AFRL-RI-RS-TR-2023-078

IDAS EVALUATION TEAM

PROVATEK, LLC.

MAY 2023

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2023-078 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /

WILLIAM E. MCKEEVER
Work Unit Manager

/ S /

GREGORY J. HADYNSKI
Assistant Technical Advisor
Computing and Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
MAY 2023	FINAL TECHNICAL REPORT	MAY 2020	JUNE 2022

4. TITLE AND SUBTITLE
IDAS EVALUATION TEAM

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
FA8750-20-C-0209	N/A	62303E

5d. PROJECT NUMBER	5e. TASK NUMBER	5f. WORK UNIT NUMBER
		R2Z9

6. AUTHOR(S)
Timothy Fraser, Jake Decker, Matthew Evenson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Provatek LLC
15301 Amberly Drive, Suite 250
Tampa FL 33647

8.PERFORMING ORGANIZATIONREPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10.SPONSOR/MONITOR'S ACRONYM(S)	11.SPONSOR/MONITOR'S REPORT NUMBER(S)
Air Force Research Laboratory/RITA 525 Brooks Road Rome NY 13441-4505	AFRL/RI	AFRL-RI-RS-TR-2023-078

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. PA#: AFRL-2023-2047
Date Cleared: 28 APRIL 2023

13. **SUPPLEMENTARY NOTES**

14. ABSTRACT

Updating legacy software to meet changing mission requirements and environmental constraints is a huge cost for DoD. The Intent-Defined Adaptive Software (IDAS) program sought to develop and demonstrate novel techniques and tools to reduce this cost or even avoid it entirely. This report presents the results of the IDAS research effort.

15. SUBJECT TERMS

Intent-Defined Adaptive Software, IDAS, Controller-Oriented Programming, Symlang

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
U	U	U	SAR	50

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
WILLIAM E. MCKEEVER	N/A

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

PREVIOUS EDITION IS OBSOLETE.

Table of Contents

List of Figures

List of Tables

1.0 SUMMARY

Updating legacy software to meet changing mission requirements and environmental constraints is a huge cost for the Department of Defense (DoD. A piece of software represents the accumulated consequences of many decisions made by developers. Some of these decisions are fundamental and necessary to solve the problem posed by the mission requirements. Others are unimportant choices between equally good options made to fill gaps the requirements left open. Even current best-practice agile software development methods accept the idea that many of the seemingly unimportant daily decisions developers make to fill gaps will paint them into a corner when those requirements change tomorrow. To update legacy software, developers first must untangle the unimportant decisions they can change from the fundamental ones that must stand. Then begins the lengthy redesign and rework of the old software's foundations needed to simply set the stage for meeting the new requirements. Today's agile methods accept the immense cost of this analysis and rework as an unfortunate fact of life and make a virtue out of tackling it with expensive labor hours. The Intent-Defined Adaptive Software (IDAS) program sought to develop and demonstrate novel techniques and tools to reduce this cost or even avoid it entirely.

IDAS began in May 2020 with seven teams and a four-year series of planned evaluation and demonstration activities. There were four Technical Area (TA) 1 research teams: Perspecta Labs, Apogee, Vanderbilt University, Grammatech. The goal of TA1 is to create technologies that enable software engineers to develop and verify adaptive software through a deferred-concretization methodology.

Provatek acted as the TA3 Integrated Test and Evaluation team charged with planning, executing, and reporting on evaluations, measuring TA1 technical progress, and identifying complimentary approaches. A TA2 Carnegie Mellon University Software Engineering Institute (CMU/SEI) Problem Set Generation team and a TA4 Sierra Nevada Corporation (SNC) Control team aided this effort. The Government revised its plans for the IDAS program in May 2021, before the first evaluation produced any results.

In the revised IDAS program, TA3 Provatek executed a single experiment that compared the effort required to solve a set of software development challenge problems using TA1 Apogee's experimental Controller Oriented Programming (COP) design methodology and Symlang programming language to the effort required using traditional design methods and the popular Python programming language. The experiment found no significant difference between the amount of effort required by the two approaches.

2.0 INTRODUCTION

Updating legacy software to meet changing mission requirements and environmental constraints is a huge cost for the Department of Defense (DoD). A piece of software represents the accumulated consequences of many decisions made by developers. Some of these decisions are fundamental and necessary to solve the problem posed by the mission requirements. Others are unimportant choices between equally good options made to fill gaps the requirements left open. Even current best-practice agile software development methods accept the idea that many of the seemingly unimportant daily decisions developers make to fill gaps will paint them into a corner when those requirements change tomorrow. To update legacy software, developers must first untangle the unimportant decisions they can change from the fundamental ones that must stand. Then begins the lengthy redesign and rework of the old software's foundations needed to simply set the stage for meeting the new requirements. Today's agile methods accept the immense cost of this analysis and rework as an unfortunate fact of life and make a virtue out of tackling it with expensive labor hours. The Intent-Defined Adaptive Software (IDAS) program sought to develop and demonstrate novel techniques and tools to reduce this cost or even avoid it entirely.

IDAS began in May 2020 with seven teams and a four-year series of planned evaluation and demonstration activities. There were four Technical Area (TA) 1 research teams:

- The TA1 Perspecta Labs team sought to automate the process of distinguishing unimportant decisions from fundamental ones and mapping the impacts of requirements changes to very specific pieces of code.
- The TA1 Apogee team sought to avoid this analysis cost with a novel design methodology, design pattern, and programming language that sought to separate the results of fundamental and unimportant decisions from the start.
- The TA1 Vanderbilt team sought to reduce the cost of the post-analysis rework with tools that enabled agile developers to make incremental revisions while maintaining an argument that those revisions hadn't invalidated key correctness properties.
- The TA1 Grammatech team sought to reduce rework cost by broadening the kinds of situations where developers could simply throw out old code that no longer respected requirements and automatically synthesize new compliant code to replace it.

Provatek acted as the TA3 Integrated Test and Evaluation team charged with planning, executing, and reporting on evaluations, measuring TA1 technical progress, and identifying complimentary approaches. A TA2 Carnegie Mellon University Software Engineering Institute (CMU/SEI) Problem Set Generation team and a TA4 Sierra Nevada Corporation (SNC) Control team aided this effort. The Government revised its plans for the IDAS program in May 2021, before the first evaluation produced any results.

The revised IDAS program involved only two of the original teams: TA1 Apogee and TA3 Provatek. For seven months, the two teams collaborated on a single activity that had two purposes:

1. To compare the effort required to solve a set of software development challenge problems proposed by the Government and TA1 Apogee using TA1 Apogee's

experimental Controller Oriented Programming (COP) design methodology and Symlang programming language to the effort that ought to be required according to the Constructive Cost Model (COCOMO) II software development effort estimation technique [BOE2000], and

2. Produce a demonstration of COP and Symlang's ability to solve problems of interest to users with real-world operational needs.

In December 2021 the Government split the effort into two separate activities, directing TA3 Provatek to work solely on the experiment described in point 1 and TA1 Apogee to work on the operationally relevant demonstration from point 2. TA3 Provatek proceeded with an improved version of the experiment that replaced the artificial COCOMO II experimental control with actual control solutions to the challenge problems produced by developers using traditional design methods and the popular Python general-purpose programming language. TA3 Provatek completed this experiment in June 2022.

TA3 Provatek did not participate in TA1 Apogee's operationally relevant demonstration; this report contains no details on that activity.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

The following subsections describe TA3 Provatek's methods, assumptions, and procedures for its original IDAS evaluation effort (Subsection 3.1) and for the final evaluation it accomplished after the May-2021 IDAS program revision (Subsection 3.2).

3.1 The original IDAS effort

The IDAS program originally envisioned a four-year schedule containing 8 evaluations and 3 demonstrations in which the research teams would develop software to meet an initial set of challenge problem requirements and then update that software to meet several rounds of requirement changes. The May 2021 IDAS revision came before the execution of the first evaluation; there are no measurements to report for the original portion of the IDAS program. However, the following discussion of the intended structure of the original IDAS evaluations, the roles of each of the teams, and the details of their technical approaches may be of some interest to readers considering how they might structure future research programs on related topics.

3.1.1 TA3 Provatek evaluation team

TA3 Provatek evaluation team was generally responsible for planning, executing, and reporting on the IDAS evaluations and demonstrations. TA3 Provatek had the further specific responsibility to understand the technical approaches of the TA1 research teams and provide the TA2 CMU/SEI problem set generation team with guidance on what kinds of requirements changes would be *relevant* to those approaches and thus encourage interesting research results. Table 1 provides an example of this advice with excerpt from TA3 Provatek's July 2020 Technical Report #99 "TA3 Advice on Test Evaluation 1 Challenges" [FRA2020].

3.1.2 TA2 CMU/SEI problem set generation team

The TA2 CMU/SEI problem set generation team was responsible for producing the challenge problem requirements. They intended to base each evaluation and demonstration on a particular problem domain. For example, their challenge problem for the first evaluation concerned the logistics of shipping pallets of goods between warehouses by truck and enjoined the TA1 research teams to develop software capable of finding efficient assignments of pallets to trucks and trucks to routes. TA2 CMU/SEI was responsible for ensuring that the challenge problems were *realistic*, thus encouraging research results that might someday transfer or transition to real-world use. Table 2 provides an example of their challenge problem requirements with an excerpt of the requirements from TA2 CMU/SEI's September 2020 "IDAS: Requirements Specification for Logistics Domain, Transportation Subdomain – V4" [TAY2020].

3.1.3 Evaluation plan

The May 2021 revision came as TA3 Provatek was constructing a common Ubuntu GNU/Linux-based virtual machine environment in which the TA1 teams would demonstrate their solutions to each round of challenge problem requirements. TA3 Provatek was developing a suite of *public* tests the TA1 teams could use to determine when they had met the requirements for each round. For each of these public tests, TA3 Provatek developed a *private* test that confirmed compliance with the same requirement

as its public cousin but differed in specific details. TA3 Provatek intended to keep these private tests to themselves and use them to confirm that the TA1 research teams did not simply examine the public tests, determine the correct output needed to pass, and encode this correct output in a trivial program that did not truly meet the requirements.

The first evaluation was a *test evaluation*. Its goal was merely to give the teams experience with the evaluation and measurement procedures. TA3 intended to measure the amount of human effort the TA1 teams required to produce software that met each round of challenge problem requirements in terms of person-hours, and to informally survey the teams on their solution strategies and the proportion of the software their automated tools managed to synthesize from abstract models. The revision occurred before TA3 was able to execute the first evaluation; TA3 ultimately did not make these measurements.

3.1.4 TA4 SNC control team

TA4 SNC was to provide an experimental control against which TA3 could compare the effectiveness of the TA1 research approaches. They intended to solve the same challenge problems as the TA1 research teams by (in their own words) picking "the right tool for the job" from a large collection of design patterns, code and test synthesizers, frameworks, and middleware.

Figure 1 contains a diagram of their manual developer workflow. They planned to base the early stages of this workflow on the popular Agile Model Driven Design (AMDD) methodology, respecting its core tenets:

- Seeing a program pass a test suite is the only way to know that your model is valid,
- building very detailed models at the start goes too long without validation and is too risky, so
- instead, iterate through model-test-develop-test cycles.

They planned to base the later stages on the popular Test Driven Development (TDD) methodology, in which:

- developers first consider the requirements, architectural, and detailed models and create tests for behaviors the models allow, and then
- they write new application code only when prompted to do so by a failed test.

Table 1 - Example relevance advice from TA3 Provatek to TA2 CMU/SEI.

TA1 Grammatech wants to demonstrate property-based testing. This technique synthesizes tests that attempt to demonstrate that assertions in a program do not hold. In order to define meaningful assertions, TA1 Grammatech needs requirements that specify properties the solution must maintain at runtime.

(A1) Specify formats for data that the user will see, or that the solution must share with other external systems. Specify details that imply runtime checks, such as:

- allowable ranges on the values of individual record fields,
- checksums computed across multiple fields, or
- semantic relationships between fields including lengths, record counts, and links or indexes between fields.

Table 2 - Example TA2 CMU/SEI challenge problem requirements.

[E1-RND0-3016] A valid Plan is a set of one or more Routes that satisfy all submitted Requests, provided the Routes satisfy the following:

- Each Route is valid per E1-RND0-3010, E1-RND0-3011, E1-RND0-3012, E1-RND0-3013, E1-RND0-3014
- No Warehouse would, at any time during Plan execution, contain fewer than 0 Palettes of any type of inventory
- No Warehouse would, at any time during Plan execution, contain more than 3200 total Palettes of inventory
- Each Request has a Route for which the Truck ends at the Destination Warehouse within 6 hours of the requested Delivery Time (early or late).

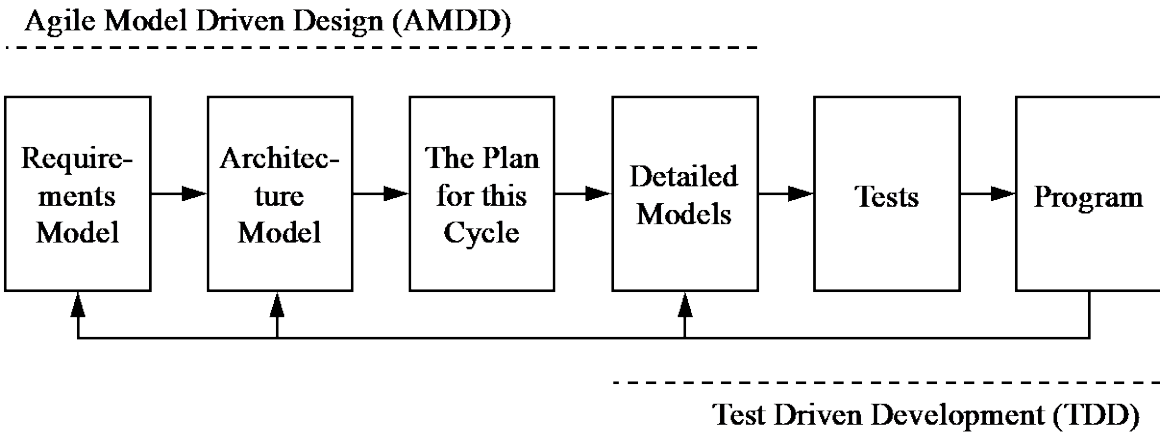

Figure 1 - TA4 SNC control team manual developer workflow.

3.2 The final COP/Symlang experiment

Provatek's final activity of the IDAS research program was an experiment that compared an experimental software development approach based on Apogee Research's experimental COP design methodology and their Symlang programming language with a control approach based on the popular Python general-purpose programming language coupled with traditional design methods.

After solving some warm-up problems, two developers each solved a series of 14 software development challenge problems twice, once using the experimental approach and once using the control approach. The first problem challenged the developers to extend a solution to an earlier warm-up problem to meet a new set of requirements proposed by the Government and Apogee. Each of the subsequent problems challenged the developers to extend their earlier solutions to meet further new or changed requirements.

Provatek compared the two approaches on three metrics:

1. extensibility in terms of the number of person-minutes of effort the developers required to produce each solution,
2. extensibility in terms of the proportion of source lines of code the developers added or changed to extend each prior solution to meet new and changed requirements, and
3. for the most challenging problem, the performance of the solutions in terms of the time and space complexity of their key algorithms.

The experiment produced no evidence that either the traditional/Python approach or the COP/Symlang approach had an advantage over the other in terms of the effort required to extend solutions or in terms of the time/space efficiency of those solutions at runtime.

Section 3.2.1 provides a brief summary of Apogee Research's experimental COP and Symlang approach and the traditional design methodologies and Python general programming language Provatek's developers chose for their control approach. Section 3.2.2 describes the design of the experiment and its series of software development challenge problems.

3.2.1 Experimental and control approaches

Apogee Research described their experimental COP design methodology and Symlang programming language in the following whitepaper:

> Lee, Holland, Gerson, Huang, Fortunato, Fisher, "Controller-Oriented Programming: A New Paradigm for Managing Software Complexity Creating Adaptable & Efficient Code through Partitions & Controllers," to appear [LEE2022].

This paper presents an example in which developers use COP and Symlang to develop and extend a dictionary-like virtual data store service that dispatches store and lookup operations to a distributed set of actual balanced-tree data stores and guarantees that lookups will complete within a particular time bound.

The COP design methodology encourages designers to identify the central element of a problem, identify the key property that would drive high-level executive decisions and assign those instances of the key elements to logical "partitions" based on this property. In the paper's example, the key element was the actual balanced-tree data stores, their key property was whether a balanced tree was about to grow too deep to permit sufficiently quick lookups, and the solutions provided one partition for data stores whose trees were on the verge of growing too deep and another partition for data stores with plenty of capacity.

Apogee tailored the Symlang programming language to support COP designs with operators to define partitions, move elements from one partition to another as their key properties change, and to call external Java routines to handle low-level operations on elements. Perhaps critically, they included special "do" operators to specify that a particular operation be handled by one, all, or some *k* elements in a particular partition. This operator enabled the paper's data store solutions to succinctly express the concept of assigning the task of storing a given element to any arbitrary data store in the plenty-of-capacity partition–all data stores in that partition being functionally equivalent from the executive logic's point of view.

In March 2022, Apogee research provided Provatek's developers with an hour-long tutorial on COP and Symlang and their latest documentation.

Provatek's developers selected the Python general-purpose programming language for the experiment's control. Python scored as one of the most popular programming languages in the 2019, 2020, and 2021 Stack Overflow developer surveys, beating C, C++, C#, and Java but losing to JavaScript. They settled on two traditional design methods:

- A control-flow-oriented approach called structured or procedural design that encourages a divide-and-conquer approach in which developers break large tasks demanded by the requirements into ever-smaller tasks until they arrive at a directed graph of tasks that are each small enough to be solved by their own routine or subroutine.
- A data-flow design approach that encourages developers to view a problem as a pipeline of transformation procedures, each of which takes an input, operates on it, and produces an output that takes the pipeline closer to the solution demanded by the requirements.

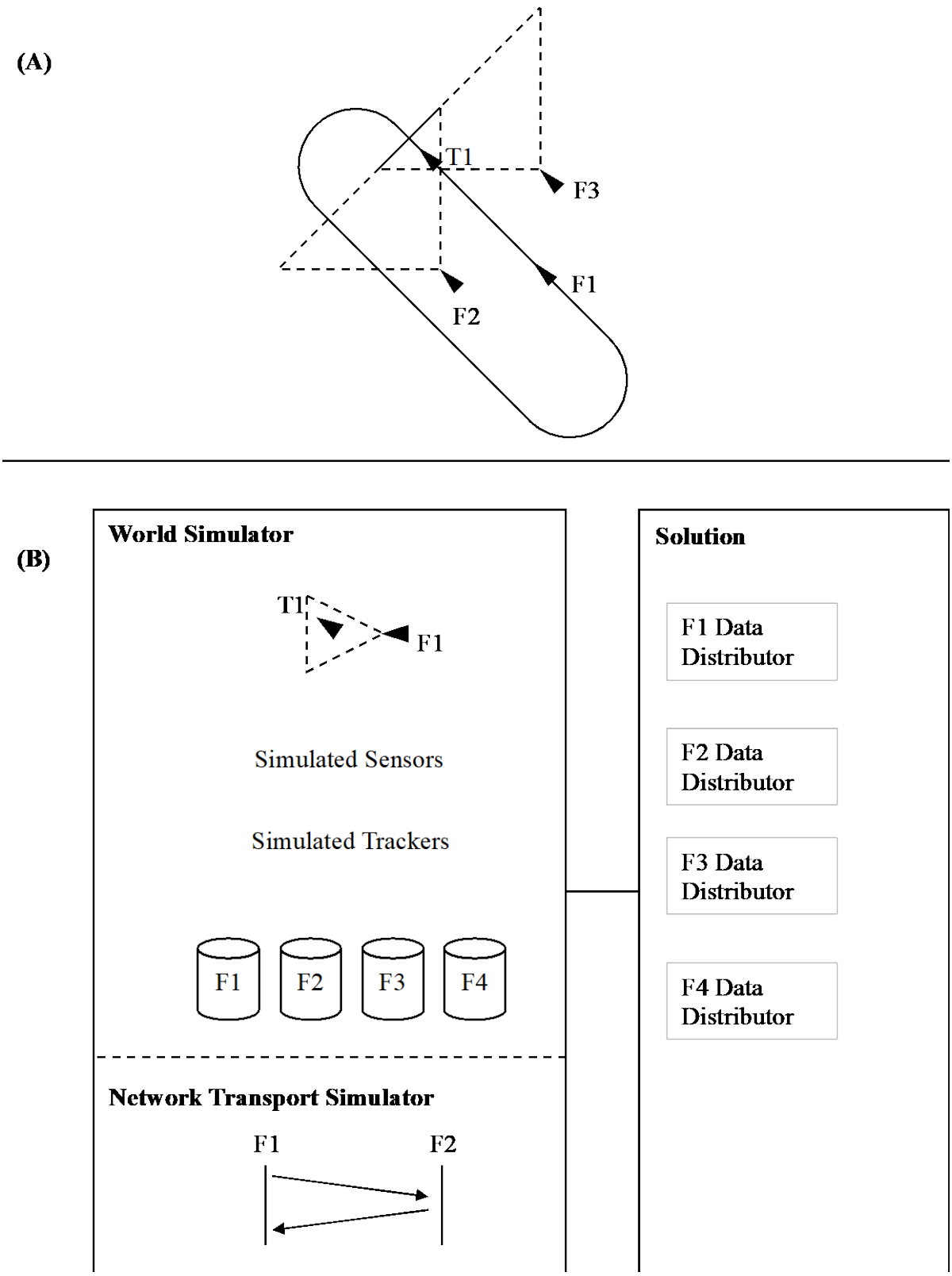

Figure 2 Rev Problem #1 (A). Test rig architecture (B).

Table 3 - Challenge problem summary.

Unmeasured warm-up problems:

- TP1 Four friendlies must exchange tracks via radio until all know all targets.
- TP2 TP1 plus friendlies must adopt the highest-quality track for each target.

Challenge problems measured in this report:

- RP1 Friendlies must combine low-quality tracks into a synthetic track that is of sufficient quality to guide a weapon to a target. This was the problem of greatest interest to Apogee Research.
- RP2 Friendly aircraft whose sensors generate tracks that are of quality sufficient to guide weapons to targets are "seers." Friendlies with weapons are "shooters." Allocate a single target to each shooter.
- JP1 RP2, but allocate each target to its closest shooter whenever possible.
- TP3 JP1 with surplus seers edge case not handled by earlier solutions.
- TP4 JP1 with an edge case where multiple targets share a closest shooter.
- TP5 JP1 with a shortage of shooters edge case.
- TP6 RP1 plus friendlies must discover whether peers are seers or shooters.
- TP7 TP4 with a surplus of shooters edge case.
- TP8 RP1 without a distinct designated friendly to act as the shooter.
- TP9 TP8 plus only one of the two potential shooters can take the shot.
- TP10 TP9 plus only one of the two potential shooters has a weapon.
- TP11 Allocate shooters to targets in a way that minimizes the total shooter-target distance across the entire flight of friendly aircraft. Each friendly carries a different number of weapons. The most difficult problem.
- TP12 TP11 plus no friendly may completely exhaust its supply of weapons.
- TP13 TP11 but each friendly aircraft must take at least one shot.

Additional challenge problems not measured in this report:

- TP14 RP2, but friendlies must not shoot targets that are close to other friendlies.
- TP15 RP2, but friendlies must not shoot when another friendly is in the way.
- TP16 TP2 but the network no longer supports broadcast messages.
- TP17 TP1 with network unreliability requiring message resends.
- TP18 TP1 with network unreliability requiring timeouts, acks, and resends.

3.2.2 The experiment

Provatek's May 2022 technical report #114 "IDAS Final Experiment Plan" describes all the details of the experiment, including the requirements for all the challenge problems and the procedures for measurement [FRA2022]. Provatek delivered this technical report to the Government as its CLIN 0002 DIN A014 Demo Plan. This subsection provides a brief summary of those details.

The plan defined 21 challenge problems. Each problem described a scenario similar to the one diagrammed in Figure 2A in which a flight of friendly aircraft (shown as deltas F1, F2, and F3 in the diagram) pursued a number of target aircraft (delta T1) around a racetrack-shaped flight path (solid oval line). Each friendly aircraft had sensors that produced tracks that estimated the locations of any hapless target aircraft who wandered into their fan-shaped field of regard (dashed triangles). Some friendly aircraft carried weapons. Those that did could guide a weapon to a target for which they had a track, subject to the following complication: The accuracy of the sensor estimates decreased as the distance from friendly to target grew; only the tracks for relatively close targets were accurate enough to guide a weapon. Friendly aircraft could share tracks with each other over a somewhat unreliable radio network, enabling a distant friendly aircraft with a weapon to gain a track from an unarmed friendly close to a target and put it to use.

The problems challenged developers to create a "data distributor" software component to run aboard each friendly aircraft. Each problem required these data distributors to cooperatively reach a particular goal condition by managing track sharing and assigning particular friendly aircraft to shoot weapons at particular target aircraft. Each problem introduced new and changed requirements and challenged developers to extend their earlier solutions to meet them.

Figure 2A's problem is Rev Problem #1 (RP1), the first problem defined by the Government and Apogee Research, named for its co-creator DARPA PM Lt. Col. Jimmy "Rev" Jones. Table 3 briefly describes the series of all 21 challenge problems. The series included three problems defined by the Government and Apogee: RP1, RP2, and with some refinement by Provatek, Tim Problem #1. Provatek defined the remaining Tim and Jake problems (TP, JP). All the problems involved friendly and target aircraft in the style of RP1.

In the experiment, software developers sought to solve each challenge problem twice, once using the experimental COP design methodology and Symlang programming language, and once with the popular Python general-purpose programming language coupled with traditional design methods as a control. Two of this report's co-authors acted as the software developers:

Matthew Evenson, a software developer with two decades of experience in software companies large and small who preferred to couple a traditional structured design method with Python, and

Jake Decker, a software developer embarking on his first job after graduating from college who preferred to couple a traditional data-flow design method with Python.

Co-author Timothy Fraser acted as coordinator, responsible for distributing challenge problems to the developers and reviewing their designs and solutions.

Provatek used the first two problems (TP1, TP2) as warm-up exercises to give the developers some experience with COP and Symlang. The challenge began in earnest with the coordinator presenting the next three problems (RP1, RP2, JP1) to the developers at the same time. The coordinator presented each subsequent problem to each developer as that developer demonstrated they had a correct solution to the previous one. The developers solved as many problems as they could over a period of roughly three calendar months.

To solve each problem in a given programming language, the developers:

1. produced a design document that met particular criteria for thoroughness,
2. presented the design to the coordinator and passed the coordinator's review,
3. developed a solution based on that design, and
4. demonstrated the correctness of that solution to the coordinator using the test rig.

Provatek measured two quantities for all solutions:

Effort in terms of the total number of person-minutes spent by the developer and coordinator on the above activities. Note that considering the total time spent on design and development together rather than attempting to distinguish between the two enabled Provatek to avoid the difficult problem of operationally defining which minutes of the developer's intellectual activity counted as design and which development.

Effort in terms of the proportion of a new solution's source lines of code that were added or changed by the developer in their effort to produce the new solution by extending an old one.

Provatek also manually analyzed the performance of the solutions to the most difficult challenge problem in terms of their big-O time and space algorithmic complexity. Provatek preferred this analytic approach to the alternative of repeatedly running the solutions and taking measurements of actual usage. The solutions spent most of their time waiting for the arrival of input and the problem goals generally required a specific condition to hold for 20 continuous seconds. The results of timed runs would have represented these wait times rather than actual solution performance.

Provatek anticipated that the developers would expend more effort the first time they solved a given problem than the second, as the first solution would bear the cost of understanding the problem and devising a strategy to solve it and the second solution would have the benefit of this thinking with none of the cost. To prevent this first-solution effort penalty from overshadowing the impact of the design method and programming language approaches, the developers attempted their solutions in a particular order that alternated approaches and ensured the COP/Symlang and traditional/Python approaches each had an equal number of first implementations. Table 4 shows this alternation for the first two problems.

Ultimately, the developers solved enough problems to present measurements for 14 challenge problems (RP1 through TP13) with a Symlang and Python solution from each of the two developers arranged in a pattern with the following properties:

- There are an equal number of solutions from each developer,
- each developer used Symlang first and Python first an equal number of times, and
- the number of Jake-Symlang-first, Matt-Symlang-first, Jake-Python-first, and Matt-Python-first solutions are all equal.

Provatek developed a test rig to support the experiment. Figure 2B contains a diagram of its architecture. It had three major components, all simulators:

- a world simulator that simulated the movement of friendly and target aircraft and the track production of sensors,
- a network transport simulator that simulated the unreliable radio network and its habit of sometimes dropping messages, and
- the developer's solution to the problem that simulated the operation of the data distributor component aboard each friendly aircraft.

Each problem required developers to create data distributor logic sufficient to reach its goal condition and to simulate that logic in a solution component that interoperated with the world and network transport simulators.

Table 4 - One cycle of the approach alternation pattern.

Problem	Developer Jake	Developer Matt
RP1	COP/Symlang first, then Data flow/Python.	Structured/Python first, then COP/Symlang.
RP2	Data flow/Python first, then COP/Symlang.	COP/Symlang first, then Structured/Python.

4.0 RESULTS AND DISCUSSION

The following subsections present and discuss the results of TA3 Provatek's original IDAS evaluation effort (Subsection 4.1) and those of the final evaluation it accomplished after the May-2021 IDAS program revision (Subsection 4.2).

4.1 The original IDAS effort

TA3 Provatek began its IDAS effort with a survey of the technical approaches of the TA1 research teams. TA3 Provatek completed this survey in July 2020 and used its findings to advise TA2 CMU/SEI on the kinds of requirements changes that would be relevant to each team's technical approach in evaluations. This subsection presents the results of the survey. It refers to the functionality the teams expected to provide in the first IDAS evaluation: Test Evaluation 1. It uses the IDAS term *concretization* to describe the decisions developers make that fix details of a program. Concretizations include decisions that are fundamental and necessary to solve the problem and those that are unimportant choices between equally good options. Figure 2 contains diagrams of the semi-automated developer workflows for each of the TA1 approaches. The following subsection provide succinct summaries of these approaches as they stood at the time of the survey.

4.1.1 TA1 Apogee

Designers record initial concretizations based on customer requirements in a *SymLang program*; the designer refines it with further concretizations until it is complete enough to translate into a shippable Java executable.

SymLang encourages designers to:

- view problems as processes that loop indefinitely, consuming input, changing their internal state, and producing output, and to
- view solutions as controllers that examine that state on each iteration and adjust it to maintain particular desired properties.

SymLang's most distinguishing feature is its DoOne/DoK dispatch mechanism.

In Test Evaluation #1, concretization was to be manual, aided by automated stability analysis to show that controller adjustments did not become increasingly extreme.

Apogee was developing a design methodology at the time of Provatek's survey. TA3 believed it was based on abstraction and controllers. Apogee believed it transcended abstraction.

(The paragraph above represents an early snapshot of TA1 Apogee's status in July 2020. TA1 Apogee continued work on the IDAS program after the May 2021 revision and eventually arrived at a design methodology they named Controller-Oriented Programming.)

4.1.2 TA1 Grammatech

Concretizations accumulate in the *Argot*: a tree of constraints, partial programs with types and assertions, and references to all past implementations.

How concretizations might have occurred in Test Evaluation 1:

- Manual updates to the Argot tree, its constraints, and partial programs.
- Automatically find patches by mutating program along failed test trace. Use mutations whose structure is found in other parts of the program.
- Automatically type programs by trying a series of potential typings sorted by statistical popularity until you find one that passes your type checker.
- Synthesis of tests via property-based testing.
- Semi-automated Eclipse-style source-level refactoring transformations.
- Automated source-level merge conflict resolution.

4.1.3 TA1 Perspecta Labs

Designers manually create an *Intent Specification*: an architecture with typed interfaces and requires/ensures assertions in domain-specific abstractions.

How further concretizations might have accumulated as *Candidate Programs* in Test Evaluation 1:

- Automation uses an (initial) brute-force SMT-based search to attempt to find a combination of program pieces that matches the structure and constraints of each component in the intent specification architecture.
- It draws program pieces from a domain-specific corpus. Each is annotated with the same kind of types and assertions as in the Intent Specification.
- If it finds no matching combination, the automation prompts the designer to implement matching pieces manually and add them to the corpus.

4.1.4 TA1 Vanderbilt

Concretizations accumulate in a version-controlled append-only database of *Models.* There are five interlinked models: Domain, Objectives-Intent-Constraints (OIC), Application, Synthesis, and Target, each with its own job.

How concretization might have occurred in Test Evaluation 1:

- Manual update of domain, customer requirement, and target execution environment details encoded in the above models.
- Semi-automated synthesis of code via Kestrel's APT toolkit.
- Automated generation of glue code, skeleton code, and/or hierarchical state machine code from models.
- Manual import of code from existing libraries.
- Manual implementation of code not covered by the above methods.

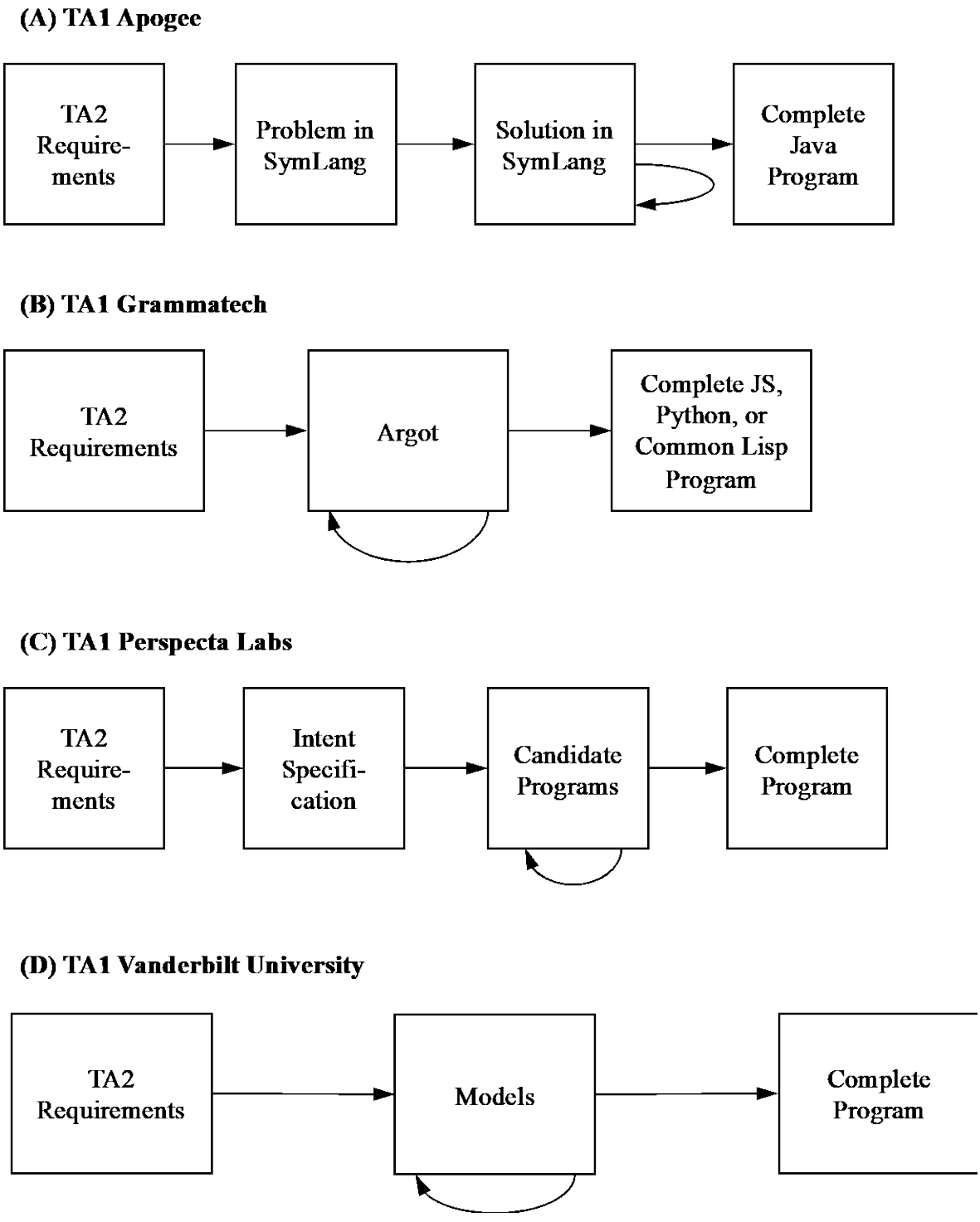

Figure 3 - Semi-automated TA1 research team developer workflows

4.2 The final COP/Symlang experiment

Table 5 lists the effort measurements for developer Jake Decker in terms of both person-minutes (PM) expended and the proportion of new or changed source Lines Of Code (LOC) in solutions. Each row presents the measurements for a given combination of problem and design/language approach. The "BASED ON" column identifies the earlier problem whose solution the developer extended to solve the current problem. First solutions have a "Y" in the column marked "FIRST?"; second solutions have an "N". Table 6 lists the equivalent effort measurements for developer Matthew Evenson.

The subsections below describe a series of statistical tests on this data. All statistical tests chose critical values sufficient to achieve 95% confidence.

4.2.1 Traditional/Python vs. COP/Symlang effort

Table 7 shows the measurements arranged in matched pairs where each pair couples the effort measurements for a given developer's traditional/Python and COP/Symlang efforts on the same problem. Each pair is thus a contest between Python and Symlang with the same developer on the same problem. There are no pairs that mix developers. There are no pairs that mix problems. There are 28 pairs. 14 are from developer Jake Decker. 14 are from developer Matthew Evenson. Half of each developer's pairs give Python the disadvantage of being used for the first solution. The other half give that disadvantage to Symlang.

The "CHANGE PM" column compares the effort required with Python to the effort required with Symlang in terms of person-minutes. This column uses the Python effort as the basis for comparison, with negative numbers representing cases where Symlang took less effort and positive numbers representing cases where Python took less effort. The "CHANGE LOC" column compares effort in terms of the proportion of new or changed lines of code. It is a simple difference with negative numbers representing cases where the Symlang solution had fewer added or changed lines of code and positive numbers representing cases where the Python solution had fewer.

Table 8 and Table 9 each show a matched pairs experiment to test the hypothesis that the mean effort required for the COP/Symlang approach is less than that of traditional/Python approach. Table 8 considers effort in terms of person-minutes expended; Table 9 considers effort in terms of added or changed LOC.

Out of concern that our 28 pairs may be too few to properly support a matched pairs experiment, Provatek also ran Wilcoxon signed ranks tests (a method capable of drawing conclusions from a small number of pairs.) Table 10 and Table 11 present Wilcoxon signed ranks tests of the hypotheses that the median effort required for COP/Symlang solutions is less than that of traditional/Python solutions in terms of person-minutes and added or changed LOC, respectively.

None of the four tests produced sufficient evidence to reject their null hypothesis that the mean or median effort required by the traditional/Python control approach and the COP/Symlang experimental approach are equal.

4.2.2 First-solution vs. second-solution effort

During the experiment, the developers carefully alternated the language they used for their first implementations to mitigate the impact of first-implementation disadvantage on our Python-vs-Symlang effort comparison. This alternation can also be used to mitigate

the impact of Python-vs-Symlang language choice on a comparison of the effort required for first and second implementations. This alternate mitigation enables tests of the hypothesis that the mean effort required for first implementations is greater than that required for second implementations. Although this test is not directly relevant to the traditional/Python vs. COP/Symlang comparison, it is relevant to Provatek's general mission to research methods of comparing programming languages.

Table 12 shows our measurements arranged in matched pairs where each pair couples the effort measurements for a given developer's first and second implementations of the same problem. Table 13 presents a matched pairs experiment that considers effort in terms of person-minutes expended. Table 14 presents a matched pairs experiment that considers effort in terms of added or changed LOC. Table 15 and Table 16 present Wilcoxon signed ranks tests for similar hypotheses.

The Wilcoxon signed ranks test in Table 15 provides evidence that the median effort for second implementations in terms of person-minutes expended is indeed lower. The corresponding matched pairs experiment in Table 13 agrees, estimating that the second implementation reduces mean person-minutes of effort by roughly 25%. However, LOC-oriented tests in Table 14 and Table 16 do not provide sufficient evidence to reject their null hypotheses that the mean or median efforts of first and second implementation in terms of added or changed LOC are the same.

Table 17 and Table 18 test for linear correlation between the person-minute-based and LOC-based effort differences observed in the Python-vs-Symlang and first-vs-second-implementation experiments, respectively. Neither test produced sufficient evidence to reject the null hypothesis that the numbers representing the two notions of effort are uncorrelated.

4.2.3 Traditional/Python vs. COP/Symlang time and space complexity

Table 19 presents the results of a manual big-O analysis of the time and space usage of all four solutions to TP11, the most difficult problem in terms of person-minutes required. Although the table shows differences between developers, it shows no differences between the Python and Symlang solutions produced by the same developer.

Table 5 - Measurements for developer Jake Decker.

PROBLEM	FIRST?	DESIGN	LANG-UAGE	BASED ON	DESIGN (pm)	IMPL (pm)	TOTAL (pm)	% NEW / MOD LOC
RP1	N	Data Flow	Python	TP2	125	168	293	36.5%
RP1	Y	COP	Symlang	TP2	155	272	427	28.9%
RP2	Y	Data Flow	Python	RP1	31	47	78	12.8%
RP2	N	COP	Symlang	RP1	39	26	65	7.1%
JP1	N	Data Flow	Python	RP2	31	47	78	27.2%
JP1	Y	COP	Symlang	RP2	39	26	65	29.7%
TP3	Y	Data Flow	Python	JP1	9	14	23	4.5%
TP3	N	COP	Symlang	JP1	8	5	13	6.4%
TP4	N	Data Flow	Python	TP3	18	31	49	19.9%
TP4	Y	COP	Symlang	TP3	36	87	123	10.4%
TP5	Y	Data Flow	Python	TP4	10	7	17	1.5%
TP5	N	COP	Symlang	TP4	4	10	14	2.2%
TP6	N	Data Flow	Python	RP1	3	7	10	0.5%
TP6	Y	COP	Symlang	RP1	10	7	17	0.0%
TP7	Y	Data Flow	Python	JP1	26	12	38	8.1%
TP7	N	COP	Symlang	JP1	6	9	15	7.6%
TP8	N	Data Flow	Python	TP6	3	11	14	0.0%
TP8	Y	COP	Symlang	TP6	14	4	18	0.2%
TP9	Y	Data Flow	Python	TP8	13	13	26	2.5%
TP9	N	COP	Symlang	TP8	3	21	24	1.1%
TP10	N	Data Flow	Python	TP9	5	13	18	3.4%
TP10	Y	COP	Symlang	TP9	7	20	27	3.6%
TP11	Y	Data Flow	Python	TP5	64	391	455	44.2%
TP11	N	COP	Symlang	TP5	19	135	154	21.4%
TP12	N	Data Flow	Python	TP11	3	1	4	1.1%
TP12	Y	COP	Symlang	TP11	4	1	5	0.4%
TP13	Y	Data Flow	Python	TP11	3	4	7	1.1%
TP13	N	COP	Symlang	TP11	3	1	4	0.3%
							Solution count:	28
							Python-first solution count:	7
							Symlang-first soltution count:	7

Table 6 - Measurements for developer Matthew Evenson

PROBLEM	FIRST?	DESIGN	LANG-UAGE	BASED ON	DESIGN (pm)	IMPL (pm)	TOTAL (pm)	% NEW / MOD LOC
RP1	Y	Structured	Python	TP2	94	173	267	26.0%
RP1	N	COP	Symlang	TP2	59	161	220	34.1%
RP2	N	Structured	Python	RP1	69	63	132	13.3%
RP2	Y	COP	Symlang	RP1	47	88	135	14.9%
JP1	Y	Structured	Python	RP2	53	28	81	20.4%
JP1	N	COP	Symlang	RP2	57	121	178	27.8%
TP3	N	Structured	Python	JP1	6	0	6	0.0%
TP3	Y	COP	Symlang	JP1	17	0	17	0.0%
TP4	Y	Structured	Python	RP2	29	4	33	0.8%
TP4	N	COP	Symlang	RP2	5	9	14	2.9%
TP5	N	Structured	Python	JP1	4	0	4	0.0%
TP5	Y	COP	Symlang	JP1	14	0	14	0.0%
TP6	Y	Structured	Python	RP1	24	0	24	0.0%
TP6	N	COP	Symlang	RP1	14	0	14	0.0%
TP7	N	Structured	Python	JP1	32	70	102	19.8%
TP7	Y	COP	Symlang	JP1	51	100	151	24.5%
TP8	Y	Structured	Python	RP1	14	30	44	2.6%
TP8	N	COP	Symlang	RP1	16	26	42	5.2%
TP9	N	Structured	Python	TP8	21	86	107	17.8%
TP9	Y	COP	Symlang	TP8	41	108	149	27.5%
TP10	Y	Structured	Python	TP9	18	43	61	8.4%
TP10	N	COP	Symlang	TP9	11	80	91	13.0%
TP11	N	Structured	Python	TP7	92	350	442	51.0%
TP11	Y	COP	Symlang	TP7	97	641	738	39.1%
TP12	Y	Structured	Python	TP11	23	121	144	1.1%
TP12	N	COP	Symlang	TP11	13	81	94	2.6%
TP13	N	Structured	Python	TP11	7	11	18	2.4%
TP13	Y	COP	Symlang	TP11	10	29	39	3.8%
							Solution count:	28
							Python-first solution count:	7
							Symlang-first soltution count:	7

Table 7 - Matched pairs for traditional/Python vs. COP/Symlang.

PROBLEM	DEVEL-OPER	PYTHON PM	SYM-LANG PM	CHANGE PM	PYTHON LOC	SYM-LANG LOC	CHANGE LOC
RP1	Jake	293	427	45.7%	36.5%	28.9%	-7.6%
RP2	Jake	78	65	-16.7%	12.8%	7.1%	-5.7%
JP1	Jake	78	65	-16.7%	27.2%	29.7%	2.6%
TP3	Jake	23	13	-43.5%	4.5%	6.4%	1.9%
TP4	Jake	49	123	151.0%	19.9%	10.4%	-9.5%
TP5	Jake	17	14	-17.6%	1.5%	2.2%	0.6%
TP6	Jake	10	17	70.0%	0.5%	0.0%	-0.5%
TP7	Jake	38	15	-60.5%	8.1%	7.6%	-0.5%
TP8	Jake	14	18	28.6%	0.0%	0.2%	0.2%
TP9	Jake	26	24	-7.7%	2.5%	1.1%	-1.4%
TP10	Jake	18	27	50.0%	3.4%	3.6%	0.2%
TP11	Jake	455	154	-66.2%	44.2%	21.4%	-22.7%
TP12	Jake	4	5	25.0%	1.1%	0.4%	-0.7%
TP13	Jake	7	4	-42.9%	1.1%	0.3%	-0.8%
RP1	Matt	267	220	-17.6%	26.0%	34.1%	8.1%
RP2	Matt	132	135	2.3%	13.3%	14.9%	1.6%
JP1	Matt	81	178	119.8%	20.4%	27.8%	7.4%
TP3	Matt	6	17	183.3%	0.0%	0.0%	0.0%
TP4	Matt	33	14	-57.6%	0.8%	2.9%	2.0%
TP5	Matt	4	14	250.0%	0.0%	0.0%	0.0%
TP6	Matt	24	14	-41.7%	0.0%	0.0%	0.0%
TP7	Matt	102	151	48.0%	19.8%	24.5%	4.6%
TP8	Matt	44	42	-4.5%	2.6%	5.2%	2.7%
TP9	Matt	107	149	39.3%	17.8%	27.5%	9.7%
TP10	Matt	61	91	49.2%	8.4%	13.0%	4.6%
TP11	Matt	442	738	67.0%	51.0%	39.1%	-11.9%
TP12	Matt	144	94	-34.7%	1.1%	2.6%	1.5%
TP13	Matt	18	39	116.7%	2.4%	3.8%	1.4%

Table 8 - Matched pairs experiment, tradtional/Python vs. COP/Symlang, person-minutes.

Matched Pairs Experiment for Python vs. Symlang implementation effort.

Null hypothesis: the mean effort for Traditional/Python implementations equals the mean effort for COP/Symlang implementations.

Alternate hypothesis: the mean efforts differ.

PROBLEM	DEVEL-OPER	PYTHON PM	SYM-LANG PM	CHANGE PM
RP1	Jake	293	427	45.7%
RP2	Jake	78	65	-16.7%
JP1	Jake	78	65	-16.7%
TP3	Jake	23	13	-43.5%
TP4	Jake	49	123	151.0%
TP5	Jake	17	14	-17.6%
TP6	Jake	10	17	70.0%
TP7	Jake	38	15	-60.5%
TP8	Jake	14	18	28.6%
TP9	Jake	26	24	-7.7%
TP10	Jake	18	27	50.0%
TP11	Jake	455	154	-66.2%
TP12	Jake	4	5	25.0%
TP13	Jake	7	4	-42.9%
RP1	Matt	267	220	-17.6%
RP2	Matt	132	135	2.3%
JP1	Matt	81	178	119.8%
TP3	Matt	6	17	183.3%
TP4	Matt	33	14	-57.6%
TP5	Matt	4	14	250.0%
TP6	Matt	24	14	-41.7%
TP7	Matt	102	151	48.0%
TP8	Matt	44	42	-4.5%
TP9	Matt	107	149	39.3%
TP10	Matt	61	91	49.2%
TP11	Matt	442	738	67.0%
TP12	Matt	144	94	-34.7%
TP13	Matt	18	39	116.7%
			Change mean:	29.2%
			Change stddev:	77.9%
			Number of pairs:	28
			Degrees of freedom:	27
			test statistic:	1.984
			critical value for 95% confidence from table:	2.052

Test statistic fails to exceed +- critical value. Fail to reject null hypothesis.

Experiment provides no evidence that one approach is better or worse than the other in terms of effort.

Table 9 - Matched pairs experiment, tradtional/Python vs. COP/Symlang, LOC.

Matched Pairs Experiment for Python vs. Symlang new/changed LOC.				
Null hypothesis: the mean added/changed LOC % for Traditional/Python				
	implementations equals the mean for COP/Symlang.			
Alternate hypothesis: the added/changed LOC % means differ.				
PROBLEM	DEVEL-OPER	PYTHON LOC	SYM-LANG LOC	CHANGE LOC
RP1	Jake	0.365	0.2894737	-7.6%
RP2	Jake	0.1280488	0.0712743	-5.7%
JP1	Jake	0.2716763	0.2973396	2.6%
TP3	Jake	0.0451977	0.0644628	1.9%
TP4	Jake	0.1989796	0.1041991	-9.5%
TP5	Jake	0.0152284	0.0216383	0.6%
TP6	Jake	0.0049751	0	-0.5%
TP7	Jake	0.0806452	0.0757576	-0.5%
TP8	Jake	0	0.0018832	0.2%
TP9	Jake	0.0247525	0.011215	-1.4%
TP10	Jake	0.0343137	0.0361011	0.2%
TP11	Jake	0.4415094	0.2140921	-22.7%
TP12	Jake	0.0113208	0.004065	-0.7%
TP13	Jake	0.011194	0.0027027	-0.8%
RP1	Matt	0.2604167	0.3411306	8.1%
RP2	Matt	0.1327801	0.1491228	1.6%
JP1	Matt	0.2035088	0.2779851	7.4%
TP3	Matt	0	0	0.0%
TP4	Matt	0.0082988	0.0285088	2.0%
TP5	Matt	0	0	0.0%
TP6	Matt	0	0	0.0%
TP7	Matt	0.1982507	0.244713	4.6%
TP8	Matt	0.0255474	0.0524272	2.7%
TP9	Matt	0.1775701	0.2749196	9.7%
TP10	Matt	0.083815	0.1301775	4.6%
TP11	Matt	0.5095785	0.390553	-11.9%
TP12	Matt	0.0114068	0.0263459	1.5%
TP13	Matt	0.0243446	0.0383315	1.4%
			Change mean:	-0.4%
			Change stddev:	6.5%
			Number of pairs:	28
			Degrees of freedom:	27
			test statistic:	-0.351
		critical value for 95% confidence from table:		2.052
Test statistic fails to exceed +- critical value. Fail to reject null hypothesis.				
Experiment provides no evidence that one approach is better or worse				
than the other in terms of added/changed LOC %.				

Table 10 - Wilcoxon singed ranks test, Python vs. Symlang, person-minutes.

Wilcoxon signed ranks test for Python vs. Symlang implementation, all problems, both developers.						
Null hypothesis: the median effort for Traditional/Python implementations equals the median effort for COP/Symlang implementations.						
Alternate hypothesis: the median efforts differ.						
PROBLEM	DEVEL-OPER	CHANGE PM	ABS CHANGE	RANK	NEG RANKS	POS RANKS
RP2	Matt	2.3%	2.3%	1		1
TP8	Matt	-4.5%	4.5%	2	2	
TP9	Jake	-7.7%	7.7%	3	3	
RP2	Jake	-16.7%	16.7%	4.5	4.5	
JP1	Jake	-16.7%	16.7%	4.5	4.5	
RP1	Matt	-17.6%	17.6%	6.5	6.5	
TP5	Jake	-17.6%	17.6%	6.5	6.5	
TP12	Jake	25.0%	25.0%	8		8
TP8	Jake	28.6%	28.6%	9		9
TP12	Matt	-34.7%	34.7%	10	10	
TP9	Matt	39.3%	39.3%	11		11
TP6	Matt	-41.7%	41.7%	12	12	
TP13	Jake	-42.9%	42.9%	13	13	
TP3	Jake	-43.5%	43.5%	14	14	
RP1	Jake	45.7%	45.7%	15		15
TP7	Matt	48.0%	48.0%	16		16
TP10	Matt	49.2%	49.2%	17		17
TP10	Jake	50.0%	50.0%	18		18
TP4	Matt	-57.6%	57.6%	19	19	
TP7	Jake	-60.5%	60.5%	20	20	
TP11	Jake	-66.2%	66.2%	21	21	
TP11	Matt	67.0%	67.0%	22		22
TP6	Jake	70.0%	70.0%	23		23
TP13	Matt	116.7%	116.7%	24		24
JP1	Matt	119.8%	119.8%	25		25
TP4	Jake	151.0%	151.0%	26		26
TP3	Matt	183.3%	183.3%	27		27
TP5	Matt	250.0%	250.0%	28		28
				Sum:	136	270
				Test statistic:	136	
				Number of pairs:	28	
		Critical value for 95% confidence from table:			117	
Test statistic lies beyond critical value. This experiment provides no evidence to contradict the null hypothesis that neither Python nor Symlang are better than the other in terms of effort.						

Table 11 - Wilcoxon signed ranks test, Python vs. Symlang, LOC.

Wilcoxon signed ranks test for Python vs. Symlang implementation, all problems, both developers.						
Null hypothesis: the median added/changed LOC % for Traditional/Python implementations equals the median for COP/Symlang implementations.						
Alternate hypothesis: the medians differ.						
PROBLEM	DEVEL-OPER	CHANGE LOC	ABS CHANGE	RANK	NEG RANKS	POS RANKS
TP3	Matt	0.0%	0.0%			
TP5	Matt	0.0%	0.0%			
TP6	Matt	0.0%	0.0%			
TP10	Jake	0.2%	0.2%	1.5		1.5
TP8	Jake	0.2%	0.2%	1.5		1.5
TP7	Jake	-0.5%	0.5%	3.5	3.5	
TP6	Jake	-0.5%	0.5%	3.5	3.5	
TP5	Jake	0.6%	0.6%	5		5
TP12	Jake	-0.7%	0.7%	6	6	
TP13	Jake	-0.8%	0.8%	7	7	
TP9	Jake	-1.4%	1.4%	8.5	8.5	
TP13	Matt	1.4%	1.4%	8.5		8.5
TP12	Matt	1.5%	1.5%	10		10
RP2	Matt	1.6%	1.6%	11		11
TP3	Jake	1.9%	1.9%	12		12
TP4	Matt	2.0%	2.0%	13		13
JP1	Jake	2.6%	2.6%	14		14
TP8	Matt	2.7%	2.7%	15		15
TP10	Matt	4.6%	4.6%	16.5		16.5
TP7	Matt	4.6%	4.6%	16.5		16.5
RP2	Jake	-5.7%	5.7%	18	18	
JP1	Matt	7.4%	7.4%	19		19
RP1	Jake	-7.6%	7.6%	20	20	
RP1	Matt	8.1%	8.1%	21		21
TP4	Jake	-9.5%	9.5%	22	22	
TP9	Matt	9.7%	9.7%	23		23
TP11	Matt	-11.9%	11.9%	24	24	
TP11	Jake	-22.7%	22.7%	25	25	
				Sum:	137.5	187.5
				Test statistic:	137.5	
				Number of pairs:	25	
		Critical value for 95% confidence from table:			90	
Test statistic lies beyond critical value. This experiment provides no evidence to contradict the null hypothesis that neither Python nor Symlang are better than the other in terms of % added/changed LOC.						

Table 12 - Matched pairs for 1st vs. 2nd solutions.

PROBLEM	DEVEL-OPER	1ST PM	2ND PM	CHANGE PM	1ST LOC	2ND LOC	CHANGE LOC
RP1	Jake	427	293	-31.4%	28.9%	36.5%	7.6%
RP2	Jake	78	65	-16.7%	12.8%	7.1%	-5.7%
JP1	Jake	65	78	20.0%	29.7%	27.2%	-2.6%
TP3	Jake	23	13	-43.5%	4.5%	6.4%	1.9%
TP4	Jake	123	49	-60.2%	10.4%	19.9%	9.5%
TP5	Jake	17	14	-17.6%	1.5%	2.2%	0.6%
TP6	Jake	17	10	-41.2%	0.0%	0.5%	0.5%
TP7	Jake	38	15	-60.5%	8.1%	7.6%	-0.5%
TP8	Jake	18	14	-22.2%	0.2%	0.0%	-0.2%
TP9	Jake	26	24	-7.7%	2.5%	1.1%	-1.4%
TP10	Jake	27	18	-33.3%	3.6%	3.4%	-0.2%
TP11	Jake	455	154	-66.2%	44.2%	21.4%	-22.7%
TP12	Jake	5	4	-20.0%	0.4%	1.1%	0.7%
TP13	Jake	7	4	-42.9%	1.1%	0.3%	-0.8%
RP1	Matt	267	220	-17.6%	26.0%	34.1%	8.1%
RP2	Matt	135	132	-2.2%	14.9%	13.3%	-1.6%
JP1	Matt	81	178	119.8%	20.4%	27.8%	7.4%
TP3	Matt	17	6	-64.7%	0.0%	0.0%	0.0%
TP4	Matt	33	14	-57.6%	0.8%	2.9%	2.0%
TP5	Matt	14	4	-71.4%	0.0%	0.0%	0.0%
TP6	Matt	24	14	-41.7%	0.0%	0.0%	0.0%
TP7	Matt	151	102	-32.5%	24.5%	19.8%	-4.6%
TP8	Matt	44	42	-4.5%	2.6%	5.2%	2.7%
TP9	Matt	149	107	-28.2%	27.5%	17.8%	-9.7%
TP10	Matt	61	91	49.2%	8.4%	13.0%	4.6%
TP11	Matt	738	442	-40.1%	39.1%	51.0%	11.9%
TP12	Matt	144	94	-34.7%	1.1%	2.6%	1.5%
TP13	Matt	39	18	-53.8%	3.8%	2.4%	-1.4%

Table 13 - Matched pairs experiment, 1st vs. 2nd solutions, person-minutes.

Matched Pairs Experiment for first vs. second implementation effort.				
Null hypothesis: the mean effort for first implementations equals the mean effort				
	for second implementations.			
Alternate hypothesis: the mean efforts differ.				
PROBLEM	DEVEL-OPER	1ST PM	2ND PM	CHANGE PM
RP1	Jake	427	293	-31.4%
RP2	Jake	78	65	-16.7%
JP1	Jake	65	78	20.0%
TP3	Jake	23	13	-43.5%
TP4	Jake	123	49	-60.2%
TP5	Jake	17	14	-17.6%
TP6	Jake	17	10	-41.2%
TP7	Jake	38	15	-60.5%
TP8	Jake	18	14	-22.2%
TP9	Jake	26	24	-7.7%
TP10	Jake	27	18	-33.3%
TP11	Jake	455	154	-66.2%
TP12	Jake	5	4	-20.0%
TP13	Jake	7	4	-42.9%
RP1	Matt	267	220	-17.6%
RP2	Matt	135	132	-2.2%
JP1	Matt	81	178	119.8%
TP3	Matt	17	6	-64.7%
TP4	Matt	33	14	-57.6%
TP5	Matt	14	4	-71.4%
TP6	Matt	24	14	-41.7%
TP7	Matt	151	102	-32.5%
TP8	Matt	44	42	-4.5%
TP9	Matt	149	107	-28.2%
TP10	Matt	61	91	49.2%
TP11	Matt	738	442	-40.1%
TP12	Matt	144	94	-34.7%
TP13	Matt	39	18	-53.8%
			Change mean:	-25.8%
			Change stddev:	39.2%
			Number of pairs:	28
			Degrees of freedom:	27
			test statistic:	-3.490
		critical value for 95% confidence from table:		2.052
Test statistic exceeds +- critical value. Reject null hypothesis.				
Experiment estimates second implementation take proportionally less				
effort as indicated by change mean with 95% confidence.				

Table 14 - Matched pairs experiment, 1st vs. 2nd solutions, LOC.

Matched Pairs Experiment for first vs. second LOC added or changed.					
Null hypothesis: the mean LOC added/changed for first implementations equals					
	the mean LOC added/changed for second implementations.				
Alternate hypothesis: the LOC added/changed means differ.					
PROBLEM	DEVEL-OPER	1ST LOC	2ND LOC	CHANGE LOC	
RP1	Jake	28.9%	36.5%	7.6%	
RP2	Jake	12.8%	7.1%	-5.7%	
JP1	Jake	29.7%	27.2%	-2.6%	
TP3	Jake	4.5%	6.4%	1.9%	
TP4	Jake	10.4%	19.9%	9.5%	
TP5	Jake	1.5%	2.2%	0.6%	
TP6	Jake	0.0%	0.5%	0.5%	
TP7	Jake	8.1%	7.6%	-0.5%	
TP8	Jake	0.2%	0.0%	-0.2%	
TP9	Jake	2.5%	1.1%	-1.4%	
TP10	Jake	3.6%	3.4%	-0.2%	
TP11	Jake	44.2%	21.4%	-22.7%	
TP12	Jake	0.4%	1.1%	0.7%	
TP13	Jake	1.1%	0.3%	-0.8%	
RP1	Matt	26.0%	34.1%	8.1%	
RP2	Matt	14.9%	13.3%	-1.6%	
JP1	Matt	20.4%	27.8%	7.4%	
TP3	Matt	0.0%	0.0%	0.0%	
TP4	Matt	0.8%	2.9%	2.0%	
TP5	Matt	0.0%	0.0%	0.0%	
TP6	Matt	0.0%	0.0%	0.0%	
TP7	Matt	24.5%	19.8%	-4.6%	
TP8	Matt	2.6%	5.2%	2.7%	
TP9	Matt	27.5%	17.8%	-9.7%	
TP10	Matt	8.4%	13.0%	4.6%	
TP11	Matt	39.1%	51.0%	11.9%	
TP12	Matt	1.1%	2.6%	1.5%	
TP13	Matt	3.8%	2.4%	-1.4%	
			Change mean:	0.3%	
			Change stddev:	6.5%	
			Number of pairs:	28	
			Degrees of freedom:	27	
			test statistic:	0.223	
		critical value for 95% confidence from table:		2.052	
Test statistic fails to exceed +- critical value. Fail to reject null hypothesis.					
Experiment provides no evidence that first implementations are better					
or worse than second implementaitons in terms of added/changed LOC %.					

Table 15 - Wilcoxon signed ranks test, 1st vs. 2nd solutions, person-minutes.

Wilcoxon signed ranks test for first vs. second implementation, all problems, both developers.						
Null hypothesis: the median effort for first implementations equals the median effort for second implementations.						
Alternate hypothesis: the median efforts differ.						
PROBLEM	DEVEL-OPER	CHANGE PM	ABS CHANGE	RANK	NEG RANKS	POS RANKS
RP2	Matt	-2.2%	2.2%	1	1	
TP8	Matt	-4.5%	4.5%	2	2	
TP9	Jake	-7.7%	7.7%	3	3	
RP2	Jake	-16.7%	16.7%	4	4	
RP1	Matt	-17.6%	17.6%	5.5	5.5	
TP5	Jake	-17.6%	17.6%	5.5	5.5	
JP1	Jake	20.0%	20.0%	7.5		7.5
TP12	Jake	-20.0%	20.0%	7.5	7.5	
TP8	Jake	-22.2%	22.2%	9	9	
TP9	Matt	-28.2%	28.2%	10	10	
RP1	Jake	-31.4%	31.4%	11	11	
TP7	Matt	-32.5%	32.5%	12	12	
TP10	Jake	-33.3%	33.3%	13	13	
TP12	Matt	-34.7%	34.7%	14	14	
TP11	Matt	-40.1%	40.1%	15	15	
TP6	Jake	-41.2%	41.2%	16	16	
TP6	Matt	-41.7%	41.7%	17	17	
TP13	Jake	-42.9%	42.9%	18	18	
TP3	Jake	-43.5%	43.5%	19	19	
TP10	Matt	49.2%	49.2%	20		20
TP13	Matt	-53.8%	53.8%	21	21	
TP4	Matt	-57.6%	57.6%	22	22	
TP4	Jake	-60.2%	60.2%	23	23	
TP7	Jake	-60.5%	60.5%	24	24	
TP3	Matt	-64.7%	64.7%	25	25	
TP11	Jake	-66.2%	66.2%	26	26	
TP5	Matt	-71.4%	71.4%	27	27	
JP1	Matt	119.8%	119.8%	28		28
				Sum:	350.5	55.5
				Test statistic:	55.5	
				Number of pairs:	28	
			Critical value for 95% confidence from table:		117	
Test statistic lies within critical value. Reject null hypothesis.						
This experiment provides evidence that second implementations take less effort than first implementations, at 95% confidence.						

Table 16 - Wilcoxon signed ranks test, 1st vs. 2nd solutions, LOC.

Wilcoxon signed ranks test for first vs. second implementation, all problems, both developers.						
Null hypothesis: the median % new/changed LOC for first implementations equals the median for second implementations.						
Alternate hypothesis: the medians differ.						
PROBLEM	DEVEL-OPER	CHANGE LOC	ABS CHANGE	RANK	NEG RANKS	POS RANKS
TP3	Matt	0.0%	0.0%			
TP5	Matt	0.0%	0.0%			
TP6	Matt	0.0%	0.0%			
TP10	Jake	-0.2%	0.2%	1.5	1.5	
TP8	Jake	-0.2%	0.2%	1.5	1.5	
TP7	Jake	-0.5%	0.5%	3.5	3.5	
TP6	Jake	0.5%	0.5%	3.5		3.5
TP5	Jake	0.6%	0.6%	5		5
TP12	Jake	0.7%	0.7%	6		6
TP13	Jake	-0.8%	0.8%	7	7	
TP9	Jake	-1.4%	1.4%	8.5	8.5	
TP13	Matt	-1.4%	1.4%	8.5	8.5	
TP12	Matt	1.5%	1.5%	10		10
RP2	Matt	-1.6%	1.6%	11	11	
TP3	Jake	1.9%	1.9%	12		12
TP4	Matt	2.0%	2.0%	13		13
JP1	Jake	-2.6%	2.6%	14	14	
TP8	Matt	2.7%	2.7%	15		15
TP10	Matt	4.6%	4.6%	16.5		16.5
TP7	Matt	-4.6%	4.6%	16.5	16.5	
RP2	Jake	-5.7%	5.7%	18	18	
JP1	Matt	7.4%	7.4%	19		19
RP1	Jake	7.6%	7.6%	20		20
RP1	Matt	8.1%	8.1%	21		21
TP4	Jake	9.5%	9.5%	22		22
TP9	Matt	-9.7%	9.7%	23	23	
TP11	Matt	11.9%	11.9%	24		24
TP11	Jake	-22.7%	22.7%	25	25	
				Sum:	138	187
				Test statistic:	138	
				Number of pairs:	25	
		Critical value for 95% confidence from table:			90	
Test statistic lies beyond critical value. This experiment provides no evidence to contradict the null hypothesis that neither first nor second implementations are better than the other in terms of % new/added LOC.						

Table 17 - Linear correlation test between person-minutes and LOC, Python vs. Symlang.

Language effort difference vs. changed LOC difference correlation computation for all problems, both developers.						
Null hypothesis: the % effort differences and % changed LOC differences in the Python vs. Symlang experiments are not correlated.						
Alternate hypothesis: there is some linear correlation.						
PROBLEM	DEVEL-OPER	CHANGE PM	CHANGE LOC			
RP1	Jake	45.7%	-7.6%			
RP2	Jake	-16.7%	-5.7%			
JP1	Jake	-16.7%	2.6%			
TP3	Jake	-43.5%	1.9%			
TP4	Jake	151.0%	-9.5%			
TP5	Jake	-17.6%	0.6%			
TP6	Jake	70.0%	-0.5%			
TP7	Jake	-60.5%	-0.5%			
TP8	Jake	28.6%	0.2%			
TP9	Jake	-7.7%	-1.4%			
TP10	Jake	50.0%	0.2%			
TP11	Jake	-66.2%	-22.7%			
TP12	Jake	25.0%	-0.7%			
TP13	Jake	-42.9%	-0.8%			
RP1	Matt	-17.6%	8.1%			
RP2	Matt	2.3%	1.6%			
JP1	Matt	119.8%	7.4%			
TP3	Matt	183.3%	0.0%			
TP4	Matt	-57.6%	2.0%			
TP5	Matt	250.0%	0.0%			
TP6	Matt	-41.7%	0.0%			
TP7	Matt	48.0%	4.6%			
TP8	Matt	-4.5%	2.7%			
TP9	Matt	39.3%	9.7%			
TP10	Matt	49.2%	4.6%			
TP11	Matt	67.0%	-11.9%			
TP12	Matt	-34.7%	1.5%			
TP13	Matt	116.7%	1.4%			
				Number of pairs:		28
				Linear correlation coefficient:		0.062
				95% confidence critical value for 25 pairs from table:		0.396
				95% confidence critical value for 30 pairs from table:		0.361
Linear correlation coefficient is less than the critical value. This experiment produced no evidence to contradict the null hypothesis that there is no correlation, and thus that effort change and LOC change are good indicators of each other.						

Table 18 - Linear correlation test between person-minutes and LOC, 1st vs. 2nd.

Order effort difference vs. changed LOC difference correlation computation for all problems, both developers.						
Null hypothesis: the % effort differences and % changed LOC differences in the first vs. second experiments are not correlated.						
Alternate hypothesis: there is some linear correlation.						
PROBLEM	DEVEL-OPER	CHANGE PM	CHANGE LOC			
RP1	Jake	-31.4%	7.6%			
RP2	Jake	-16.7%	-5.7%			
JP1	Jake	20.0%	-2.6%			
TP3	Jake	-43.5%	1.9%			
TP4	Jake	-60.2%	9.5%			
TP5	Jake	-17.6%	0.6%			
TP6	Jake	-41.2%	0.5%			
TP7	Jake	-60.5%	-0.5%			
TP8	Jake	-22.2%	-0.2%			
TP9	Jake	-7.7%	-1.4%			
TP10	Jake	-33.3%	-0.2%			
TP11	Jake	-66.2%	-22.7%			
TP12	Jake	-20.0%	0.7%			
TP13	Jake	-42.9%	-0.8%			
RP1	Matt	-17.6%	8.1%			
RP2	Matt	-2.2%	-1.6%			
JP1	Matt	119.8%	7.4%			
TP3	Matt	-64.7%	0.0%			
TP4	Matt	-57.6%	2.0%			
TP5	Matt	-71.4%	0.0%			
TP6	Matt	-41.7%	0.0%			
TP7	Matt	-32.5%	-4.6%			
TP8	Matt	-4.5%	2.7%			
TP9	Matt	-28.2%	-9.7%			
TP10	Matt	49.2%	4.6%			
TP11	Matt	-40.1%	11.9%			
TP12	Matt	-34.7%	1.5%			
TP13	Matt	-53.8%	-1.4%			
				Number of pairs:	28	
				Linear correlation coefficient:	0.251	
				95% confidence critical value for 25 pairs from table:	0.396	
				95% confidence critical value for 30 pairs from table:	0.361	
Linear correlation coefficient is less than the critical value. This experiment produced no evidence to contradict the null hypothesis that there is no correlation, and thus that effort change and LOC change are good indicators of each other.						

Table 19 - time and space algorithmic complexity analysis for TP11.

	Traditional/Python	COP/Symlang
Jake time	$O(n^6)$	$O(n^6)$
Jake space	$O(2^n)$	$O(2^n)$
Matt time	$O(n^6)$	$O(n^6)$
Matt space	$O(n^6)$	$O(n^6)$

Where n is proportional to the total number of friendly and target aircraft in the scenario. (TP11 had only 6 aircraft, enabling these algorithms to work despite their complexity.)

4.2.4 Discussion

The experiment produced no evidence that either the traditional/Python approach or the COP/Symlang approach had an advantage over the other in terms of the effort required to extend solutions or in terms of the time/space efficiency of those solutions at runtime. Before drawing its conclusions, Provatek re-examined the experiment's design seeking flaws that might have prevented the COP/Symlang approach from performing better. A discussion of this re-examination follows.

One flaw hypothesis was that the two warm-up problems did not provide the developers with sufficient familiarity with COP and Symlang to use them to their full potential, and that COP/Symlang might have fared better on the later problems than they had on the early ones. Table 20 and Table 21 present Wilcoxon signed ranks tests for the person-minute and LOC effort hypotheses using only the last six problems in the series. These tests do not support the insufficient warm-up hypothesis; like the earlier tests of the full data set, they indicate no benefit from either approach.

Another flaw hypothesis was that one developer's COP/Symlang results were good but were overridden by bad results by the other developer, either because one made a better initial COP design than the other, or because one consistently made better use of Symlang's features than the other. There were differences between developers:

- Both developers had decided on tracks as the key element of the problems, but developer Jake Decker's designs partitioned tracks based on their accuracy, while developer Matt Evenson's designs partitioned tracks based on their source.
- Jake Decker's solutions were on average 50% Symlang code and 50% Java. Matt Evenson's were 23% Symlang and 77% Java on average.

Table 22, Table 23, Table 24, and Table 25 present Wilcoxon signed ranks tests that repeat the earlier person-minute and LOC effort tests separately for the two developers. These tests do not support the bad-overrides-good hypothesis; like the earlier tests with both developers together, they indicate no benefit from either approach.

Although Provatek was able to reject those two explanations with additional statistical analyses, two plausible explanations remained that were not so easily dismissed. First, Provatek's analyses used critical values sufficient only for 95% confidence. It could be that COP/Symlang is indeed an improvement over Python coupled with traditional design methods and the results reported here were simply the product of bad luck. A similar experiment with more developers or more challenge problems might produce more matched pairs of measurements and deliver a different result with higher confidence.

Second, it could be that the results reported here are accurate but misleading. It could be that COP and Symlang are like logic programming and Datalog: a programming paradigm and language optimized for and suited to a particular kind of problem. The Rev problems proposed by the Government and Apogee Research may not have been examples of that kind of problem. Provatek noted that their developers did not find an opportunity to use the "DoK" form of Symlang's "Do" operator that dispatches control to arbitrarily chosen functionally equivalent elements in a partition in their solutions. The absence of this form of dispatch may suggest that the Rev problems were not any more amenable to COP solutions than they were to traditional solutions.

The Government and Apogee Research originally hoped to use the Rev problems both as challenge problems for this experiment and as the basis for a demonstration of an operationally relevant use case. Although Provatek was not privy to the details of the use case, the authors imagine their choice of problem may have represented a necessary compromise between the conflicting needs of those two purposes and may have ultimately led to challenge problems that were outside of COP's ideal domain. A similar experiment with challenge problems more relevant to COP might demonstrate the superiority of COP/Symlang in that particular problem domain.

Table 20 - Wilcoxon signed ranks, Python vs. Symlang, person-minutes, later problems.

Wilcoxon signed ranks test for Python vs. Symlang implementation, later problems, both developers.						
Null hypothesis: the median effort for Traditional/Python implementations equals the median effort for COP/Symlang implementations.						
Alternate hypothesis: the median efforts differ.						
PROBLEM	DEVEL-OPER	CHANGE PM	ABS CHANGE	RANK	NEG RANKS	POS RANKS
TP8	Matt	-4.5%	4.5%	1	1	
TP9	Jake	-7.7%	7.7%	2	2	
TP12	Jake	25.0%	25.0%	3		3
TP8	Jake	28.6%	28.6%	4		4
TP12	Matt	-34.7%	34.7%	5	5	
TP9	Matt	39.3%	39.3%	6		6
TP13	Jake	-42.9%	42.9%	7	7	
TP10	Matt	49.2%	49.2%	8		8
TP10	Jake	50.0%	50.0%	9		9
TP11	Jake	-66.2%	66.2%	10	10	
TP11	Matt	67.0%	67.0%	11		11
TP13	Matt	116.7%	116.7%	12		12
				Sum:	25	53
				Test statistic:	25	
				Number of pairs:	12	
		Critical value for 95% confidence from table:			14	
Test statistic lies beyond critical value. This experiment provides no evidence to contradict the null hypothesis that neither Python nor Symlang are better than the other in terms of effort.						

Table 21 - Wilcoxon signed ranks, Python vs. Symlang, LOC, later problems.

Wilcoxon signed ranks test for Python vs. Symlang implementation, later problems, both developers.						
Null hypothesis: the median added/changed LOC % for Traditional/Python implementations equals the median for COP/Symlang implementations.						
Alternate hypothesis: the medians differ.						
PROBLEM	DEVEL-OPER	CHANGE LOC	ABS CHANGE	RANK	NEG RANKS	POS RANKS
TP10	Jake	0.2%	0.2%	1.5		1.5
TP8	Jake	0.2%	0.2%	1.5		1.5
TP12	Jake	-0.7%	0.7%	3	3	
TP13	Jake	-0.8%	0.8%	4	4	
TP9	Jake	-1.4%	1.4%	5.5	5.5	
TP13	Matt	1.4%	1.4%	5.5		5.5
TP12	Matt	1.5%	1.5%	7		7
TP8	Matt	2.7%	2.7%	8		8
TP10	Matt	4.6%	4.6%	9		9
TP9	Matt	9.7%	9.7%	10		10
TP11	Matt	-11.9%	11.9%	11	11	
TP11	Jake	-22.7%	22.7%	12	12	
				Sum:	35.5	42.5
				Test statistic:	35.5	
				Number of pairs:	12	
		Critical value for 95% confidence from table:			14	
Test statistic lies beyond critical value. This experiment provides no evidence to contradict the null hypothesis that neither Python nor Symlang are better than the other in terms of % added/changed LOC.						

Table 22 - Wilcoxon signed ranks, Python vs. Symlang, person-minutes, Jake Decker.

Wilcoxon signed ranks test for Python vs. Symlang implementation, all problems, only developer Jake.						
Null hypothesis: the median effort for Traditional/Python implementations equals the median effort for COP/Symlang implementations.						
Alternate hypothesis: the median efforts differ.						
PROBLEM	DEVEL-OPER	CHANGE PM	ABS CHANGE	RANK	NEG RANKS	POS RANKS
TP9	Jake	-7.7%	7.7%	1	1	
RP2	Jake	-16.7%	16.7%	2.5	2.5	
JP1	Jake	-16.7%	16.7%	2.5	2.5	
TP5	Jake	-17.6%	17.6%	4	4	
TP12	Jake	25.0%	25.0%	5		5
TP8	Jake	28.6%	28.6%	6		6
TP13	Jake	-42.9%	42.9%	7	7	
TP3	Jake	-43.5%	43.5%	8	8	
RP1	Jake	45.7%	45.7%	9		9
TP10	Jake	50.0%	50.0%	10		10
TP7	Jake	-60.5%	60.5%	11	11	
TP11	Jake	-66.2%	66.2%	12	12	
TP6	Jake	70.0%	70.0%	13		13
TP4	Jake	151.0%	151.0%	14		14
				Sum:	48	57
				Test statistic:	48	
				Number of pairs:	14	
			Critical value for 95% confidence from table:		21	
Test statistic lies beyond critical value. This experiment provides no evidence to contradict the null hypothesis that neither Python nor Symlang are better than the other in terms of effort.						

Table 23 - Wilcoxon signed ranks, Python vs. Symlang, person-minutes, Matt Evenson.

Wilcoxon signed ranks test for Python vs. Symlang implementation, all problems, only developer Matt.						
Null hypothesis: the median effort for Traditional/Python implementations equals the median effort for COP/Symlang implementations.						
Alternate hypothesis: the median efforts differ.						
PROBLEM	DEVEL-OPER	CHANGE PM	ABS CHANGE	RANK	NEG RANKS	POS RANKS
RP2	Matt	2.3%	2.3%	1		1
TP8	Matt	-4.5%	4.5%	2	2	
RP1	Matt	-17.6%	17.6%	3	3	
TP12	Matt	-34.7%	34.7%	4	4	
TP9	Matt	39.3%	39.3%	5		5
TP6	Matt	-41.7%	41.7%	6	6	
TP7	Matt	48.0%	48.0%	7		7
TP10	Matt	49.2%	49.2%	8		8
TP4	Matt	-57.6%	57.6%	9	9	
TP11	Matt	67.0%	67.0%	10		10
TP13	Matt	116.7%	116.7%	11		11
JP1	Matt	119.8%	119.8%	12		12
TP3	Matt	183.3%	183.3%	13		13
TP5	Matt	250.0%	250.0%	14		14
				Sum:	24	81
				Test statistic:	24	
				Number of pairs:	14	
			Critical value for 95% confidence from table:		21	
Test statistic lies beyond critical value. This experiment provides no evidence to contradict the null hypothesis that neither Python nor Symlang are better than the other in terms of effort.						

Table 24 - Wilcoxon signed ranks, Python vs. Symlang, LOC, Jake Decker.

Wilcoxon signed ranks test for Python vs. Symlang implementation, all problems, developer Jake.						
Null hypothesis: the median added/changed LOC % for Traditional/Python implementations equals the median for COP/Symlang implementations.						
Alternate hypothesis: the medians differ.						
PROBLEM	DEVEL-OPER	CHANGE LOC	ABS CHANGE	RANK	NEG RANKS	POS RANKS
TP10	Jake	0.2%	0.2%	1.5		1.5
TP8	Jake	0.2%	0.2%	1.5		1.5
TP7	Jake	-0.5%	0.5%	3.5	3.5	
TP6	Jake	-0.5%	0.5%	3.5	3.5	
TP5	Jake	0.6%	0.6%	5		5
TP12	Jake	-0.7%	0.7%	6	6	
TP13	Jake	-0.8%	0.8%	7	7	
TP9	Jake	-1.4%	1.4%	8	8	
TP3	Jake	1.9%	1.9%	9		9
JP1	Jake	2.6%	2.6%	10		10
RP2	Jake	-5.7%	5.7%	11	11	
RP1	Jake	-7.6%	7.6%	12	12	
TP4	Jake	-9.5%	9.5%	13	13	
TP11	Jake	-22.7%	22.7%	14	14	
				Sum:	78	27
				Test statistic:	27	
				Number of pairs:	14	
		Critical value for 95% confidence from table:			21	
Test statistic lies beyond critical value. This experiment provides no evidence to contradict the null hypothesis that neither Python nor Symlang are better than the other in terms of % added/changed LOC.						

Table 25 - Wilcoxon signed ranks, Python vs. Symlang, LOC, Matt Evenson.

Wilcoxon signed ranks test for Python vs. Symlang implementation, all problems, developer Matt.						
Null hypothesis: the median added/changed LOC % for Traditional/Python implementations equals the median for COP/Symlang implementations.						
Alternate hypothesis: the medians differ.						
PROBLEM	DEVEL-OPER	CHANGE LOC	ABS CHANGE	RANK	NEG RANKS	POS RANKS
TP3	Matt	0.0%	0.0%			
TP5	Matt	0.0%	0.0%			
TP6	Matt	0.0%	0.0%			
TP13	Matt	1.4%	1.4%	1		1
TP12	Matt	1.5%	1.5%	2		2
RP2	Matt	1.6%	1.6%	3		3
TP4	Matt	2.0%	2.0%	5		5
TP8	Matt	2.7%	2.7%	7		7
TP10	Matt	4.6%	4.6%	8.5		8.5
TP7	Matt	4.6%	4.6%	8.5		8.5
JP1	Matt	7.4%	7.4%	10		10
RP1	Matt	8.1%	8.1%	11		11
TP9	Matt	9.7%	9.7%	12		12
TP11	Matt	-11.9%	11.9%	13	13	
				Sum:	13	68
				Test statistic:	13	
				Number of pairs:	11	
			Critical value for 95% confidence from table:		11	
Test statistic lies beyond critical value. This experiment provides no evidence to contradict the null hypothesis that neither Python nor Symlang are better than the other in terms of % added/changed LOC.						

5.0 CONCLUSIONS

The following subsections present the conclusions TA3 Provatek drew from its original IDAS evaluation effort (Subsection 5.1) and from the final evaluation it accomplished after the May-2021 IDAS program revision (Subsection 5.2).

5.1 The original IDAS effort

The authors offer the following observation to readers considering potential future research programs that, like IDAS, might seek to reduce the development effort required to update software to meet new requirements or changing environmental constraints.

IDAS presumed the same developers who produced the initial version of a piece of software would be the ones responsible for updating that software to meet new requirements. While we have no statistics on how often this has been the case in the past, the immense amount of software in use by the DoD and the long lifespans of some examples leads us to suspect that entirely new development teams do sometimes inherit the job of updating old software written by others.

In his paper "Programming as Theory Building" [NAU1985], Turing award winner Peter Naur proposes that the reasons why a given piece of software is structured the way it is, and thus the ways in which it might best be modified to meet some new requirement, is a theory that exists only in the minds of that software's developers. This theory is not traditionally encoded in the software's source code, nor is it to be found in typical specification or design documents. It can be conveyed to a new developer only by that developer joining the team of original developers and working in close contact with them for some time. If a piece of software's original team of developers is not available, Naur concludes, it would be less effort for a new development team to write a completely new piece of software from scratch to meet a set of updated requirements than it would be for them to modify the old one.

This is certainly an interesting conjecture; future research programs might explore cases where new developers update old software produced by others.

5.2 The final COP/Symlang experiment

Conclusions relevant to Apogee Research's experimental COP design methodology and Symlang programming language:

1. The experiment produced no evidence that either the traditional/Python control approach or the COP/Symlang experimental approach had an advantage over the other in terms of the effort required to extend solutions or in terms of the time/space efficiency of those solutions at runtime.
2. COP and Symlang may be a programming paradigm and programming language like logic programming and Datalog: optimized for and suited to a particular class of problem. The Rev problems that formed the basis of the challenges in this experiment may not have been members of COP/Symlang's ideal problem domain; a future experiment with challenges drawn from this ideal domain might deliver a different result.

3. Provatek's analyses used critical values sufficient only for 95% confidence; the results of this experiment could simply be due to bad luck. A similar experiment with more developers or more challenge problems might deliver a different result with higher confidence.

Conclusions relevant to the design of experiments to compare programming languages:

4. This experiment produced evidence that second solutions of a given problem took 25% fewer person-minutes of effort to complete than first solutions, likely because the first solutions bore the cost of understanding the problem and devising a strategy to solve it and the second solution had the benefit of that thinking with none of the cost.
5. This experiment produced no evidence that the mean proportion of lines of code added or changed was different between first and second solutions.
6. This experiment produced no evidence of a linear correlation between development effort in terms of person-minutes spent and the proportion of added or changed lines of code. Although easier to measure, counting added and changed lines of code does not appear to be a good predictor of person-minutes of effort.

6.0 REFERENCES

[BOE2000] Boehm, Barry and others, **Software Cost Estimation with COCOMO II**, 1st ed, Prentice Hall, Upper Saddle River, NJ, 2000.

[FRA2020] Fraser, Tim, *TA3 Advice on Test Evaluation 1 Challenges*, Provatek, Technical Report #99, Provatek, LLC 15301 Amberly Drive, Suite 250, Tampa, FL 33647, July 2020.

[FRA2022] Fraser, Tim, *IDAS Final Experiment Plan*, Provatek, Technical Report #114, Provatek, LLC 15301 Amberly Drive, Suite 250, Tampa, FL 33647, May 2022.

[LEE2022] Lee, Jessa and others, "Controller-Oriented Programming: A New Paradigm for Managing Software Complexity Creating Adaptable & Efficient Code through Partitions & Controllers," to appear, 2022.

[NAU1985] Naur, Peter, "Programming as Theory Building," *Microprocessing and microprogramming 15,* no. 5, 1985, pages 253-261.

[TAY2020] Taylor, Ryan, *IDAS: Requirements Specification for Logistics Domain, Transportation Subdomain – V4*, Carnegie Mellon University Software Engineering Institute, Pittsburgh, PA, September 2020.

7.0 LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AFRL	Air Force Research Laboratory
AMDD	Agile Model Driven Design
APT	Automated Program Transformations
CLIN	Contract Line Item Number
COCOMO	Constructive Cost Model
COP	Controller Oriented Programming
CMU/SEI	Carnegie Mellon University Software Engineering Institute
DARPA	Defense Advanced Research Projects Agency
DIN	Data Item Number
DoD	Department of Defense
GNU	GNU's Not UNIX
IDAS	Intent-Defined Adaptive Software
JP	Jake Problem
LOC	Lines Of Code
Lt. Col.	Lieutenant Colonel
OIC	Objectives-Intent-Constraints
PM	Program Manager
RP	Rev Problem
SMT	Satisfiability Modulo Theory
SNC	Sierra Nevada Corporation
TA	Technical Area
TP	Tim Problem
TDD	Test Driven Development