34th Annual INCOSE international symposium

hybrid event

Dublin, Ireland
July 2 - 6, 2024

Raymond Madachy, PhD, *Naval Postgraduate School*
Ryan Longshore, *Naval Postgraduate School, Naval Information Warfare Center Atlantic*
Ryan Bell, *Naval Postgraduate School, Naval Information Warfare Center Atlantic*

# Systems and Software Engineering Cost Modeling for AI Assistance

## Introduction

- AI assistant tool usage is transforming systems engineering and software development processes.
- – E.g., ChatGPT, Gemini, Claude, Copilot, others
- AI can help generate system artifacts for all lifecycle aspects
- Addressing new challenges in disrupted parametric cost models for systems and software engineering

## AI Background

- Large Language Models (LLMs) are a type of Generative AI that utilize a deep learning algorithm to generate human-like text based on natural language prompts
- Typically interface with a chatbot (GPT-3.5 = model, ChatGPT = chatbot)
- Well suited for tasks such as language translation, text summarization, and question answering
- Some LLMs are exceptionally good at generating code
- Cons:
  - Can generate false information
  - Very large models require advanced GPUs and lots of data to train (time, cost)
  - Open models may not be appropriate for sensitive information

## Objectives

- Develop a road map for advancing cost models.
- Leverage existing modeling and measurement frameworks.
- Understand, codify and measure AI’s advantages and pitfalls.
- Recognize different usage scenarios and impacts across systems and software engineering phases and activities.

## Modeling and Measurement Framework

- Introduction of initial *AI Assistance Usage* factor for software cost modeling using Constructive Cost Model (COCOMO) framework
- Development of Delphi data collection instrument and process to calibrate it.
- Introduction of *query points* as a new size measure.
- Setting the stage for further exploration into systems engineering process impacts.

# Potential Cost Decreases by Activity

| Activity | Benefits |
|---|---|
| Requirements | AI tools can help clarify terminologies and concepts on-the-fly, reducing the need for prolonged meetings or external consultations. Developers or analysts can query AI tools for insights or comparative analysis, which might help in refining requirements. |
| Design | Developers can quickly consult AI tools for recommended design patterns or architectural practices suitable for their problem. Preliminary design ideas can be discussed with AI tools for quick feedback. |
| Code | Developers can seek assistance on coding challenges, syntax, and algorithmic solutions. AI tools can assist in code reviews, highlighting potential pitfalls or anti-patterns. |
| Testing and Integration | AI tools can suggest potential edge cases or testing scenarios. For failing tests or integration issues, developers can discuss potential causes and solutions with AI tools. |
| Maintenance | AI tools can assist in understanding old codebases, suggesting potential refactoring techniques or identifying deprecated methods. For known errors or bugs, AI tools can suggest common solutions or workarounds. |

# Potential Cost Risks by Activity

| Activity | Downfalls |
|---|---|
| Requirements | Relying too much on AI tools for domain-specific knowledge might lead to missed nuances that an expert in the field would be aware of. |
| Design | Over-relying on AI for design decisions without human review can lead to suboptimal choices. |
| Code | If developers use AI tool suggestions verbatim without understanding, it might introduce bugs or inefficient code. Waiting on AI tool responses for every small issue can become a crutch and delay development if developers stop trying to problem-solve on their own. |
| Testing and Integration | If AI tool suggestions are taken without thorough review, it might lead to unnecessary tests or efforts spent on non-issues. |
| Maintenance | If teams over-rely on AI tools for maintenance, they might overlook deeper architectural or design issues that require human expertise. |

# Constructive Effort Formula for Systems and Software Engineering

$$\text{Effort}(PM) = A * Size^{B} * EAF$$

where

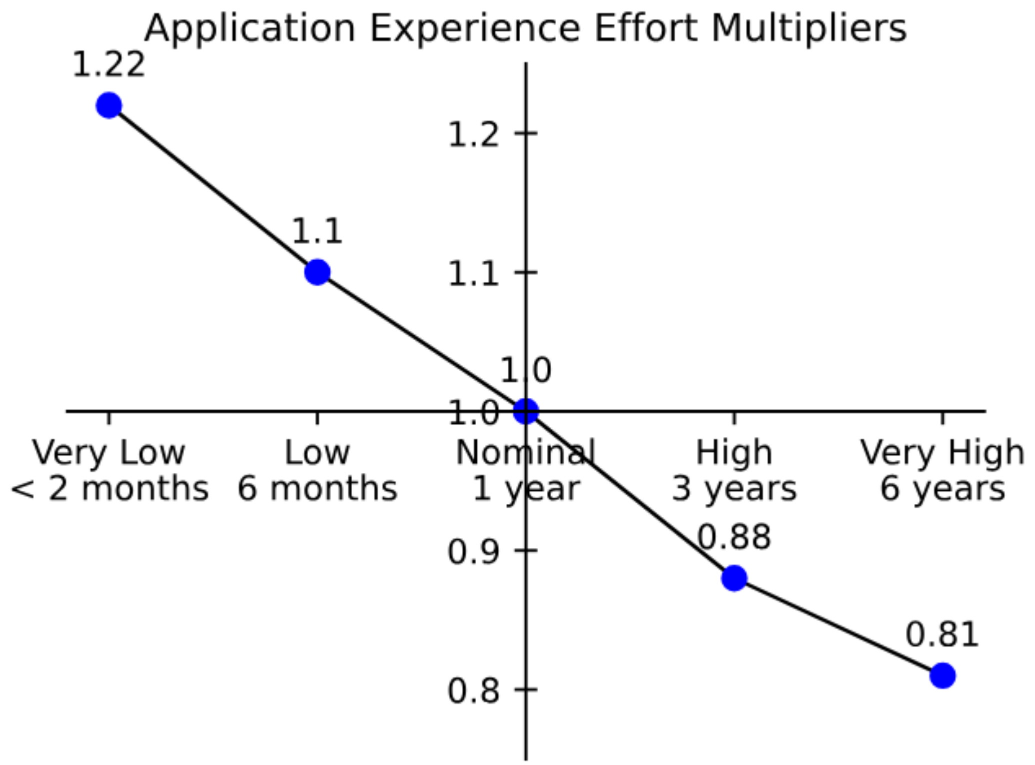
- Effort($PM$) is labor measured in Person-Months (PM)
- $A$ is a constant derived from historical project data.
- $B$ is a factor for the diseconomy of scale
- $Size$ is a measure of the product.
  - software size in KSLOC (thousand source lines of code), converted from function points or other size measures [2]
  - system size as aggregate of requirements, interfaces, algorithms and scenarios weighted for difficulty [6]
- $EAF$ is an Effort Adjustment Factor (EAF) to the nominal effort defined as a product of cost driver Effort Multipliers (EMs) per:

$$EAF = \prod_{i=1}^{N} EM_i$$

Where $EM_i$ is the effort multiplier for the $i^{th}$ cost driver. The product results in an overall multiplier to the nominal effort.

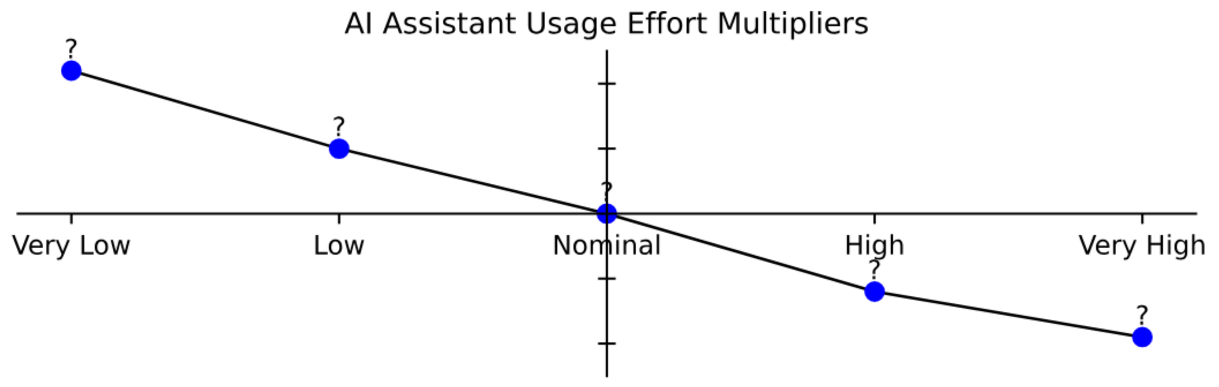
# Software Application Experience Cost Factor Example

- Effort multipliers for each rating represent the relative effort to Nominal.
- The EM for a Very Low rating of Applications Experience is 1.22 indicating a 22% increase in effort from Nominal.
- The Nominal rating is always 1.0 by definition for a typical project.
- Very High EM is 0.81 for 81% effort compared to nominal, or a decrease of 19% effort from Nominal.
- The overall Effort Multiplier Ratio (EMR) for Applications Experience is the ratio of the highest to lowest multipliers, or 1.22/.81 = 1.5.



# Initial Rating Scale for AI Assistance Usage

| Very Low | Low | Nominal | High | Very High |
|---|---|---|---|---|
| • Minimal to no AI assistance.<br>• Development relies primarily on traditional methods and tools.<br>• AI tools may be present but are rarely, if ever, consulted. | • Occasional AI consultation, typically for clarification or basic information retrieval.<br>• AI tools are not deeply integrated into the development workflow. | • Regular use of AI tools for various tasks like code help, design insights, or testing assistance.<br>• AI tools are a recognized part of the toolkit but aren't central to development. | • Frequent and strategic use of AI assistance.<br>• AI tools play a central role in multiple phases of development, from design to code review. | • AI tools are deeply ingrained in most development phases. They are crucial for decision making, problem solving, and automating specific tasks.<br>• The development process is designed around maximizing AI tool benefits. |



# Affected Software Cost Factors and Parameters

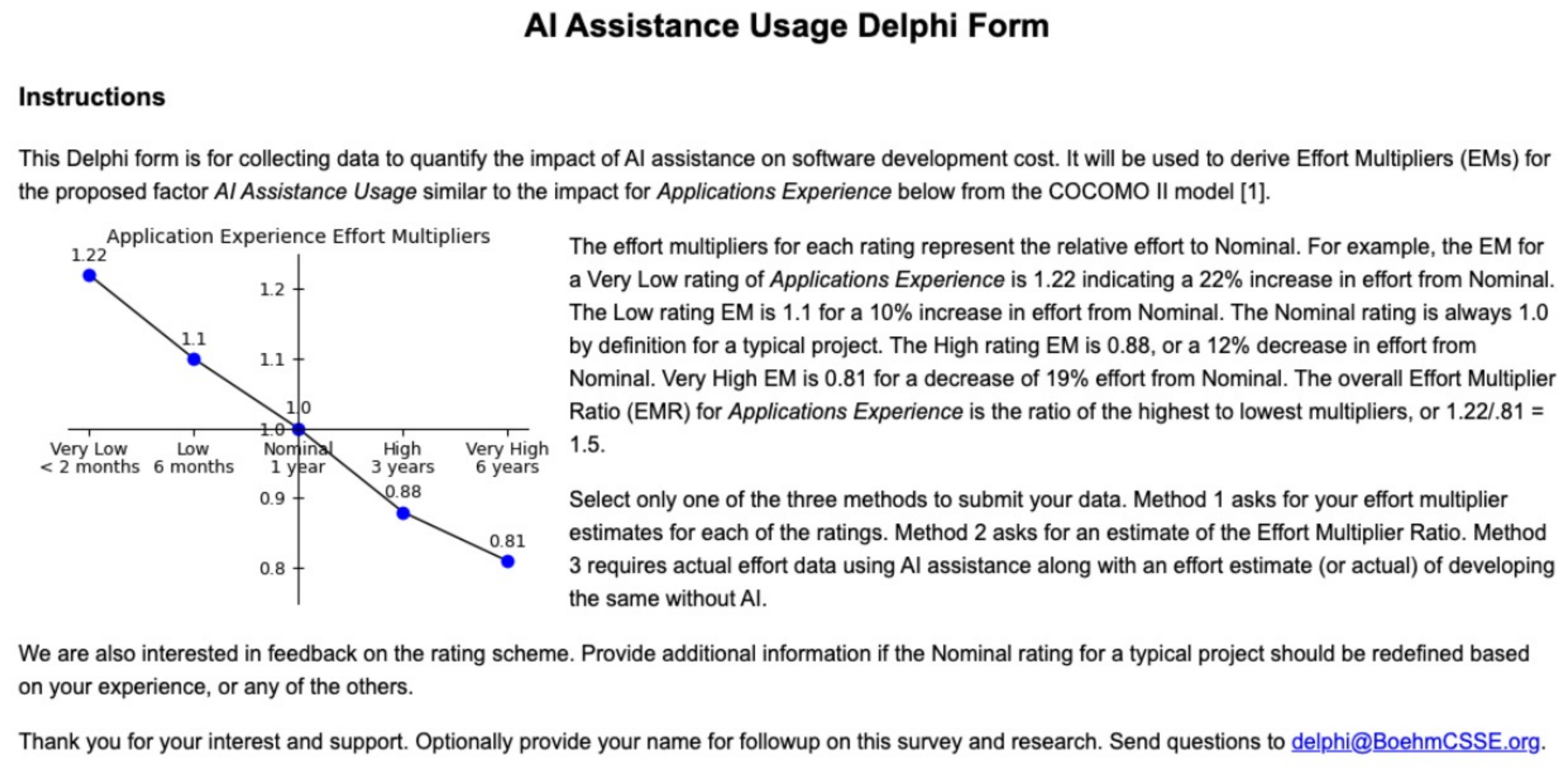
- Cost Factor Definitions
  - AI Assistant Usage may augment *Use of Software Tools* or be separate
  - Use of AI assistants becomes part of required skillset for Personnel Capability factors
- Multipliers and Calibrations
  - *Required Software Reliability*
  - *Product Complexity*
  - *Personnel Capability*
  - *Personnel Experience*
  - *Team Cohesion*
  - Overall model constants *A* and B

## Data Collection and Analysis Methods

- Multi-project data collection in conjunction with other cost factors, e.g. COCOMO II and III.
- Controlled group experiments, e.g. Github Copilot [4].
- Delphi surveys for expert judgment.
- Bayesion approaches combining empirical project data and delphi results, e.g. COCOMO II [2], [3]
- Small-scale empirical case studies and expert judgment.

# Delphi Form

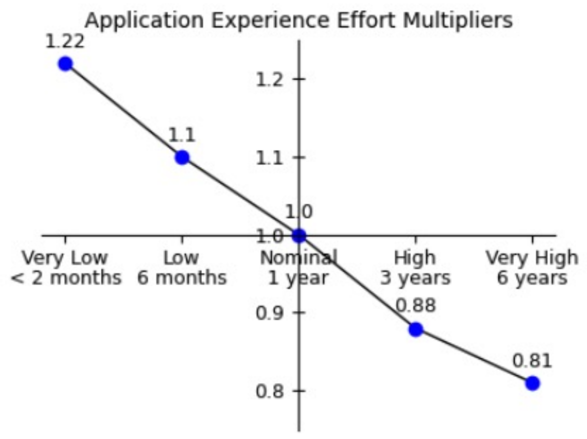
- Submit at http://softwarecost.org/data/ai



## AI Assistance Usage Delphi Form

### Instructions

This Delphi form is for collecting data to quantify the impact of AI assistance on software development cost. It will be used to derive Effort Multipliers (EMs) for the proposed factor *AI Assistance Usage* similar to the impact for *Applications Experience* below from the COCOMO II model [1].



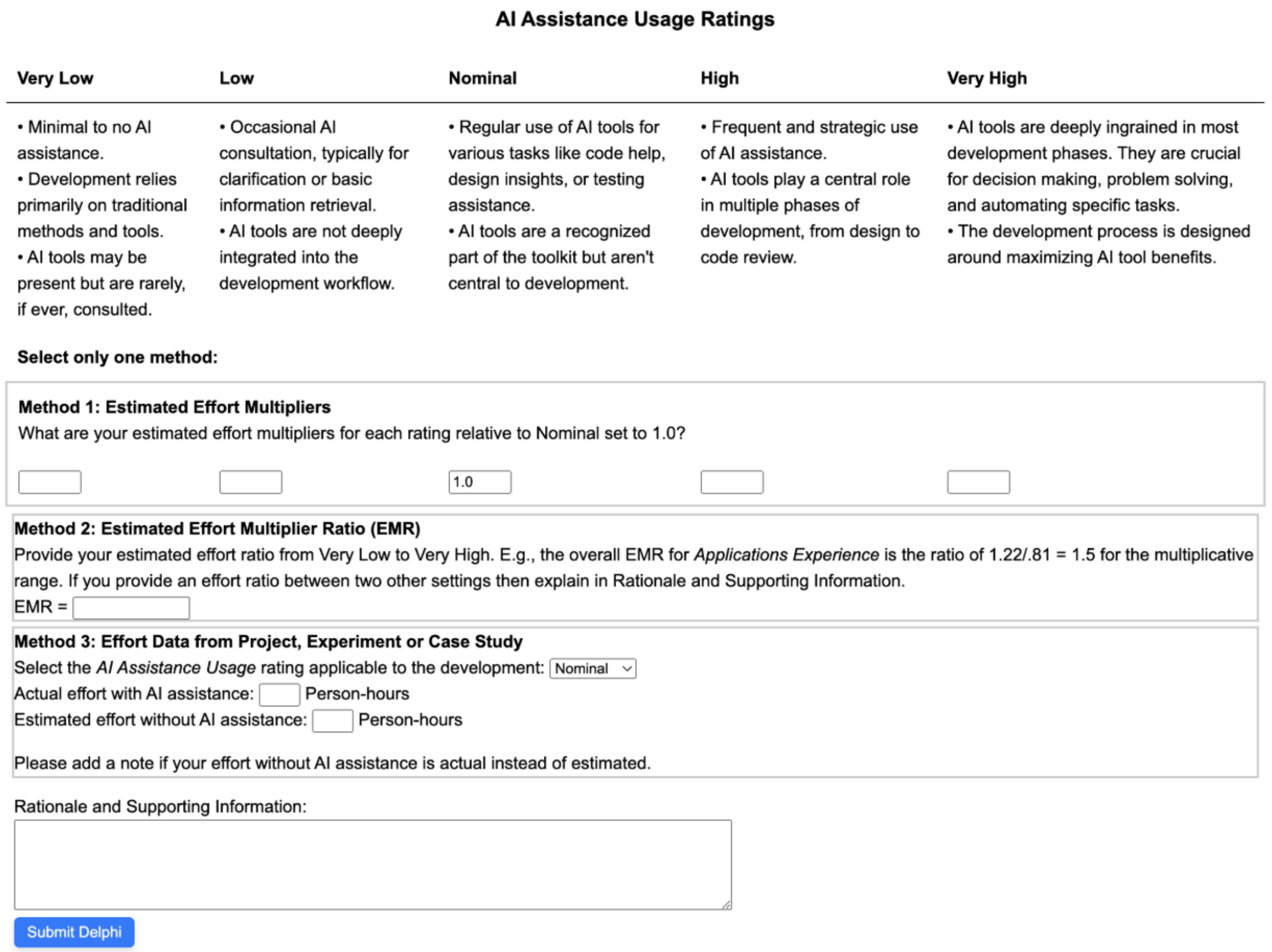
The effort multipliers for each rating represent the relative effort to Nominal. For example, the EM for a Very Low rating of *Applications Experience* is 1.22 indicating a 22% increase in effort from Nominal. The Low rating EM is 1.1 for a 10% increase in effort from Nominal. The Nominal rating is always 1.0 by definition for a typical project. The High rating EM is 0.88, or a 12% decrease in effort from Nominal. Very High EM is 0.81 for a decrease of 19% effort from Nominal. The overall Effort Multiplier Ratio (EMR) for *Applications Experience* is the ratio of the highest to lowest multipliers, or 1.22/.81 = 1.5.

Select only one of the three methods to submit your data. Method 1 asks for your effort multiplier estimates for each of the ratings. Method 2 asks for an estimate of the Effort Multiplier Ratio. Method 3 requires actual effort data using AI assistance along with an effort estimate (or actual) of developing the same without AI.

We are also interested in feedback on the rating scheme. Provide additional information if the Nominal rating for a typical project should be redefined based on your experience, or any of the others.

Thank you for your interest and support. Optionally provide your name for followup on this survey and research. Send questions to delphi@BoehmCSSE.org.

# Delphi Form (Cont.)



**AI Assistance Usage Ratings**

| Very Low | Low | Nominal | High | Very High |
|---|---|---|---|---|
| • Minimal to no AI assistance. • Development relies primarily on traditional methods and tools. • AI tools may be present but are rarely, if ever, consulted. | • Occasional AI consultation, typically for clarification or basic information retrieval. • AI tools are not deeply integrated into the development workflow. | • Regular use of AI tools for various tasks like code help, design insights, or testing assistance. • AI tools are a recognized part of the toolkit but aren't central to development. | • Frequent and strategic use of AI assistance. • AI tools play a central role in multiple phases of development, from design to code review. | • AI tools are deeply ingrained in most development phases. They are crucial for decision making, problem solving, and automating specific tasks. • The development process is designed around maximizing AI tool benefits. |

**Select only one method:**

**Method 1: Estimated Effort Multipliers**
What are your estimated effort multipliers for each rating relative to Nominal set to 1.0?

| | | | | |
|---|---|---|---|---|
| | | 1.0 | | |

**Method 2: Estimated Effort Multiplier Ratio (EMR)**
Provide your estimated effort ratio from Very Low to Very High. E.g., the overall EMR for *Applications Experience* is the ratio of 1.22/.81 = 1.5 for the multiplicative range. If you provide an effort ratio between two other settings then explain in Rationale and Supporting Information.
EMR =

**Method 3: Effort Data from Project, Experiment or Case Study**
Select the *AI Assistance Usage* rating applicable to the development: Nominal
Actual effort with AI assistance: Person-hours
Estimated effort without AI assistance: Person-hours

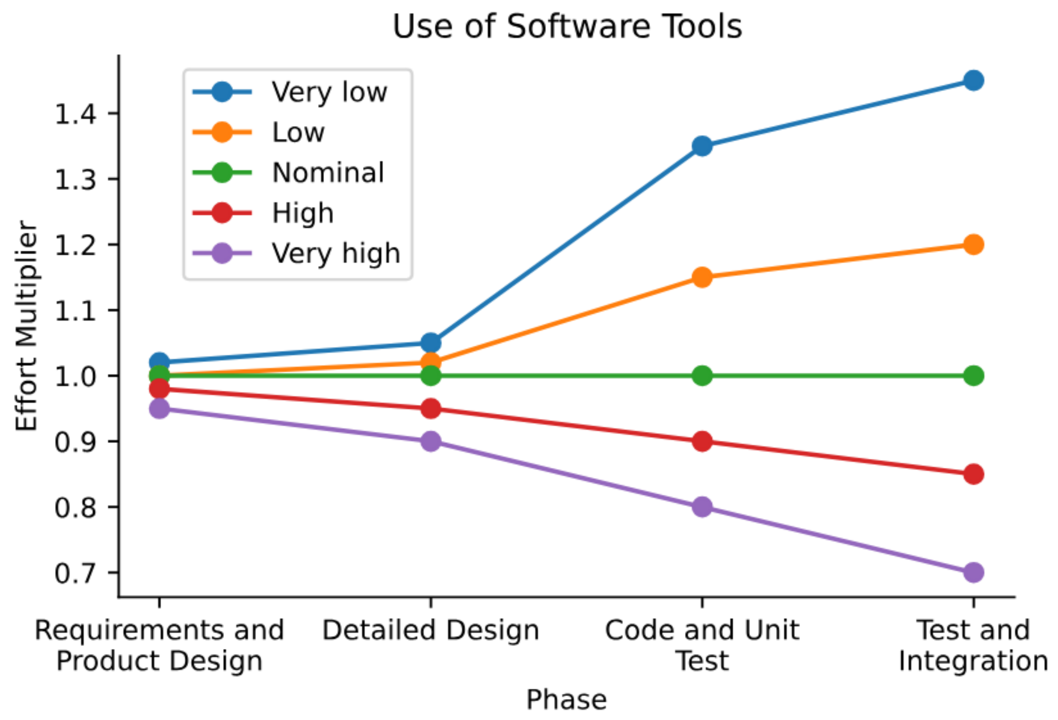
Please add a note if your effort without AI assistance is actual instead of estimated.

Rationale and Supporting Information:

Submit Delphi

# Detailed COCOMO Framework for Phase Impacts

- Phase-sensitive effort multipliers account for different impacts across lifecycle
  - E.g., *Use of Software Tools* from COCOMO 81 [1]
- Some cost drivers vary more across phases than others
- *AI Assistance Usage* impact to be evaluated by phases and activities
  - Codifying the practices and ratings with detailed descriptions
  - Empirical data collection commensurate with phases



## Ideal Effort Multiplier

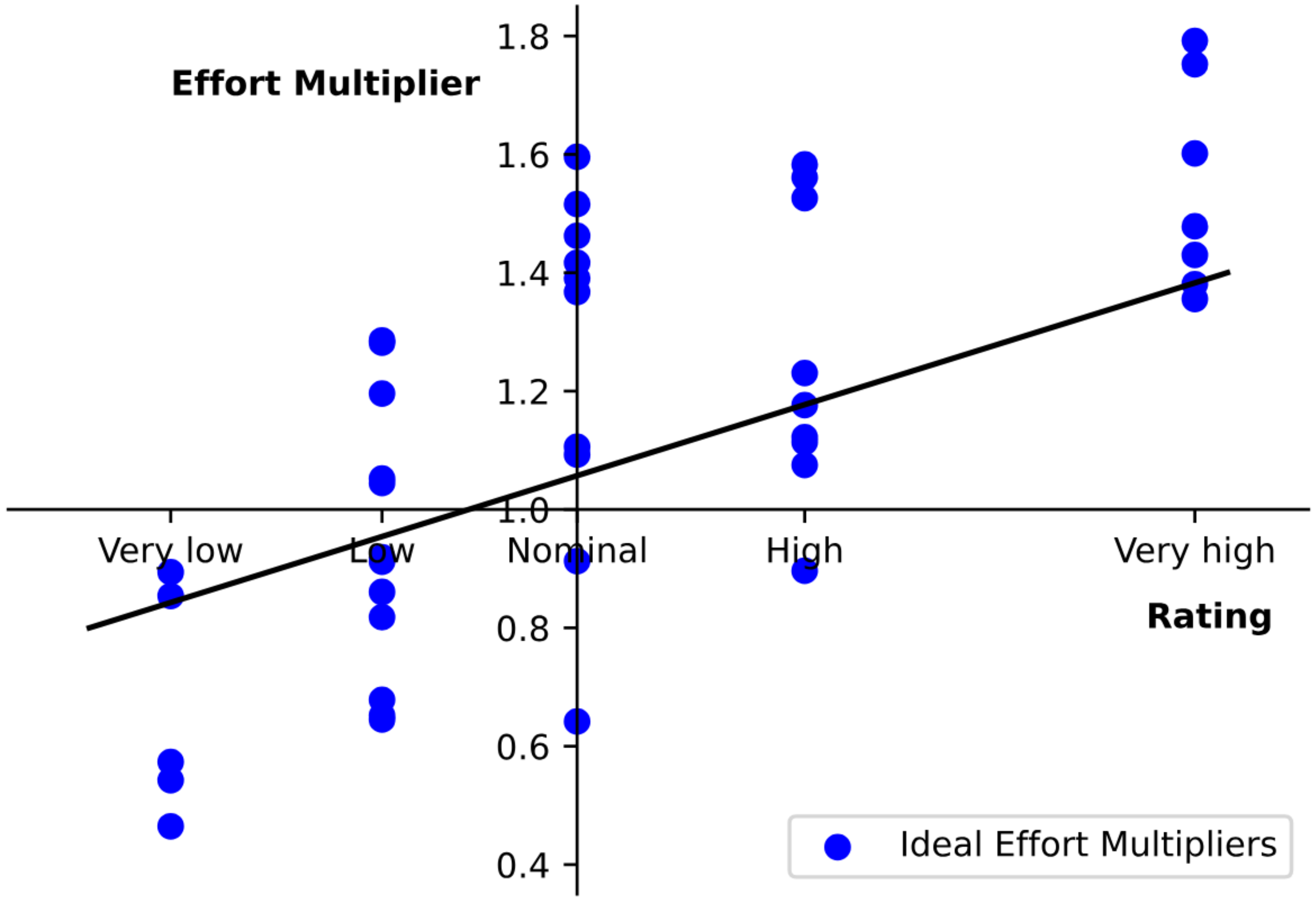
- Method to normalize out contaminating effects of other individual cost factors in order to isolate the contribution of the factor being analyzed on productivity.
- In our case, to analyze the contribution of AI Assistance Usage eliminating other cost factor sources of variance.

$$IEM(P, \text{Cost Factor}) = PM(P, actual) / PM(P, \text{Cost Factor})$$

where

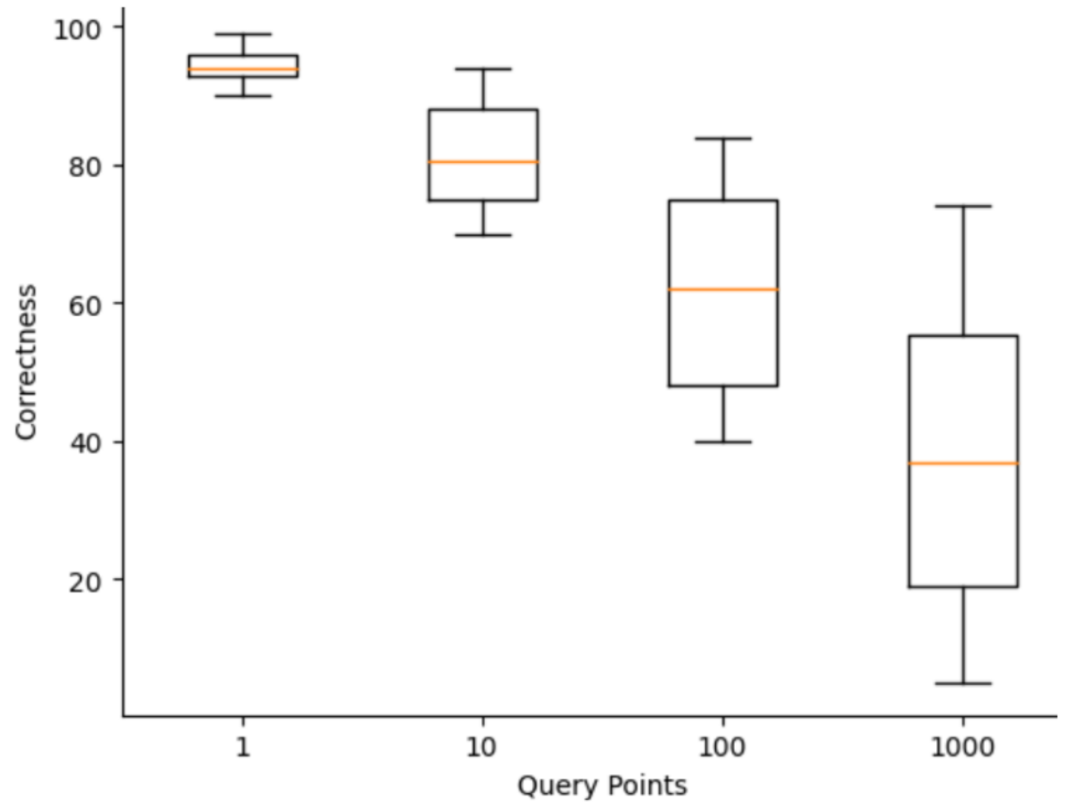
- *IEM*(*P*, Cost Factor) is the ideal effort multiplier for project P
- *PM*(*P*, *actual*) is the actual development effort of project P
- *PM*(*P*, Cost Factor) is the cost model estimate excluding the Cost Factor
- *PM* is Person-Months of effort

# Ideal Effort Multiplier (Cont.)



# Query Points

- Adapted from function points to quantify the size and complexity of the AI generated solutions.
  - normalize correctness and defect density measures
  - measure scale effects on generated solutions
  - for software, lines of code are increasingly less useful to gauge effort when generated
- Clear initial evidence that AI assistants do not scale
- Notional example of expected results:



# Case Study: COSYSMO Program

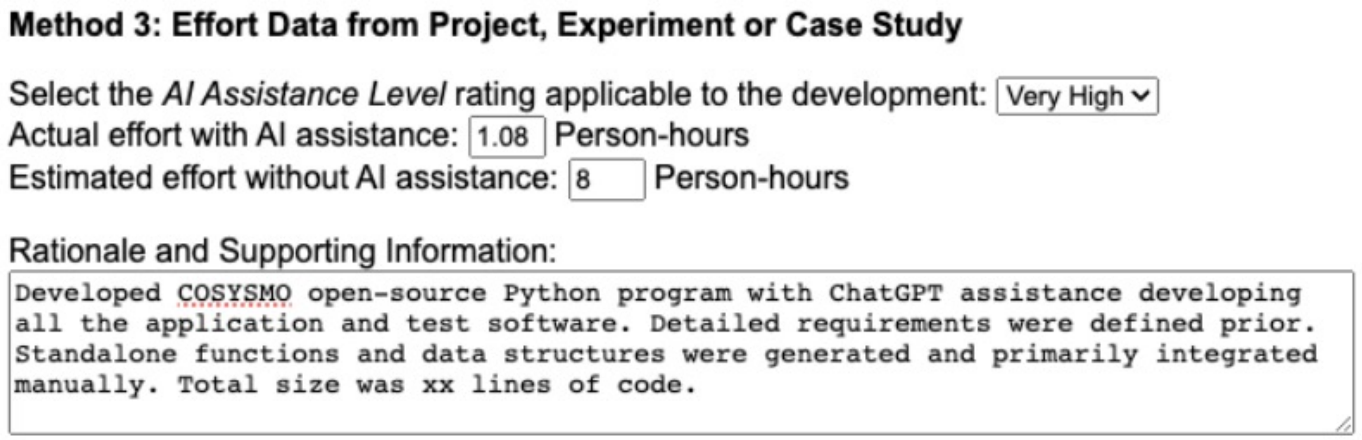
- Goal: Develop COSYSMO [6] open-source software program with complete cost drivers, effort multipliers, sizing model and effort decomposition.
- Test-driven development strategy with small iterations
- Preconditions in place due to earlier failed experiments with ChatGPT to correctly develop COCOMO II from the ground up
  - Known COSYSMO formula
  - Existing table of cost factors and effort multipliers for pasting
  - Desired cost factor dictionary format example
  - Test harness template
  - Initial test cases defined

## Case Study: COSYSMO Program (Cont.)

- Steps with ChatGPT Assistance, where each step is a single query and response followed by regression testing.
  1. Generate COSYSMO test harness with defined test cases
  2. Generate cost factor dictionary
  3. Generate Effort Adjustment Factor function
  4. Update test case harness for cost factor dictionary input
  5. Generate phase effort decomposition function
  6. Generate size weight dictionary of dictionaries
  7. Generate total equivalent size function using size weights
- Results
  – Zero defects on part of ChatGPT. All tests passed throughout iterations.
  – Actual effort to generate, integrate and test was 65 minutes
  – Estimated effort without ChatGPT assistance 8 hours.

## Case Study: COSYSMO Program (Cont.)

- Delphi data submittal:



**Method 3: Effort Data from Project, Experiment or Case Study**

Select the *AI Assistance Level* rating applicable to the development: Very High

Actual effort with AI assistance: 1.08 Person-hours

Estimated effort without AI assistance: 8 Person-hours

Rationale and Supporting Information:

```
Developed COSYSMO open-source Python program with ChatGPT assistance developing
all the application and test software. Detailed requirements were defined prior.
Standalone functions and data structures were generated and primarily integrated
manually. Total size was xx lines of code.
```

- Post measurement
  - Extended program by generating desktop application with Tkinter and web-based application available at http://softwarecost.org/tools/COSYSMO, both with substantial effort saved
  - Program available in Python Systems Engineering Library (se-lib) and at https://github.com/se-lib/se-lib/tree/main/lib/cost_models

## Case Study: Mapping Application with Python and PyQt

- Goal: Provide an application in Python that can display custom maps, custom plot overlays, point selection on the map, and import and export files
  1. Integrate low fidelity maps
     - AI Assisted: 15 mins
     - Projected without Assistance: 45 mins
  2. Integrate custom plot overlays on the map
     - AI Assisted: 30 mins
     - Projected without Assistance: 4 hours
  3. Integration of point selection on the map
     - AI Assisted: 15 mins
     - Projected without Assistance: 30 mins
  4. File import and export
     - AI Assisted: 15 mins
     - Projected without Assistance: 60 mins
- Results: AI Assisted: 1 hour 15 mins. Projected Without Assistance: 6 hours 15 mins. Estimated 80% time reduction. Assistance usage High. Final EMR = 5

## Case Study: SysML 2 Models

- Goal 1: Query a SysML v2 requirements model for a vehicle to retrieve requirements and automate analysis of proposed alternatives using natural language prompts.
- Goal 2: Modify SysML v2 structural model of a vehicle during system design phase using natural language prompts.
  - SysML v2 models can be generated in textual or graphical notation. Textual notation lends itself to use of LLMs. •
  - Previous attempts to generate SysML v2 models from LLMs had many errors (e.g. mixed in concepts/syntax from SysML v1.x that are not valid in SysML v2)
  - Steps with ChatGPT Assistance, where each step is a single query
    1. Provide ChatGPT with baseline model
    2. Prompt ChatGPT with modification or query
    3. Inspect Output for accuracy. Repeat Step 2 if needed.

# Case Study: SysML 2 Models (Cont.)

- Results
  - ChatGPT was able to make all modifications. However, some required multiple prompts to implement the change. For simple modifications, insignificant time savings using ChatGPT. *Estimated 50-75% time savings for medium to high complexity modifications.*
  - ChatGPT was able to accurately perform all simple (search and recall queries) with no errors. *Insignificant time savings using ChatGPT.*
  - ChatGPT was able to perform more complex queries with a high degree of success, but made some mistakes. *Estimated time savings of 50% on high complexity queries (e.g. does a design solution meet the requirements).*
  - Time savings may be reduced by need to verify ChatGPT results.

## Future Work

- Institute Delphi data collection and iterative analysis with COCOMO III research.
- Further analysis of AI tool impacts across lifecycle aligning artifacts and effort data with ISO/IEC/IEEE 15288 systems and software engineering phases and activities [5].
- Provide open source tools with new factor(s) in the models.
- Research focus on how query task complexity impacts AI correctness and effort impact.
- Address large scale team and enterprise processes assisted with AI.
- Address AI tool volatility and model updates for reproducible results.

# References

[1] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[2] B. Boehm, C. Abts, W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. *Software Cost Estimation with COCOMO II*. Prentice-Hall, 2000.

[3] S. Chulani, B. Boehm, and B. Steece. “Bayesian Analysis of Empirical Software Engineering Cost Models”. In: IEEE Transactions on Software Engineering 25.4 (Dec. 1999).

[4] GitHub. Research: quantifying GitHub Copilot’s impact on developer productivity and happiness. url: https://github.blog/2022-09-07-researchquantifying-github-copilots-impact-on-developerproductivity-and-happiness/, 2022

[5] International Organization for Standardization. ISO/IEC 15288:2015 Systems engineering – System life cycle processes. Tech. rep. International Organization for Standardization, 2015.

[6] R. Valerdi. “The constructive systems engineering cost model (COSYSMO)”. PhD thesis. Los Angeles, CA: University of Southern California, 2005.

## Contact Information

Raymond Madachy: rjmadach@nps.edu

Ryan Bell: ryan.bell@nps.edu

Ryan Longshore: ryan.longshore@nps.edu