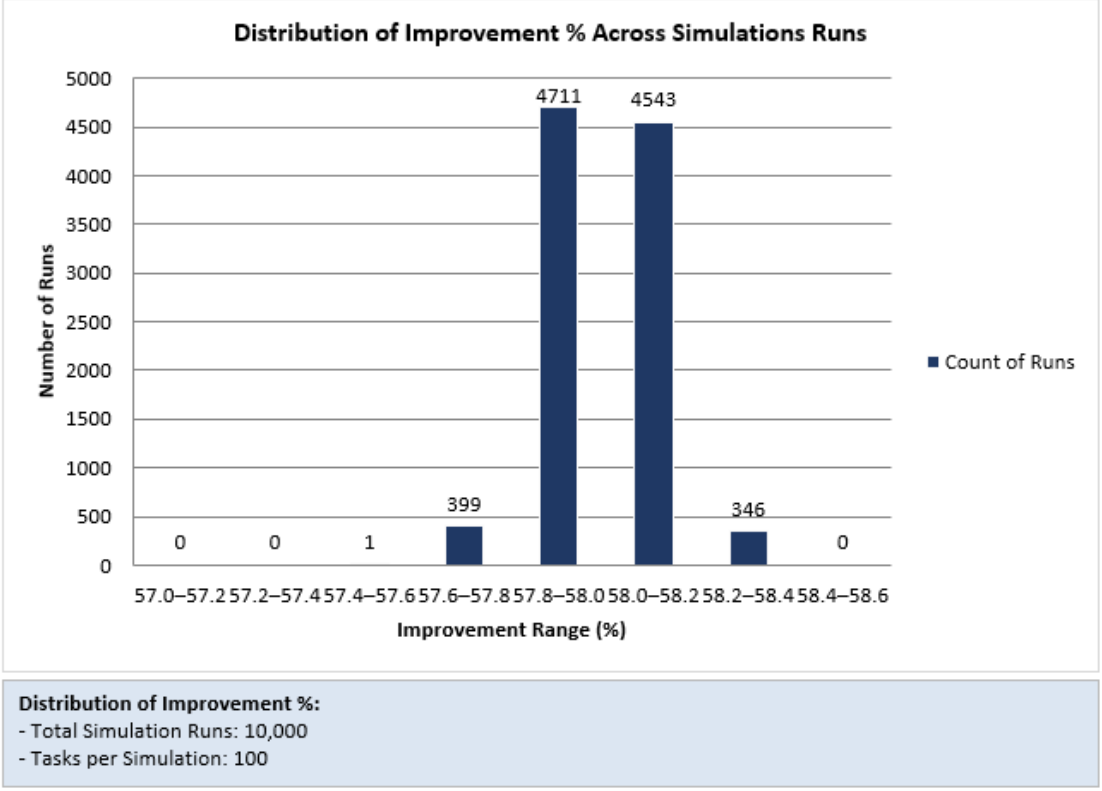
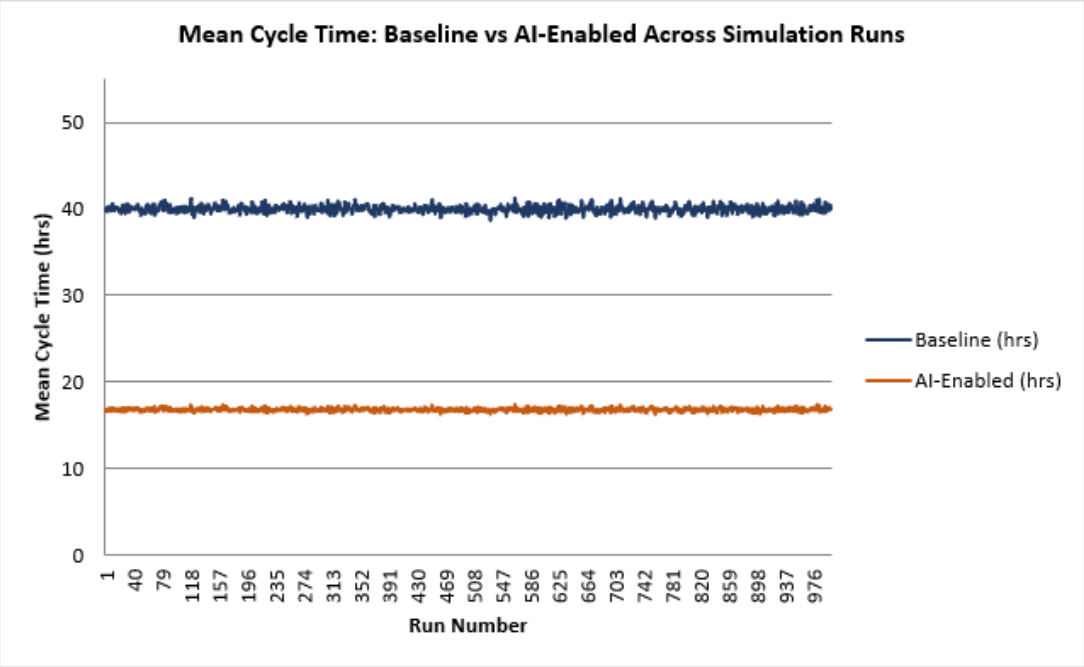


Figure 4-1. Simulation Distribution Histogram



DES Baseline vs. AI-Enabled Cycle Time:
- Total Simulation Runs: 10,000
- Tasks per Simulation: 100

Figure 4-2. Baseline vs. AI-Enabled Cycle Time Hours

4.3 H2 Results

4.3.1 H2 Hypothesis Statement

The research hypothesis H2 states: AI integration demonstrates the highest improvements in the Development phase, with a 25% reduction in cycle time.

The hypothesis statements for H2 are as follows:

- H_0 (Null Hypothesis): AI integration does not demonstrate higher improvements in the Development phase, and Development does not contribute more significantly to the overall cycle time.

  H_0: $AI_Effect_{Development} \leq AI_Effect_j \; for \; at \; least \; one \; j \neq Development$

- H_1 (Alternative Hypothesis): AI integration demonstrates the highest improvements compared to all other phases in the SDLC.

  H_1: $AI_Effect_{Development} > AI_Effect_j \; for \; all \; j \neq Development$

4.3.2 H2 Statistical Analysis

The AI Effect for each SDLC phase is the product of the Integration Level and the Impact Modifier as described in Section 3.5 in Chapter 3. The Integration Level represents the responses from respondents who reported using AI technology in the workflows that were mapped to the SDLC phases. The Impact Modifier represents the calculated perceived improvement benefits from their impact statements on AI Agent use. The simulation results showed that Development has the highest AI Effect at 0.682, and the cycle time improvement produced was 68.2%, with 8.89 hours saved per task. The observed value of 68.2% exceeded the predicted value of 25% in H2. The hours saved per task are the result of having both the highest AI Effect and the largest time allocation,

which was at 32.6% from COCOMO II. The Requirements phase ranked second in AI Effect with a 66.6% improvement, while Testing ranked second in hours saved at 5.40 hours due to a larger time allocation of 21.7%.

Table 4- 3: SDLC Phase Ranking by AI Effect and Hours Saved

Phase	Integration Level	AI Effect	Improvement %	Hours Saved	% of Total Savings
Development	93.90%	0.682	68.20%	8.89 hrs	38.40%
Requirements	91.80%	0.666	66.60%	3.46 hrs	14.90%
Design	85.90%	0.627	62.70%	4.09 hrs	17.60%
Testing	84.80%	0.622	62.20%	5.40 hrs	23.30%
Planning	36.50%	0.274	27.40%	0.95 hrs	4.10%
Deployment	17.00%	0.127	12.70%	0.39 hrs	1.70%

4.3.3 H2 Summary

The Impact Modifier is relatively consistent across all phases and ranges from 72.5% for Requirements to 75.1% for Planning. The range spans only 2.6 percentage points, which indicates that developers perceived similar improvement benefits from AI integration across all workflows. The results varied substantially for Integration Level, which ranges from 17.0% for Deployment to 93.9% for Development, a 5.5x variation. This observation indicates that the primary impact driver for the AI Effect is the Integration Level, which is the adoption rate, and not the Impact Modifier, which is the perceived benefit. The simulation results demonstrated that AI integration does not produce equal improvements across all SDLC phases. The Development phase has the highest AI Effect and showed the highest percentage improvement at 68.2%. It also showed the largest absolute time savings at 8.89 hours. The Requirements phase comes in second highest in AI Effect at 0.666 with a 66.6% improvement. The AI Effect values

are derived from fixed survey parameters and not subject to random variation across all the simulation runs. The ranking of Development as the highest phase is determined by the adoption rates in the survey. This rejects the null hypothesis that the Development phase demonstrates improvement that is less than or equal to any other phase in the SDLC.

Table 4- 4: H2 Statistical Summary

Metric	Results
Development AI Effect	0.682
Development Cycle Time Improvement	68.2%
H2 Predicted Threshold	25%
Observed vs. Predicted	Exceeds by 43.2 percentage points
Impact Modifier Range (all phases)	72.5% – 75.1% (2.6pp variation)
Integration Level Range (all phases)	17.0% – 93.9% (5.5x variation)
Primary Driver of AI Effect Variation	Integration Level (adoption rate)

4.4 H3 Results

4.4.1 H3 Hypothesis Statement

The research hypothesis H3 states: AI integration cycle time improvements vary by SDLC phase, phases with low adoption achieve less than 30% reduction while higher-adoption phases achieve greater than 50% reduction in phase cycle time.

The hypothesis statements for H3 are as follows:

- H_0 (Null Hypothesis): Phase-specific cycle time improvements do not vary, with no phases falling below 30% or above 50%.

  H_0: $\mu_{PhaseImprovement_j}$ $does\ not\ vary < 30\%\ or > 50\%\ across\ phases$

- H_1 (Alternative Hypothesis): Phase-specific cycle time improvements vary, phases with lower adoption achieve less than 30% reduction and higher-adoption phases achieve greater than 50% reduction.

$$H_1\colon\ \mu_{PhaseImprovement_j}\ varies: < 30\%\ for\ low -$$

$$adoption\ phases\ and > 50\%\ for\ higher - adoption\ phases$$

4.4.2 H3 Statistical Analysis

The phase-specific cycle time improvements varied across the SDLC phases and the observed improvements fell within the predicted thresholds of less than 30% for phases with low adoption and greater than 50% for those with higher adoption. Development was highest at 68.2% with an Integration Level = 93.9%, followed by Requirements at 66.6% with Integration Level = 91.8%. Design at 62.7% with Integration Level = 85.9%, and Testing at 62.2% with Integration Level = 84.8% also showed larger improvements. In comparison, Planning at 27.4% with Integration Level = 36.5% and Deployment at 12.7% with Integration Level = 17.0% showed smaller gains. The four higher-adoption phases, Development, Requirements, Design, and Testing, each exceeded the 50% threshold predicted in H3, while the two phases with lower adoption, Planning and Deployment, each showed gains below the 30% threshold. The phase-specific improvements ranged from 12.7% to 68.2%, with a 5.5x variation, and demonstrated that AI integration does not produce uniform improvements across all phases. The main driver of this variation is the Integration Level or the adoption rate, which ranged from 17.0% for Deployment to 93.9% for Development. The Impact Modifier ranged from 72.5% to 75.1%, which is relatively consistent across all the phases and confirmed the adoption rate, not the perceived benefit, drives the differences in phase-specific results. Development contributed the largest savings at 8.89 hours or 38.4% of total savings, followed by Testing at 5.40 hours or 23.3%, Design at 4.09 hours or 17.6%, and Requirements at 3.46 hours or 14.9%. Planning at 0.95 hours or 4.1%, and

Deployment at 0.39 hours or 1.7% ranked at the bottom two. Testing ranked higher than Requirements and Design in absolute time savings due to its higher time allocation of 21.7%.

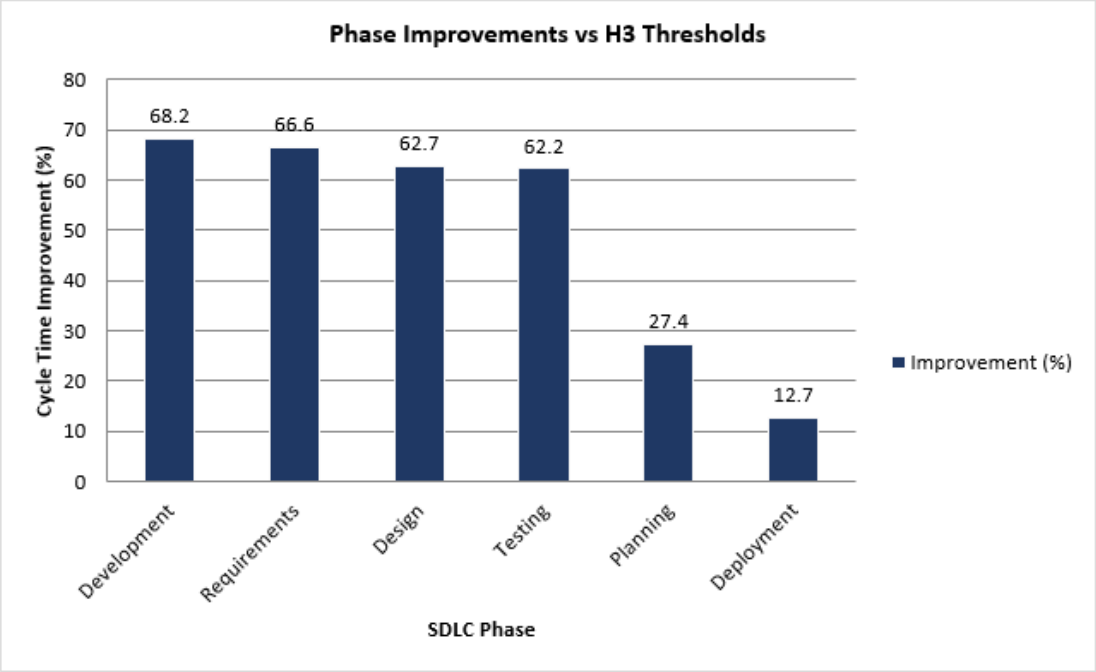


Figure 4-3. Phase Improvements vs. H3 Thresholds

For the statistical validation, the paired t-tests conducted across all 10,000 simulation runs exceeded the thresholds for t-critical = 1.66 for one-tailed at both α = 0.05 and α = 0.001. All 10,000 simulation runs produced p-value < 0.001. The mean Cohen's d = 9.96, which exceeded the "Large" threshold of 0.8 by 12.5 times. The 95% Confidence Interval of [22.71, 23.64] hours for the overall mean difference demonstrated replication stability. The sensitivity analysis tests showed that even when the AI Effect

scale was reduced by 20%, the improvements for phases with higher adoption remained above 50%, while the lower-adoption phases remained below 30%.

4.4.3 H3 Summary

The phase-specific cycle time improvements varied by SDLC phase, with results ranging from 12.7% for Deployment to 68.2% for Development. The four higher-adoption phases each exceeded the 50% threshold predicted in H3, while the two phases with lower adoption each showed results below the 30% threshold. The t-statistic of 99.56 exceeded the threshold for t-critical = 1.66, with p-values consistently < 0.001 across all simulation runs. The Cohen's d of 9.96 exceeded the threshold 0.8 for "Large" effect size. This rejects the null hypothesis that phase improvements do not vary below 30% or above 50%. The simulation results validated the phase-specific thresholds established in H3 and demonstrated the Integration Level parameter is a primary contributor to phase-specific cycle time improvements.

Table 4- 5: H3 Statistical Summary

Statistic	Results	Interpretation
t-statistic	99.56	Exceeds t-critical = 1.66
p-value	< 0.001	All 10,000 runs
Cohen's d	9.96	Exceeds 'Large' threshold = 0.8 by 12.5x
95% CI	[22.71, 23.64] hrs	—
Phase improvement range	12.7% – 68.2%	5.5x variation
Low-adoption phases (<30%)	Planning (27.4%), Deployment (12.7%)	H3 threshold met
High-adoption phases (>50%)	Development (68.2%), Requirements (66.6%), Design (62.7%), Testing	H3 threshold met

4.5 Sensitivity Analysis Results

A sensitivity test was conducted using One-at-a-time (OAT) analysis as described in Chapter 3. Each parameter used in the simulation was varied individually while all other parameters were held at their base values. The test used the same master random seed = 42 to ensure that the results are isolated and attributed to the parameter change and not to some other random variation.

The AI Effect was scaled by ±20% variation. AI Effect is the product of Integration Level and Impact Modifier and scaling this parameter is equivalent to individually scaling either of these parameters separately. The results show that the AI Effect Scale was the most influential parameter. The ±20% variation for this parameter demonstrated improvements that ranged from 46.4% to 69.6%, a 23.2 percentage point difference. At their conservative low end of -20%, the results showed a 46.4% improvement.

The Coefficient of Variation controlled the randomness of the task times. As discussed in Chapter 3, the CV value was derived from COCOMO II. The simulation set CV = 0.20 as the base case. A CV = 0.10 represents a well-managed project where cycle times are close to their expected time. A CV = 0.30 represents projects with tasks that have more variability. The results show that the mean improvement remained at 58.0% across all CV values from 0.10 to 0.30. This is because the randomness is applied equally to both the baseline and AI-enabled tasks and the AI Effect is always the same. However, the CV did affect the 95% Confidence Interval width, from 0.47 hours at CV = 0.10 with CI = [22.94, 23.41] to 1.39 hours at CV = 0.30 with CI = [22.48, 23.87]. The results indicate that in projects with more variability, the lowest projected improvement is

still greater than zero. The CV also demonstrated an effect on Cohen's d values, where Cohen's d = 19.9 at CV = 0.10 and Cohen's d = 6.6 at CV = 0.30. In both cases, the Cohen's d exceeded the 0.8 threshold for "Large" effect size.

The results for Base Cycle Time showed no effect on percentage improvement, which remained at 58.0% whether the base task is set to 20 hours or 80 hours. The change only affected the absolute hours saved proportionally with both baseline and AI-enabled cycle times, from 11.6 hours to 46.4 hours. This is due to the proportional structure of the model as discussed in Section 3.7.

The Cohen's d and t-statistic demonstrated invariance. As noted in the methodology discussion, the AI Effect Scale and Base Cycle Time parameters are uniform scaling factors that affected both the mean paired difference and the standard deviation proportionally, which cancels out in the Cohen's d and t-statistic calculations. The results of the sensitivity analysis show both Cohen's d and t-statistic remain unchanged and only showed variation with a change to the CV parameter since altering the CV also alters the stochastic distribution itself.

Table 4- 6: Sensitivity Analysis Results

Parameter	Variation	Mean Improvement %	CI Width (hrs)	Cohen's d	t-stat
AI Effect Scale	+20%	69.60%	—	Unchanged	Unchanged
AI Effect Scale	-20%	46.40%	—	Unchanged	Unchanged
CV	0.10	58.00%	0.47 [22.94, 23.41]	19.9	Unchanged
CV	0.30	58.00%	1.39 [22.48, 23.87]	6.6	Unchanged
Base Cycle Time	20 hrs	58.00%	—	Unchanged	Unchanged
Base Cycle Time	80 hrs	58.00%	—	Unchanged	Unchanged

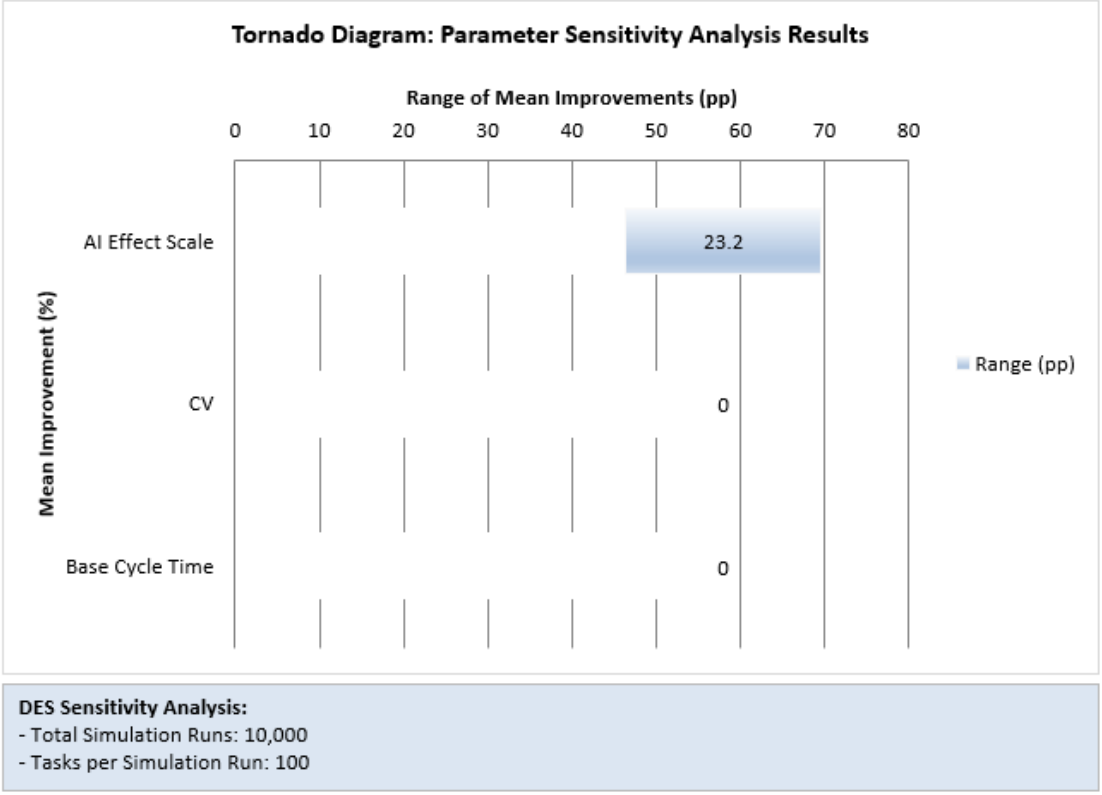


Figure 4-4. Sensitivity Results Tornado Diagram

4.6 Summary and Conclusion

This chapter presented the results of the DES simulation runs and provided statistical analysis which successfully answered the research questions and validated the research hypotheses. The overall cycle time projection of 58% from the simulation is consistent with the range of the published benchmark findings reviewed in Section 2.11 of the research literature review. These are the 55.8% finding from the controlled experiment conducted by Peng et al. (2023), the 42.36% finding from Chatterjee et al. (2024), and the 26.08% finding from Cui et al. (2025). The results from the validation proved that the research methodology met the research objective of developing a Discrete

Event Simulation model that projects cycle time improvements from AI integration across the SDLC phases.

Chapter 5—Discussion and Conclusions

5.1 Discussion

Despite the recent advancements in software development life cycle methodologies, inefficiencies in the process continue to be an issue and software development teams struggle with the challenge of keeping projects on track and within budget. Research has shown that incorporating artificial intelligence into specific phases of the SDLC demonstrated improvements in these phases. However, adoption of AI tools is sporadic and isolated. Published AI productivity improvement studies such as Peng et al. (2023), Cui et al. (2025), and Chatterjee et al. (2024) demonstrated the potential for incorporating AI into the software development process. However, these studies were focused primarily on development phase activities such as coding, debugging, and code review. These studies do not measure the cumulative effect of AI integration across all SDLC phases from planning through deployment. This praxis fills that gap by providing a simulation model that projects end-to-end AI-driven cycle time improvement across all six SDLC phases, with phase-specific breakdowns that demonstrate where AI integration has the highest and lowest returns. Phase-specific projections provide engineering managers with a prioritization tool that previous single-phase studies cannot provide. Having the capability to distinguish which phase yields the highest improvement and knowing that Integration Level is the main driver for improvements enables targeted investment decisions.

The objective of this praxis was to investigate the impact of AI integration across the SDLC and to provide a tool for projecting cycle time improvements. The research

demonstrated it was able to accomplish its objectives as described in Chapter 1. It examined AI integration across six SDLC phases and calculated the cycle time improvements utilizing a DES model with parameters derived from the 2025 Stack Overflow Developer Survey, and time allocations adapted from COCOMO II. The simulation results from the DES model provided evidence for validating the research hypotheses and aligned closely with findings from published benchmarks in the literature review. The outputs from the simulation model provide insight into prioritization of AI implementation in the SDLC phases. Based on the values derived from the survey data, the main driver for the AI Effect variation is the adoption rate as represented by the Integration Level. This allows software development teams to plan for AI adoption with a focus on phases where adoption is low, but the perceived benefit is high.

5.2 Conclusions

The objective of this praxis was to examine and demonstrate the impact of AI integration across the software development life cycle. The praxis achieved its objectives and demonstrated projected cycle time improvements utilizing a Discrete Event Simulation model. The study validated the research hypotheses through statistical analysis and hypothesis testing.

The simulation results across all 10,000 runs demonstrated an overall improvement of 58%, exceeding the predicted improvement of 20% in H1.

The phase-specific analysis confirmed that the Development phase has the highest improvements at 68.2%, which exceeded the predicted 25% in H2. Development also contributed the largest absolute time savings at 8.89 hours per task due to having the

highest AI Effect and the largest time allocation at 32.6%. The results varied by Integration Level and ranged from 17.0% for Deployment to 93.9% for Development.

The simulation confirmed the phase-specific improvement thresholds predicted in H3, with the phase-specific improvements ranging from 12.7% to 68.2%. Each of the four phases with higher adoption exceeded the 50% threshold. These are: Development at 68.2%, Requirements at 66.6%, Design at 62.7%, and Testing at 62.2%. Each of the two phases with lower adoption showed improvements below the 30% threshold. These are: Planning at 27.4% and Deployment at 12.7%. The variation is primarily driven by Integration Level rather than the Impact Modifier, which remained relatively consistent across all phases at 72.5% to 75.1%.

The sensitivity analysis confirmed the proportionality of the AI Effect scale, with projected overall improvement remaining at 46.4% even when scaled down by 20%.

The overall projected cycle time reduction of 58% from the simulation aligns closely with the 55.8% finding from the controlled experiment conducted by Peng et al. (2023) from the literature review and is consistent with the range of published benchmarks reviewed in Section 2.11, which provides external validation for the simulation results. The simulation can serve as a tool for software engineering managers to identify which phases to prioritize when planning for AI implementation, and to estimate the magnitude of productivity and efficiency improvements.

The simulation is calibrated to routine, well-defined development tasks represented by a one-week sprint with a Coefficient of Variation set to 0.20, which represents moderate task variability for well-managed projects as described in Section 3.4. Engineering managers should note that projected improvements are most applicable

at this level of project complexity. The published benchmarks support this. Cui et al. (2025) found that junior developers gained the most from AI integration with improvements ranging from 35-39%, while senior developers showed little to no statistically significant improvement. This suggests that AI productivity gains are stronger on structured, well-defined tasks and expected improvements should be adjusted downward for highly complex or research-intensive projects.

5.3 Contributions to Body of Knowledge

This Praxis contributes to the Engineering Management body of knowledge, specifically to the Software Engineering community, by providing a simulation-based decision support tool for AI implementation planning. These contributions are as follows:

1. Developed a simulation model that projects AI integration cycle time impact across SDLC phases. Prior AI productivity improvement research was phase-specific, with published benchmarks focused primarily on development phase coding tasks. This praxis provided cumulative end-to-end cycle time projections across all SDLC phases with phase-specific breakdowns. Software development metrics for AI usage are scarce. This praxis translates survey derived AI adoption and impact parameters into end-to-end cycle time projections that engineering managers can use to prioritize AI investment across the full SDLC.
2. Developed a method for translating the developer survey responses into simulation parameters using the AI Effect calculation based on the expected value theory (Raiffa, 1968) as described in Section 3.5 in Chapter 3.

3. Validated the results of the simulation projection of 58.0% against published benchmarks from literature review, and demonstrated the model produced results that aligned with independently conducted experiments.
4. Identified that the adoption rate, represented by the Integration Level variable, drives the variation in AI effectiveness across phases.
5. Implemented the simulation in C# with a fixed random seed set to 42, allowing for replication and updates to the parameters with new survey data. This was verified using an Excel-based model with frozen random values of the Run 1 tasks from the simulation output.

5.4 Recommendations for Future Research

There were limitations on the research related to data availability as outlined in Chapter 1. Future research could address these limitations as AI adoption becomes increasingly prevalent and data is available that will allow researchers to obtain objective measurements and multiple-year surveys. Recommendations for future research are as follows:

1. Repeat the parameter derivation with future survey data to track how AI adoption rates and perceived impact have changed over time.
2. Recalibrate simulation parameters to use organization-specific metrics such as internal adoption surveys and actual cycle time measurements.
3. Compare simulation projections from various software development organizations against measured cycle time metrics before and after AI integration.

4. Extend the model to include other factors such as resource allocation constraints, organization size or dynamics, and developer experience levels to evaluate capacity implications of software development cycles.
5. Develop simulation models capturing other outcomes such as code quality and defect rates to investigate the full impact of AI integration effects beyond cycle time savings.

References

Ahmad, A., Feng, C., Khan, M., Khan, A., Ullah, A., Nazir, S., & Tahir, A. (2020). A systematic literature review on using machine learning algorithms for software requirements identification on Stack Overflow. *Security and Communication Networks, 2020*, Article 8830683. https://doi.org/10.1155/2020/8830683

Alenezi, M., & Akour, M. (2025). AI-driven innovations in software engineering: A review of current practices and future directions. *Applied Sciences*, 15(3), Article 1344. https://doi.org/10.3390/app15031344

Azeem, M. I., Palomba, F., Shi, L., & Wang, Q. (2019). Machine learning techniques for code smell detection: A systematic literature review and meta-analysis. *Information and Software Technology, 108*, 115–138. https://doi.org/10.1016/j.infsof.2018.12.009

Banh, L., & Strobel, G. (2023). Generative artificial intelligence. *Electronic Markets, 33*(1), Article 63. https://doi.org/10.1007/s12525-023-00680-1

Banks, J., Carson, J. S., II, Nelson, B. L., & Nicol, D. M. (2010). *Discrete-event system simulation* (5th ed.). Pearson Prentice Hall.

Binkhonain, M., & Zhao, L. (2019). A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications: X, 1*, Article 100001. https://doi.org/10.1016/j.eswax.2019.100001

Blackford, J. P. (2024). *EMSE 8099: Survey of research formulation for engineering management* [Lecture notes]. The George Washington University.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D., & Steece, B. (2000). *COCOMO II model definition manual* (Version 2.1). Center for Software Engineering, University of Southern California. https://www.cs.montana.edu/courses/spring2004/352/public/cocomo/modelman.pdf

Borges, O., Couto, J., Ruiz, D. D. A., & Prikladnicki, R. (2020, May). How machine learning has been applied in software engineering? In *Proceedings of the 22nd International Conference on Enterprise Information Systems* (pp. 170–181). SCITEPRESS.

Box, G. E. P., & Muller, M. E. (1958). A note on the generation of random normal deviates. *The Annals of Mathematical Statistics, 29*(2), 610–611. https://doi.org/10.1214/aoms/1177706645

Chatterjee, S., Liu, C. L., Rowland, G., & Hogarth, T. (2024). *The impact of AI tool on engineering at ANZ Bank: An empirical study on GitHub Copilot within corporate environment*. arXiv. https://arxiv.org/abs/2402.05636

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Routledge. https://doi.org/10.4324/9780203771587

Conger, S. (2013). Software development life cycles and methodologies: Fixing the old and adopting the new. *All Sprouts Content*, 428. https://aisel.aisnet.org/sprouts_all/428

Cui, Z., Demirer, M., Jaffe, S., Musolff, L., Peng, S., & Salz, T. (2025). *The effects of generative AI on high-skilled work: Evidence from three field experiments with software developers.* SSRN. https://doi.org/10.2139/ssrn.4945566

Dam, H. K., Tran, T., Grundy, J., Ghose, A., & Kamei, Y. (2018). *Towards effective AI-powered agile project management*. arXiv. https://arxiv.org/abs/1812.10578

Davis, J., Hoffert, J., & Vanlandingham, E. (2016). A taxonomy of artificial intelligence approaches for adaptive distributed real-time embedded systems. In *2016 IEEE International Conference on Electro Information Technology (EIT)* (pp. 233–238). IEEE. https://doi.org/10.1109/EIT.2016.7535246

Durelli, V. H. S., Durelli, R. S., Borges, S. S., Endo, A. T., Eler, M. M., Dias, D. R. C., & Guimaraes, M. P. (2019). Machine learning applied to software testing: A systematic mapping study. *IEEE Transactions on Reliability, 68*(3), 1189–1209. https://doi.org/10.1109/TR.2019.2892517

Dutta, S. (1997). Strategies for implementing knowledge-based systems. *IEEE Transactions on Engineering Management, 44*(1), 79–90. https://doi.org/10.1109/17.552810

Gupta, S. K., Gunasekaran, A., Antony, J., Gupta, S., Bag, S., & Roubaud, D. (2019). Systematic literature review of project failures: Current trends and scope for future research. *Computers & Industrial Engineering, 127*, 274–285. https://doi.org/10.1016/j.cie.2018.12.002

Hamby, D. M. (1994). A review of techniques for parameter sensitivity analysis of environmental models. *Environmental Monitoring and Assessment, 32*(2), 135–154. https://doi.org/10.1007/BF00547132

Karim Zadeh, E., Bagheri Khoulenjani, A., & Safaei, M. (2024). Integrating AI for agile project management: Innovations, challenges, and benefits. *International Journal of Industrial Engineering and Construction Management, 1*(1), 26–35. https://www.researchgate.net/publication/382241441

Kellner, M. I., Madachy, R. J., & Raffo, D. M. (1999). Software process simulation modeling: Why? What? How? *Journal of Systems and Software, 46*(2–3), 91–105. https://doi.org/10.1016/S0164-1212(99)00003-5

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lei, K., Wang, J., Yu, Y., Chen, W., & Li, L. (2021). Research on heuristic deployment and scheduling strategy of cloud resources oriented to cost optimization. In *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)* (pp. 1–6). IEEE. https://doi.org/10.1109/ICCCNT51525.2021.9579835

Liu, K., Reddivari, S., & Reddivari, K. (2022). Artificial intelligence in software requirements engineering: State-of-the-art. In *2022 IEEE 23rd International Conference on Information Reuse and Integration for Data Science (IRI)* (pp. 149–155). IEEE. https://doi.org/10.1109/IRI54793.2022.00034

Madachy, R. J. (2008). *Software process dynamics*. Wiley-IEEE Press. https://doi.org/10.1002/9780470192719

Munikoti, S., Stewart, I., Horawalavithana, S., Kvinge, H., Emerson, T., Thompson, S. E., & Pazdernik, K. (2024). *Generalist multimodal AI: A review of architectures, challenges and opportunities*. arXiv. https://arxiv.org/abs/2406.05496

NIST/SEMATECH e-Handbook of Statistical Methods. (n.d.). Upper critical values of Student's t-distribution with degrees of freedom. Retrieved February 22, 2026, from https://www.itl.nist.gov/div898/handbook/eda/section3/eda3672.htm

Necula, S.-C., Dumitriu, F., & Greavu-Șerban, V. (2024). A systematic literature review on using natural language processing in software requirements engineering. *Electronics, 13*(11), Article 2055. https://doi.org/10.3390/electronics13112055

Ngangue Ndih, E. D., & Cherkaoui, S. (2015). Simulation methods, techniques and tools of computer systems and networks. In M. S. Obaidat, P. Nicopolitidis, & F. Zarai (Eds.), *Modeling and simulation of computer networks and systems* (pp. 485–504). Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-800887-4.00017-1

Pandey, S. K., Mishra, R. B., & Tripathi, A. K. (2021). Machine learning based methods for software fault prediction: A survey. *Expert Systems With Applications, 172*, Article 114595. https://doi.org/10.1016/j.eswa.2021.114595

Pandi, S. B., Binta, S. A., & Kaushal, S. (2023). *Artificial intelligence for technical debt management in software development*. arXiv. https://arxiv.org/abs/2306.10194

Peng, S., Kalliamvakou, E., Cihon, P., & Demirer, M. (2023). *The impact of AI on developer productivity: Evidence from GitHub Copilot*. arXiv. https://doi.org/10.48550/arXiv.2302.06590

Prakash, M. (2024). *Role of generative AI tools (GAITs) in software development life cycle (SDLC) - Waterfall model* [Master's thesis, Massachusetts Institute of Technology]. MIT System Design and Management Program.

Qiu, S., Xu, H., Deng, J., Jiang, S., & Lu, L. (2019). Transfer convolutional neural network for cross-project defect prediction. *Applied Sciences, 9*(13), Article 2660. https://doi.org/10.3390/app9132660

Rahmaniar, W. (2024). ChatGPT for software development: Opportunities and challenges. *IT Professional, 26*(3), 80–86. https://doi.org/10.1109/MITP.2024.3379831

Raiffa, H. (1968). *Decision analysis: Introductory lectures on choices under uncertainty.* Addison-Wesley.

Saltelli, A., Tarantola, S., Campolongo, F., & Ratto, M. (2004). *Sensitivity analysis in practice: A guide to assessing scientific models.* Wiley.

Shamim, M. M. I. (2024). Artificial intelligence in project management: Enhancing efficiency and decision-making. *International Journal of Management Information Systems and Data Science, 1*(1), 1–6. https://doi.org/10.62304/ijmisds.v1i1.107

Shehab, M., Abualigah, L., Jarrah, M. I., Alomari, O. A., & Daoud, M. S. (2020). Artificial intelligence in software engineering and inverse: Review. *International Journal of Computer Integrated Manufacturing, 33*(10-11), 1129–1144. https://doi.org/10.1080/0951192X.2020.1780320

Sheikh, H., Prins, C., & Schrijvers, E. (2023). Artificial intelligence: Definition and background. In *Mission AI: Research for policy*. Springer. https://doi.org/10.1007/978-3-031-21448-6_2

Singh, S., & Sambhav, S. (2023). Application of artificial intelligence in software development life cycle: A systematic mapping study. *Lecture Notes in Networks and Systems, 617*, 655–665. https://doi.org/10.1007/978-981-19-9512-5_60

Sofian, H., Md Yunus, N. A., & Ahmad, R. (2022). Systematic mapping: Artificial intelligence techniques in software engineering. *IEEE Access, 10*, 51021–51039. https://doi.org/10.1109/ACCESS.2022.3174115

Stack Overflow. (2025). *2025 Stack Overflow developer survey*. https://survey.stackoverflow.co/2025/

Trinh, E., Funke, M., Lago, P., & Bogner, J. (2024). Sustainability integration of artificial intelligence into the software development life cycle. In *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)* (pp. 199–206). IEEE. https://doi.org/10.1109/ICSA-C63560.2024.00044

Washizaki, H. (Ed.). (2024). *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide)* (Version 4.0). IEEE Computer Society. https://www.swebok.org

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data mining: Practical machine learning tools and techniques* (3rd ed.). Elsevier.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2020). *A comprehensive survey on transfer learning*. arXiv. https://arxiv.org/abs/1911.02685

Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., & Aftandilian, E. (2024). Measuring GitHub Copilot's impact on productivity. *Communications of the ACM, 67*(3), 54–63. https://doi.org/10.1145/3633453