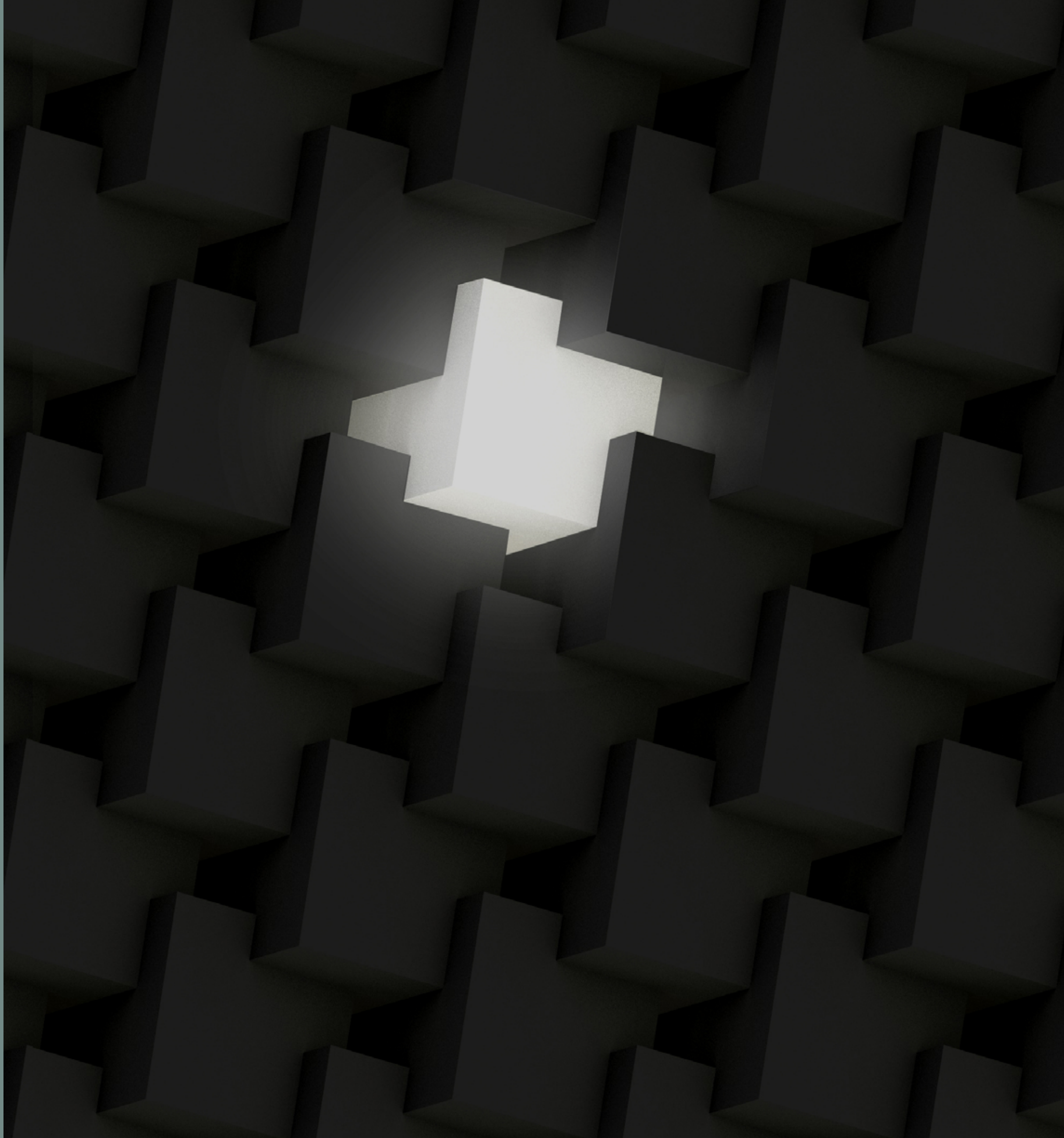
Carnegie Mellon University
Software Engineering Institute

RESEARCH REVIEW 2020



"Our research springs
from the DoD's need
for software innovation
and cybersecurity that
continually evolves
in support of its
intensifying mission."

TOM LONGSTAFF
Chief Technology Officer

Carnegie Mellon University
Software Engineering Institute

MESSAGE FROM THE CTO

We at the Carnegie Mellon University Software Engineering Institute (CMU SEI) are proud of our designation as a federally funded research and development center (FFRDC) sponsored by the Under Secretary of Defense, Research and Engineering (USD(R&E)). That pride shows in our ongoing commitment to establishing and advancing software as a strategic advantage for national defense and security.

This book highlights the fundamental research we conducted in fiscal 2020 on behalf of our DoD sponsor and presented at the 2020 CMU SEI Research Review. It presents recently concluded work and work that remains underway in our pipeline for technology development and transition: study, make, transition, and transfer.

As you will see in the following pages, we dig into the enduring challenges facing the DoD, and our decades-long engagement has informed our deep and nuanced understanding of the challenges it faces. Our research springs from the DoD's need for software innovation and cybersecurity that continually evolves in support of its intensifying mission.

The DoD needs its software-enabled systems to

- bring capabilities that make new missions possible or improve the likelihood of success of existing ones
- be timely to enable the DoD to field new software-enabled systems and upgrades faster than our adversaries
- be trustworthy in construction and implementation and resilient in the face of operational uncertainties including known and yet-unseen adversary capabilities
- be affordable such that the cost of acquisition and operations, despite increased capability, is reduced, predictable, and provides a cost advantage over our adversaries

Those four requirements drive all CMU SEI work, whether for USD(R&E), DoD programs, federal civilian agencies, or industry.

Our work in the problem space often generates fruitful collaborations with CMU academic departments, other leading universities, and industry that identify promising basic research and emerging technologies of use to the DoD. We also engage with DoD end users to gain field-level understanding of mission needs, gaps, and priorities, and with industry to assess existing capabilities.

When creating solutions, CMU SEI develops prototypes using promising early research or technology, maturing it as we do to meet the needs of the DoD. We conduct initial validation with forward-leaning DoD end users in major defense acquisition programs, combatant commands, or combat support agencies. We also identify opportunities for cost sharing with federal agencies facing challenges similar to those of the DoD.

Our transition efforts take place through additional direct engagements, and it is funded from across the inter-agency to refine prototypes to facilitate broad adoption by the entire DoD or to transfer these technologies to an industry or DoD partner for further integration or ongoing maintenance.

Across that pipeline, our steadfast purpose is to help the DoD gain and sustain an advantage over adversaries through the transformation of software acquisition, sustainment, and cyber operations within DoD. We strive to help the DoD do so in a way that matures and integrates advanced capabilities discovered by academia, government, and the private sector through a process that is routine, affordable, trustworthy, and timely.

I hope you enjoy reading about CMU SEI's fiscal 2020 research efforts, and that the following pages demonstrate the pride we take in this work. We stand by to work with you to help you make a difference, and we encourage you to contact us at **info@sei.cmu.edu**.

CONTENTS

Leverage Emerging Technology Innovation in Computing, Architectures, and Algorithms

Formalize the Development, Integration, and Use of Models

Codify Fully Integrated CI/CD Practices

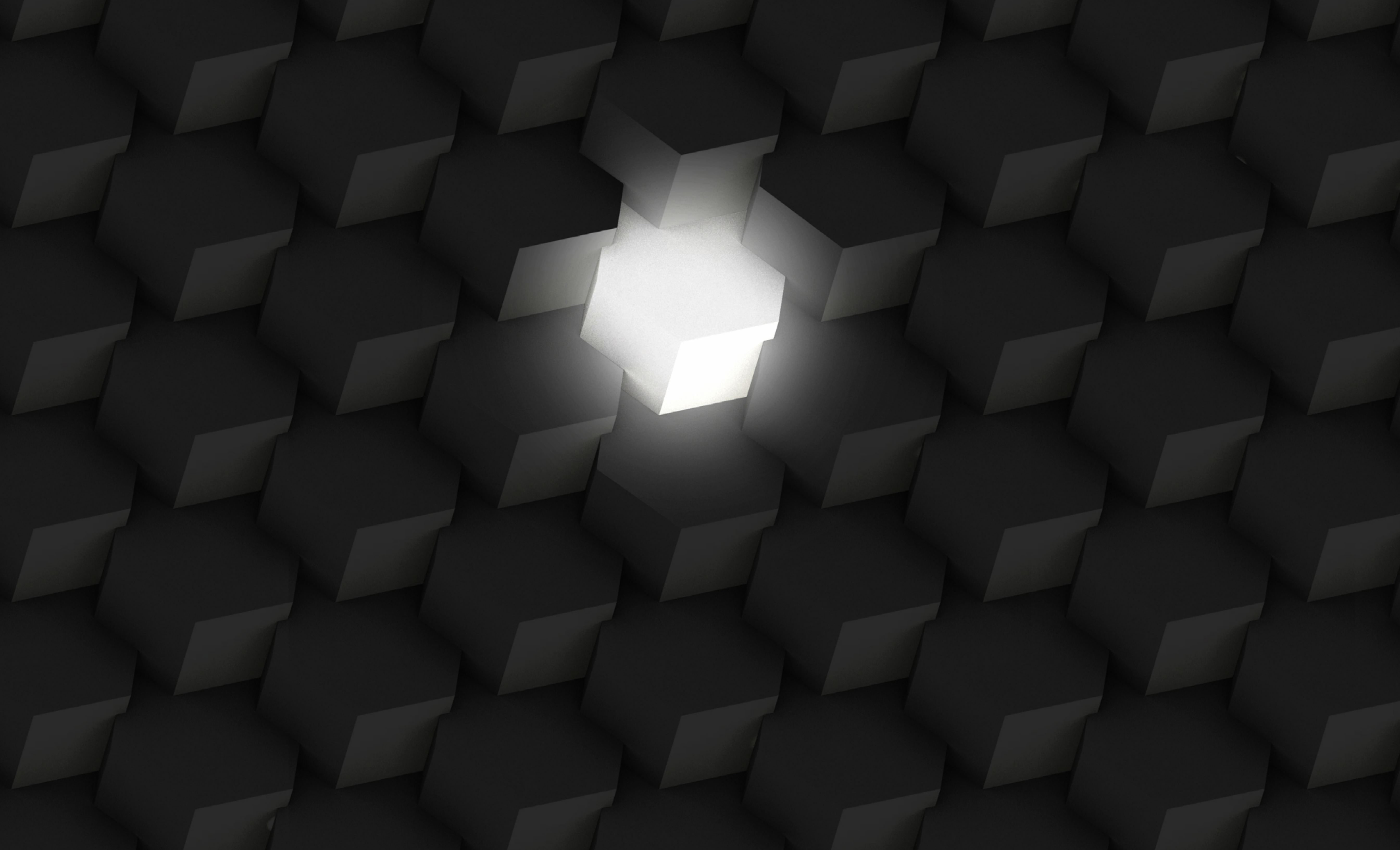
Improve Designed-In Resilience

Equip the Cyber and Information Operators with Dominant Tradecraft

SECTION 1

Leverage Emerging Technology Innovation in Computing, Architectures, and Algorithms

CMU SEI takes promising technologies and research relevant to DoD missions and adapts and enhances them to allow integration into DoD systems and processes.



RESEARCH REVIEW 2020

Spiral AI/ML: Co-optimization for High-Performance, Data-Intensive Computing in Resource Constrained Environments

Problems

- The need exists for increased computational power to process, exploit, and disseminate information for decision makers.
- Massive amounts of information, along with AI/ML algorithms, generate data and computational-intensive applications.
- Implementing these applications efficiently on increasingly complex HW/SW architectures is challenging.
- Too few engineers have the expertise to optimize algorithms for the wide variety of hardware currently available.

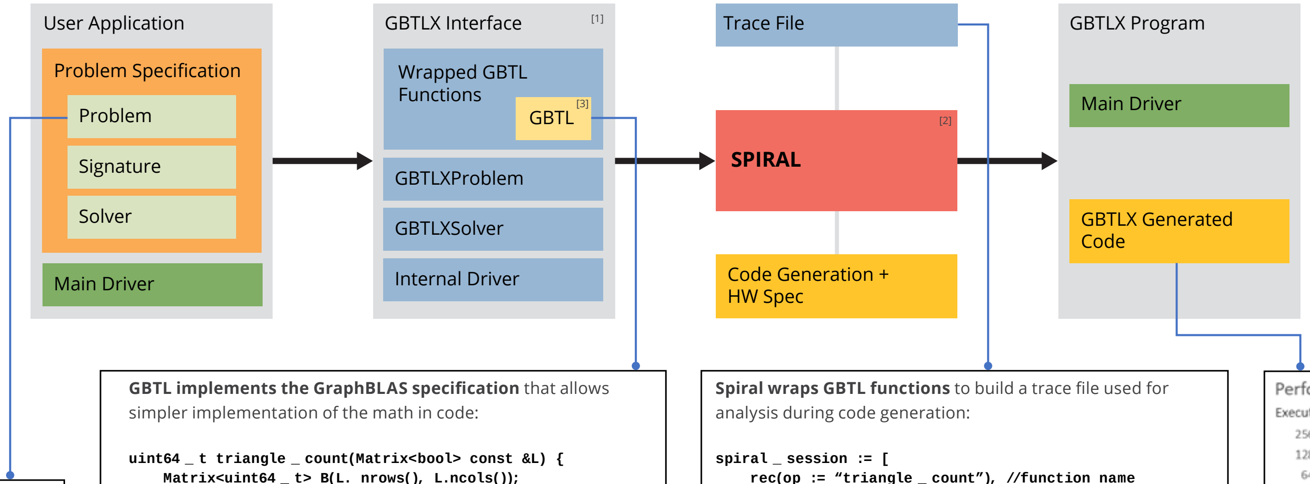
Solution

- Automatic code generation for data-intensive computations
- Simultaneous, automatic co-optimization for targeted hardware

Approach

- Identify and encode data-intensive compute primitives into CMU's SPIRAL code generation technology.
- Develop and encode hardware performance models into Spiral.
- Use Spiral to co-optimize for a set of target hardware platforms.

Hardware-software co-optimization promises timely, high-performance, and cost-effective implementation and re-implementation of AI/ML workloads on new **DoD** hardware platforms.



Graph algorithms in the language of linear algebra supports a rich notation for specifying graph, ML and AI algorithms. For example, counting triangles in graph **L**:

$$\Delta = \| \mathbf{L} \;.\mathrm{x}\; (\mathbf{L} \;+.\wedge\; \mathbf{L}) \|$$

includes use of semiring algebraic operations and masked matrix multiplies.

[6]

GBTL implements the GraphBLAS specification that allows simpler implementation of the math in code:

```
uint64 _ t triangle _ count(Matrix<bool> const &L) {
    Matrix<uint64 _ t> B(L. nrows(), L.ncols());

    // Masked matrix multiply: B = L .* (L +.^ L)
    mxm(B, L, NoAccum(), PlusAndSemiring<uint64 _ t>(), L, L);

    //Perform reduction: ||B||
    uint64 _ t count;
    reduce(count, NoAccum(), PlusMonoid<uint64 _ t>(), B);
    return count;
}
```

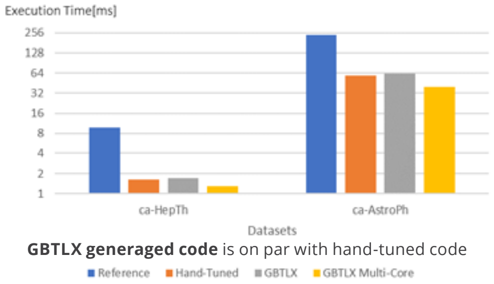
[3,4]

Spiral wraps GBTL functions to build a trace file used for analysis during code generation:

```
spiral _ session := [
    rec(op := “triangle _ count”), //function name
    rec(op := “MatrixCreation”,row:= 9877,col:= 9877,
        ptr := 0x7fffff45bb60, mat = 0x7fffff45bb60),
    rec(op := “Matrix Multiplication”,
        output = IntHexString(“0x7fffff45bb60”) ,
        mask   = IntHexString(“0x7fffff45ba30”),
        inputA = IntHexString(“0x7fffff45bb30”),
        inputB = IntHexString(“0x7fffff45bb30”),
        semiring = “PlusAnd”),
    rec(op := “reduce(matrix->scalar)”,
        /*many more arguments*/),
];
```

[1]

Performance of GBTLX on Triangle Counting



GBTLX generaged code is on par with hand-tuned code

[1,5]

References

1. S. Rao, A. Kutuluru, S. McMillan, F. Franchetti, “GBTLX: A First Look”, in 2020 IEEE High Performance Extreme Computing Conference (HPEC), 2020. ***Outstanding Student Paper Award.***
2. SPIRAL Project, Version 8.1.2. Available at **https://www.spiral.net.**
3. GraphBLAS Template Library (GBTL), Version 3.0. Available at **https://github.com/cmu-sei/gbtl**, June 2020.
4. A. Buluç, T. Mattson, S. McMillan, J. Moreira, and C. Yang, “Design of the GraphBLAS API for C,” in 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 643–652, 2017.
5. T. M. Low, V. N. Rao, M. Lee, D. Popovici, F. Franchetti, and S. McMillan, “First look: Linear algebra-based triangle counting without matrix multiplication,” in2017 IEEE High Performance Extreme ComputingConference (HPEC), pp. 1–6, 2017.
6. J. Kepner, D. Bader, A. Buluç, F. Franchetti, J. Gilbert, A. Lumsdaine, T. Mattson, S. McMillan, et al., “Mathematical Foundations of the GraphBLAS,” in 2016 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–9, 2016.

Principal Investigator

DR. SCOTT MCMILLAN
Member of the Technical Staff/Principal Engineer

Carnegie Mellon University Software Engineering Institute

SPIRAL AI/ML: CO-OPTIMIZATION FOR HIGH-PERFORMANCE, DATA-INTENSIVE COMPUTING IN RESOURCE-CONSTRAINED ENVIRONMENTS

Commanders and warfighters in the field rely on data, and the Department of Defense and U.S. intelligence community have an overwhelming data collection capability. This capability far outpaces the ability of human teams to process, exploit, and disseminate information. Artificial intelligence (AI) and machine learning (ML) techniques show great promise for augmenting human intelligence analysis. However, most AI/ML algorithms are computationally expensive, data intensive, and difficult to implement efficiently in increasingly complex computer hardware and architectures. What's more, moving very large amounts of data through tactical and operational military networks requires forward deployment of advanced AI/ML techniques to support commanders and warfighters in theaters with equipment constrained by cost, size, weight, and power (CSWAP).

As the military adopts AI/ML to augment human teams, the cost of implementing and re-implementing AI/ML software on new hardware platforms will be prohibitive. To address these challenges, we propose to build on CMU's Spiral technology, a hardware-software co-optimization system that will

- automatically search and select hardware configurations that meet CSWAP requirements
- automatically generate optimized codes for the selected hardware configuration and the irregular, data-intensive computations required for AI/ML algorithms

If successful, our solution will allow platform developers to realize high-performance AI/ML applications on leading-edge hardware architectures faster and cheaper. These advances will allow for rapid development and deployment of capabilities across the spectrum of national and tactical needs.

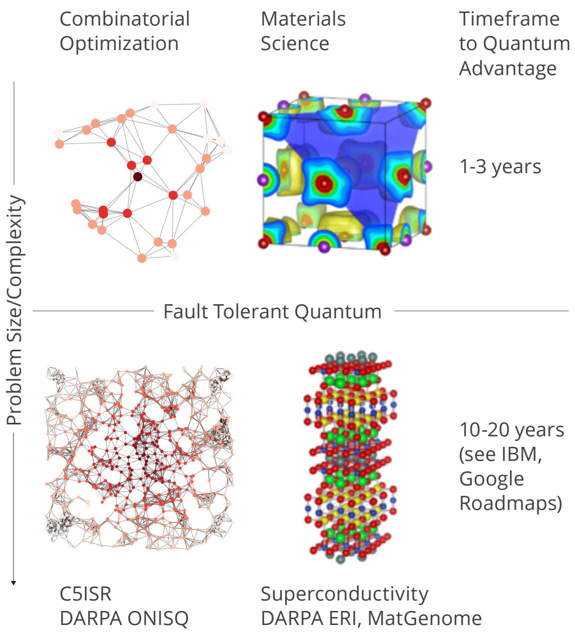
IN CONTEXT: THIS FY2019–21 PROJECT

- builds on DoD line-funded research and sponsored work on automated code generation for future-compatible high-performance graph libraries, big learning benchmarks, GraphBLAS API specification, and graph algorithms on future architectures
- is related to a set of programs at DARPA under the ERI umbrella (HIVE, SDH DSSOC, etc.) that the CMU SEI is supporting
- aligns with the CMU SEI technical objective to be affordable such that the cost of acquisition and operations, despite increased capability, is reduced and predictable and provides a cost advantage over our adversaries

Quantum Advantage Evaluation Framework

Problem

When and where can the DoD benefit from investing in quantum computing technology? To answer this question, we are working with noisy intermediate scale quantum (NISQ) computers, but we're also thinking ahead to fault-tolerant quantum error corrected computation. In particular, we want to determine when and where quantum advantage will exist for the following important DoD applications:



Combinatorial Optimization

Materials Science

Timeframe to Quantum Advantage

1-3 years

Problem Size/Complexity

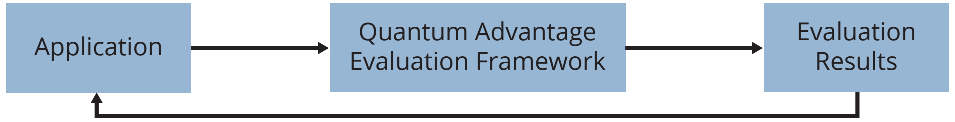
Fault Tolerant Quantum

10-20 years (see IBM, Google Roadmaps)

C5ISR
DARPA ONISQ

Superconductivity
DARPA ERI, MatGenome

Framework to evaluate current and **projected quantum computing advantage.**



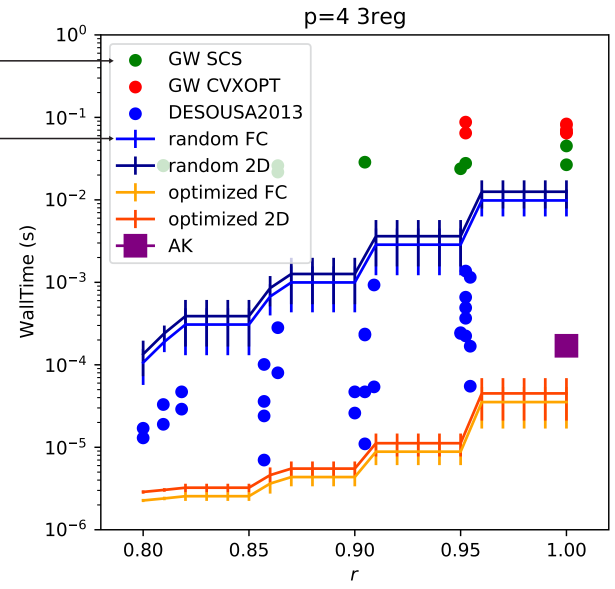
Application

Quantum Advantage Evaluation Framework

Evaluation Results

Classical State of the Art: PSC

Quantum Computers (simulated)



Where: to determine quantum advantage, benchmarks on specific problem instances must be performed on "real world" scales (O(100-1000+ nodes) (estimated 3 years IBM, Google)

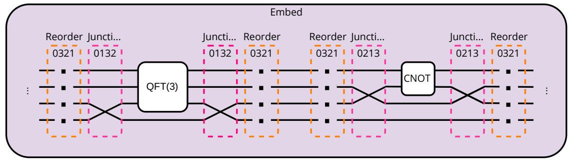
https://arxiv.org/abs/2006.04831

"Assessment of Alternative Objective Functions for Quantum Variational Combinatorial Optimization," M. Jonsson, et al, IEEE QCE Quantum Week 2020

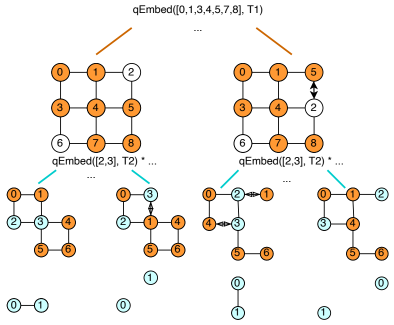
SEI Collaboration with Carnegie Mellon University

Quantum algorithm performance depends critically on quantum circuit optimization. We are working with CMU ECE Franz Franchetti's group to adapt their well-known classical computing optimization tool, SPIRAL.

Quantum Circuit Optimization in SPIRAL



Scheduling to Quantum "Baremetal"



http://spiral.net/

https://github.com/spiralgen/spiral-package-quantum

"Quantum Circuit Optimization with SPIRAL: A First Look," S Mionis, et al, Supercomputing 2020

QAEF Output: When and where can you leverage quantum computing to achieve advantage in solving your organization's problems?

- Input: the applications that have most potential for quantum advantage. It is critical to identify "real world" problem instances.
- Output: when and where will quantum advantage exist? Establish timeframe for Quantum Advantage Readiness.

Principal Investigator

DR. JASON LARKIN
Research Scientist

Carnegie Mellon University
Software Engineering Institute

QUANTUM ADVANTAGE EVALUATION FRAMEWORK

The potential of quantum computing, especially near-term, is not going to be realized without close integration with state-of-the-art classical computing. Universal gate (UG) quantum computers share many foundational features with classical computers. Furthermore, UG quantum computers must show advantage against state-of-the-art classical software and/or hardware, and the two computing paradigms will be critically integrated as complimentary technologies.

A major gap in achieving quantum advantage is the identification of applications in which quantum computing could provide computational advantage (in terms of time to solution, quality of solution, etc.). It is unclear which potential applications will realize quantum advantage among a variety of hardware, such as various UG technologies (e.g., superconducting qubit, trapped and neutral-atom, photonics). Variation in hardware is typical in the near-term noisy intermediate-scale quantum (NISQ) computing era. This is a software–hardware co-synthesis challenge for quantum computing in the near-term.

This project aims to produce a novel classical computing emulation and software–hardware co-synthesis framework for quantum computing technology aimed at applications driven by the portfolio of DoD research. UG quantum computing has emerged as the near-term (5- to 10-year) quantum computing technology that can demonstrate not just quantum supremacy (performing a computation not possible with a classical computer, regardless of usefulness), but also quantum advantage (performing a useful computation better and/or faster than a classical computer).

IN CONTEXT: THIS FY2019–21 PROJECT

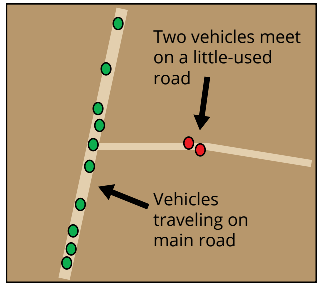
- relates to DoD interest in applying quantum computing to mission capability
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of uncertainties, including known and yet-unseen adversary capabilities
- aligns with the CMU SEI technical objective to bring capabilities through software that make new missions possible or improve the likelihood of success for existing missions
- provides a gateway into futuristic computing architectures and increased computational power for artificial intelligence and machine learning

Video Summarization and Search: Object Tracking

Problem:

Aerial surveillance demands full attention to video by PED teams

- Manual, error-prone process
- Technical barriers including object detection, and tracking
- Limitations result in poor pattern detection in a surveilled region


Two vehicles meet on a little-used road
Vehicles traveling on main road

- Vehicle tracks used to train LSTM autoencoder that learns normal behavior in order to identify anomalous tracks
- Results shown are for perfect data -- reality is not so pretty due to inadequate object detection and tracking
- This results in lost tracks and many "tracklets" that are difficult to correlate

Solution

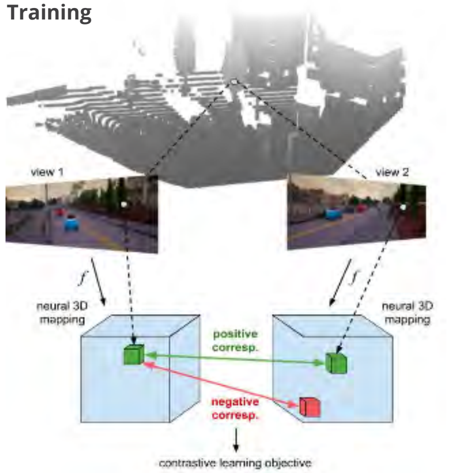
- Work directly with DoD to improve pattern detection in aerial surveillance data patterns
- Work with researchers to address core technology problems of tracking of objects

Impact (FY18–20)

- Improved DoD pattern detection in aerial surveillance data
- Developing unsupervised 3D tracking algorithms to improve on other unsupervised methods and achieve performance similar to supervised methods

3-D Tracking Research: **learning correspondence** from **static 3D points causes 3D object tracking to emerge.**

Training



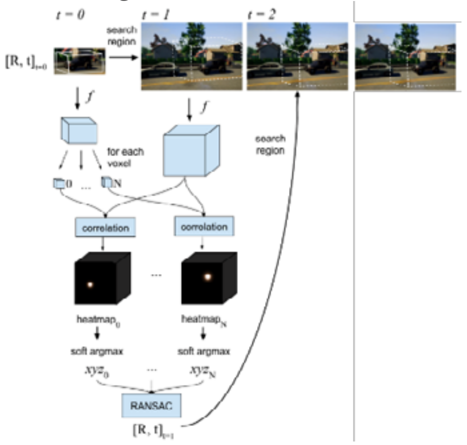
Given the bounding box for object

- Generate features for the object
- Generate features for search region
- For each voxel of object, compute correlation with search region
- Estimate the total motion with RANSAC
- Update the box automatically

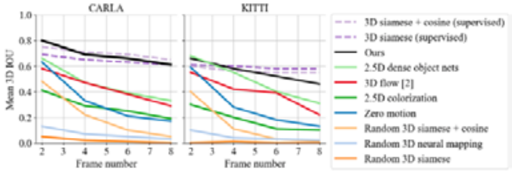
Given 2 viewpoints of the same object:

- a neural 3D mapping for each
- Identify the corresponding voxel pair in the two mappings
- Treat all other mappings as negative correspondences
- Train the features to indicate the correspondences automatically

Tracking



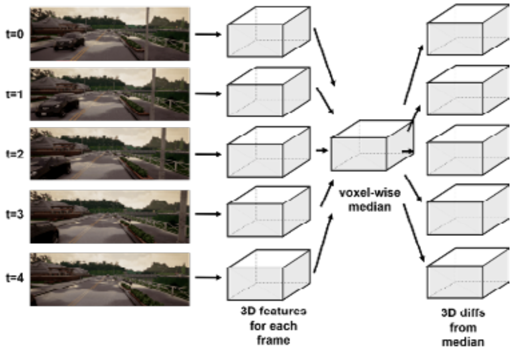
Results: Tracking based on learned correspondence of points



Object Discovery

What if the need is for a system that will discover objects autonomously?

- Extract 3D features for each frame
- Determine voxel-wise median
- Determine the difference from the median for each frame



Early results are promising!

- Work on 3D tracking will continue as part of Adam Harley's work toward his PhD at Carnegie Mellon University

Contact:

Ed Morris
info@sei.cmu.edu

Adam Harley
info@sei.cmu.edu

Principal Investigator

MR. EDWIN MORRIS
Senior Member of the
Technical Staff

Carnegie Mellon University
Software Engineering Institute

VIDEO SUMMARIZATION AND SEARCH

The U.S. relies on surveillance video to determine when activities of interest occur in a surveilled location. Yet, there is a lack of automated tools available to assist analysts in monitoring real-time video or analyzing archived video [Seligman 2016]. Consequently, analysts now need to dedicate full attention to video streams to avoid missing important information about ongoing activities and patterns of life; and, in tactical settings, warfighters miss critical information for improved situational awareness because they cannot stare at a tablet strapped to their chest.

In this work, we are developing machine learning algorithms necessary for detecting objects, better tracking those objects, and recognizing patterns of objects and object interactions.

IN CONTEXT: THIS FY2018–20 PROJECT

- builds on prior DoD line-funded research into the foundations for summarizing and learning latent structure in video
- draws from sponsored engagements for DoD programs and agencies
- aligns with the CMU SEI technical objective to bring capabilities through software that make new missions possible or improve the likelihood of success of existing ones

A Series of Unlikely Events

Learning from Sequential Behavior for Activity-Based Intelligence and Modeling Human Expertise

Introduction

Modeling patterns of sequential behavior is a task that underlies numerous difficult artificial intelligence tasks:

- How do I detect when adversaries are deviating from normal routines?
- How can I predict where a ship is going to dock?
- How can I automate the teaching of novice analysts to perform complex tasks as if they were experts?

In this work, we use a class of techniques called **Imitation Learning (IL)** to model sequential behavior to answer questions like these and others.

Methodology

Given observations of behavior:

$$\mathcal{B} = \{((s_1,a_1),(s_2,a_2),...)1,...,((s_1,a_1)...)_n\}$$

Learn a policy $\pi:\mathcal{S}\mapsto\mathcal{A}$ that best explains the behaviors.

Two Kinds of Imitation Learning Algorithms

1. Inverse Reinforcement Learning: Learn a reward function $R:\mathcal{S}\times\mathcal{A}\mapsto\mathbb{R}$ that models the preference exhibited in behavior. Then, learn policy π that maximizes expected reward.
2. Behavioral Cloning: Learn π directly to mimic the actions exhibited in the behaviors.

How to use Imitation Learning for…

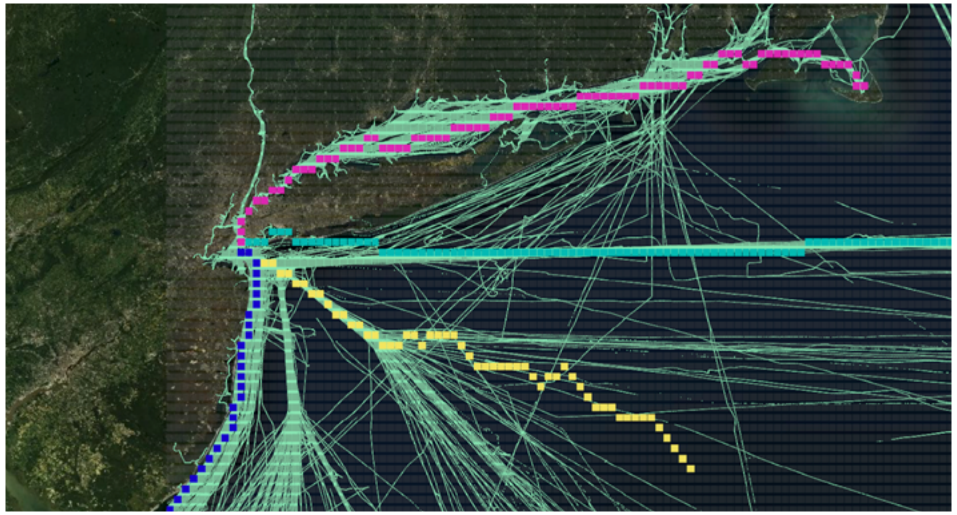
Activity-based Intelligence:

- Learn π from observed behaviors of entities of interest.
- Project future behavior by successively applying π to state.
- Detect anomalous behavior when π deems an action to be of low probability (assumes probabilistic policy).

Teaching expert behavior:

1. Learn π from expert behavior.
2. When a novice is in a state for which she doesn't know the proper action, suggest the one produced by π.

Imitation Learning techniques are an efficient and effective means to perform **activity-based intelligence** or to help automate the education of novices on how to **perform tasks like experts.**



Goals of this work

1. Investigate the practicality (assumptions made, efficiency, scalability, expressiveness) of applying IL to behavioral modeling problems.
2. Apply IL Techniques to DoD/IC relevant problems:
   - Perform efficient implementations that scale to a large number of observations.
   - Build demonstration from data ingestion to visualization tools.
3. Develop techniques that are able to explain, simulate, and demonstrate expert behavior.

Accomplishments

1. Performed technical evaluation of Generative Adversarial Imitation Learning (GAIL) (Ho and Ermon, 2016) and Disturbances for Augmenting Robot Trajectories (DART) (Lee et al., 2017) when applied to modeling nautical vessel behavior.
   - With careful engineering and domain-specific modeling assumptions, we were able to achieve a policy that was able to predict a ship's end destination state within 0.001% of their actual state (technical report forthcoming).
2. Created implementation of Maximum Causal Entropy IRL (MCEIRL) (Ziebart et al., 2010) that is 500x+ faster than academic implementation (to be publicly released).
3. Created demonstration of MCEIRL model applied to U.S. Coast Guard Nautical Vessel Data. (**https://resources.sei.cmu.edu/downloads/IRL-demo**)
4. Developed model with Stephanie Rosenthal (CMU/CSD) and Reid Simmons (CMU/RI) of expert data scientist behavior for the purpose of guiding novice data scientists through challenging tasks (technical report forthcoming).

Ho, Jonathan and Ermon, Stefano. Generative Adversarial Imitation Learning. *Advances in Neural Information Processing Systems (NIPS), 29. D. D. Lee et al. (eds]. NIPS Foundation.* 2016.
Lee, Jonathan et al. DART: Disturbances for Augmenting Robot Trajectories. *1st Conf. on Robot Learning (CORL) Project*. Nov. 2017.
Ziebart, Brian D, et al. *Maximum Causal Entropy IRL (MCEIRL).* School of Computer Science, Carnegie Mellon University. 2010.

Principal Investigator

DR. ERIC HEIM
Senior Research Scientist—
Machine Learning

Carnegie Mellon University
Software Engineering Institute

A SERIES OF UNLIKELY EVENTS

The Department of Defense (DoD) and the intelligence community (IC) frequently analyze activity based intelligence (ABI) to inform missions about routine patterns of life (POL) and unlikely events that signal important changes. For example, monitoring parking lots of military bases may indicate changing threat levels or upcoming military action. Despite growing research on general solutions for routine detection technologies, current algorithms are typically hand-crafted for particular applications, require labeled anomalous data, and have high false-positive rates that require verification by human analysts.

We propose an alternative approach, inverse reinforcement learning (IRL), that observes all states and actions in data and computes a statistical model of the world that includes whether each behavior is part of a routine. Deviations from routines have a low likelihood of occurrence within the model. The statistical model can also explain why an action is labeled as routine or anomalous and could be used by analysts to prioritize the anomalies and to retrain models to reduce false positives.

Though powerful, IRL techniques pose a number of both practical and fundamental challenges when applying them to dynamic, large-scale, DoD and IC missions. In this project, we focus on three of these challenges: 1) scaling IRL methods to DoD/IC-scale problem domains using efficient implementations of state-of-the-art techniques and high-performance computing, 2) making IRL techniques robust to novelty, thus allowing them to reason about never-seen-before behaviors, and 3) developing IRL techniques that expose key characteristics in data that could explain observed behaviors.

IN CONTEXT: THIS FY2018–20 PROJECT

- builds on DoD line-funded research, including graph algorithms and future architectures, big learning benchmarks, automated code generation for future-compatible high-performance graph libraries, data validation for large-scale analytics, and events, relationships, and script learning for situational awareness
- aligns with the CMU SEI technical objective to bring capabilities through software that make new missions possible or improve the likelihood of success for existing missions

Train, but Verify: Towards Practical AI Robustness

Problem

The Beieler Taxonomy (2019) categorizes three ways a machine learning system can be attacked. The three matching security policies for a defender to enforce are:

1. **Learn** the right thing, even from adversary influenced data.
2. **Do** the right thing, even with adversarial examples present.
3. Never **Reveal** sensitive information about the model/data.

Existing defense research primarily focuses on only one of these security policies at a time. This is an important limitation, because recent research demonstrates that state of the art methods for enforcing do policies can lead to violations of reveal policies.

Train\Verify	Verify learn	Verify do	Verify reveal
Train for learn			
Train for do		**Train, but Verify**	
Train for reveal			

Solution

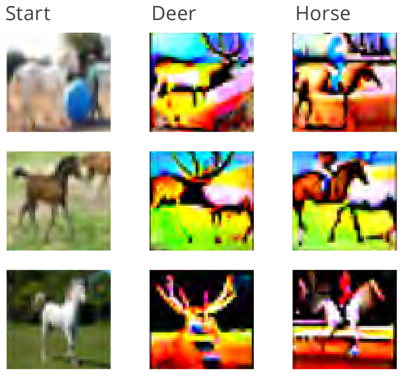
1. **Train** secure AI systems by training ML models to enforce at least two security policies.
2. **Verify** the security of AI systems by testing against realistic threat models across multiple policies.

Intended Impact (FY20-22)

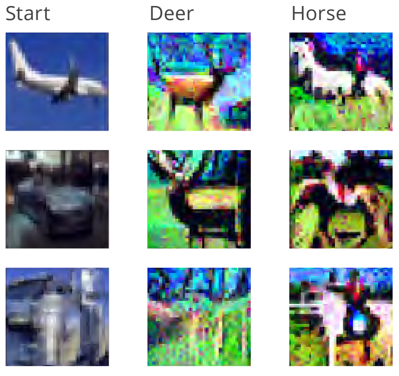
- Provide proof-of-concept defenses that either enforce multiple policies, or trade off between those policy goals.
- Provide proof-of-concept tooling to verify security policies across multiple policies.

An AI system **trained** for high-stakes decisions may **reveal critical information** about its training data.

For models trained on CIFAR 10 to enforce a do policy (TRADES, Zhang et al., 2019), adversaries with both full-model access and query-only access can recover the presence of riders on horses (about 20% of the class).



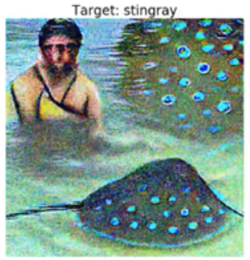
Adversary w/ full model access



Adversary w/ query only access

Examples of horses with riders from CIFAR10 (about 20% of the class)

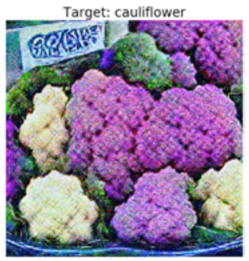
The ImageNet stingray class contains swimmers



Adversary with model access, but no data

First 9 examples of synset n01498041 (stingray)

... Cauliflower class contains purple cauliflower



Adversary with model access, but no data

First 9 examples of synset n07715103 (cauliflower)

CIFAR 10 data set documented in Krizhevsky, Alex. "Learning Multiple Layers of Features from Tiny Images." April 8, 2009.

ImageNet photos courtesy of ImageNet.

Principal Investigator

DR. NATHAN VAN HOUDNOS
Senior Machine Learning Research Scientist

Carnegie Mellon University Software Engineering Institute

MR. JON HELLAND
Associate Machine Learning Researcher

Carnegie Mellon University Software Engineering Institute

TRAIN, BUT VERIFY: TOWARDS PRACTICAL AI ROBUSTNESS

The current challenges to the training and verification of secure machine learning (ML) stem from

1. the difficulty of enforcing quality attributes in a system that is trained on data instead of directly constructed from requirements
2. the fundamental advantage that an attacker has, namely that the attacker needs to only violate a single security policy, while the defender needs to enforce all of the security policies

The DoD has not been exempt from these challenges. The current state of the art in secure ML is to train systems to either enforce a single security policy or train auxiliary systems to detect violations of a single security policy. Very little extant work focuses on multiple security policies. For example, there exist systems in the DoD that make high-stakes decisions and yet were also trained on sensitive data. This implies that the system should enforce at least two security policies simultaneously (i.e., the ML system should neither do the wrong thing when presented with adversarial input nor reveal sensitive information about the training data during its operation).

In this “Train, but Verify” project, we will attempt to address the gap in the state of the art on secure training of ML systems with two objectives:

1. Train secure AI systems by training ML models to enforce at least two security policies.
2. Verify the security of AI systems by testing against declarative, realistic threat models.

We consider security policies from the Beieler taxonomy: ensure that an ML system does not learn the wrong thing during training (e.g., data poisoning), do the wrong thing during operation (e.g., adversarial examples), or reveal the wrong thing during operation (e.g., model inversion or membership inference).

IN CONTEXT: THIS FY2020–22 PROJECT

- aligns with the CMU SEI technical objective to be trustworthy in construction and implementation and resilient in the face of operational uncertainties, including known and yet-unseen adversary capabilities.

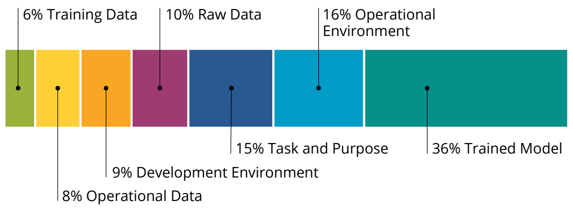
Characterizing and Detecting Mismatch in ML-Enabled Systems

Problem
Development, deployment, and operation of ML systems involves three perspectives, often with three completely separate workflows and people: data scientists build the model; software engineers integrate the model into a larger system; and then operations staff deploy, operate, and monitor the system.

Because these perspectives operate separately and often speak different languages, there are opportunities for mismatch between the assumptions made by each perspective with respect to the elements of the ML-enabled system, and the actual guarantees provided by each element.

Solution
Develop descriptors for elements of ML-enabled systems by eliciting examples of mismatch from practitioners; formalizing definitions of each mismatch in terms of data needed to support detection; and identifying potential for using this data for automation of mismatch detection.

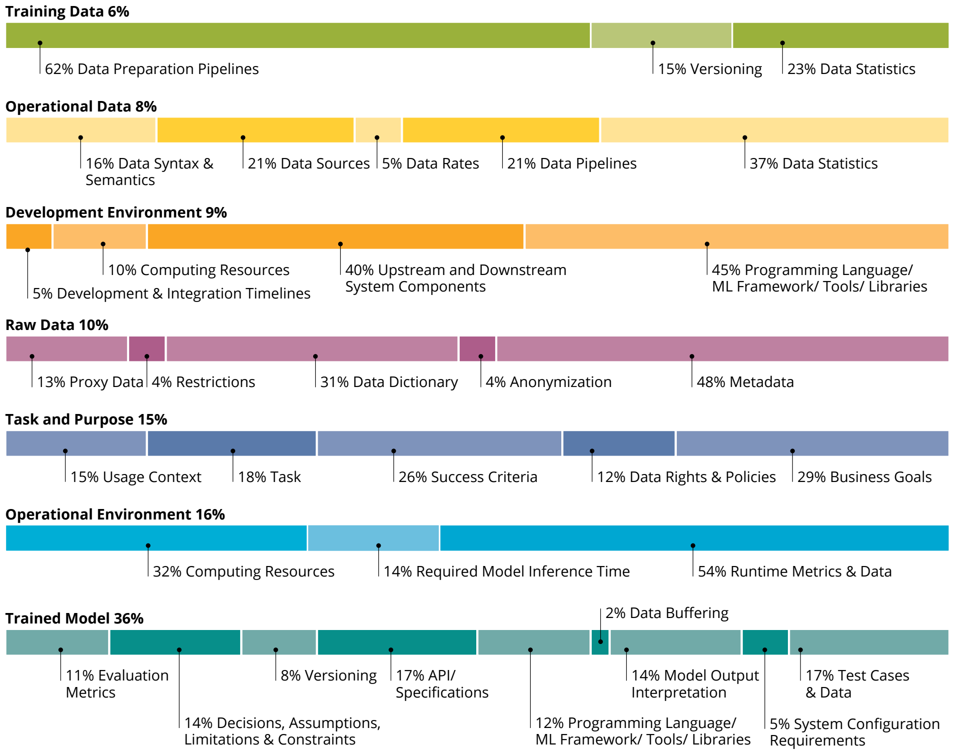
Phase 1: Practitioner interviews to elicit examples of mismatch and their consequences



Resulting Mismatch Categories from Practitioner Interviews

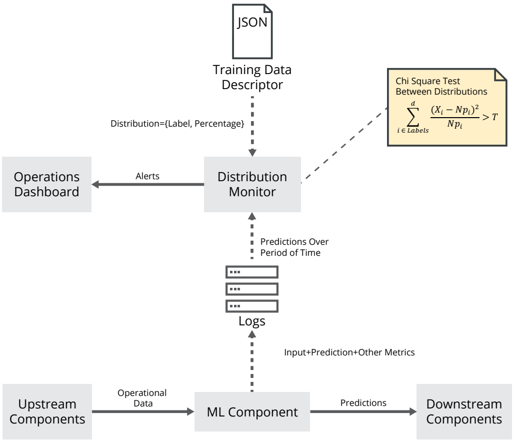
Descriptors for ML system elements **make stakeholder assumptions explicit** and prevent mismatch.

Phase 1 Findings



Training Data mismatches are mostly due to lack of clarity on data preparation pipelines (37%) and lack of data statistics (21%).
Operational Data mismatches are mostly due to lack of data statistics (37%) and lack of clarity on data pipelines (21%).
Development Environment mismatches are mostly due to differences in programming languages ... (45%) and lack of knowledge of upstream and downstream components (40%).
Raw Data mismatches are mostly associated with lack of metadata (48%) and lack of a “data dictionary” (31%).
Task and Purpose mismatches are mostly associated with unknown business goals (29%) or success criteria (26%).
Operational Environment mismatches are mostly associated with unavailable runtime metrics and data (54%) and unawareness of computing resources available for model serving (32%).
Trained Model mismatches are mostly associated with lack of test cases and test data (17%) and lack of model specifications and APIs (17%).

Looking Ahead: Automated Mismatch Detection


JSON
Training Data Descriptor
Distribution={Label, Percentage}
Chi Square Test Between Distributions
$\sum_{i \in Labels}^{d} \frac{(X_i - Np_i)^2}{Np_i} > T$
Operations Dashboard
Alerts
Distribution Monitor
Predictions Over Period of Time
Logs
Input+Prediction+Other Metrics
Upstream Components
Operational Data
ML Component
Predictions
Downstream Components

Descriptors Being Used for Automated Drift Detection

Principal Investigator

DR. GRACE A. LEWIS
Principal Researcher/
Tactical AI-Enabled Systems
Initiative Lead

Carnegie Mellon University
Software Engineering Institute

CHARACTERIZING AND DETECTING MISMATCH IN ML-ENABLED SYSTEMS

Despite the growing interest in machine learning (ML) and artificial intelligence (AI) among the DoD, government, and public sector organizations, development of ML and AI capabilities remains primarily a research activity or stand-alone project (with the exception of large companies such as Google and Microsoft). [Ghelani 2019] Deploying ML models in operational systems remains a significant challenge. [Amershi et al. 2019; Ransbotham et al. 2017; Sculley et al. 2015]

The development and operation of ML-enabled systems involve three perspectives with three different and often completely separate workflows and people: the data scientist builds the model; the software engineer integrates the model into a larger system; and operations staff deploy, operate, and monitor the system. Because these perspectives operate separately and often speak different languages, mismatches can arise between the assumptions of each perspective about the elements of the ML-enabled system and the actual guarantees provided by each element. Furthermore, these system elements, such as the trained model, training data, raw data, and operational environment, evolve independently and at a different rhythm, which could, over time, lead to unintentional mismatch. Such mismatch can manifest in poor system performance, poor model accuracy, the need for large amounts of glue code to accommodate operational data types, monitoring tools incapable of detecting diminishing model accuracy, and even system failure.

This project addresses the following questions:

- What are common types of mismatch that occur in the end-to-end development of ML-enabled systems?
- What are best practices for documenting data, models, and other system elements that will enable detection of mismatch?
- What are examples of mismatch that could be detected in an automated way, based on the codification of best practices in machine-readable descriptors for ML system elements?

We are developing machine-readable ML-Enabled System Element Descriptors to enable mismatch detection and prevention in ML-enabled systems. These descriptors codify attributes of system elements and make all assumptions explicit. They can be used by system stakeholders manually, for information awareness and evaluation activities, and by automated mismatch detectors at design time and runtime for cases in which attributes lend themselves to automation.

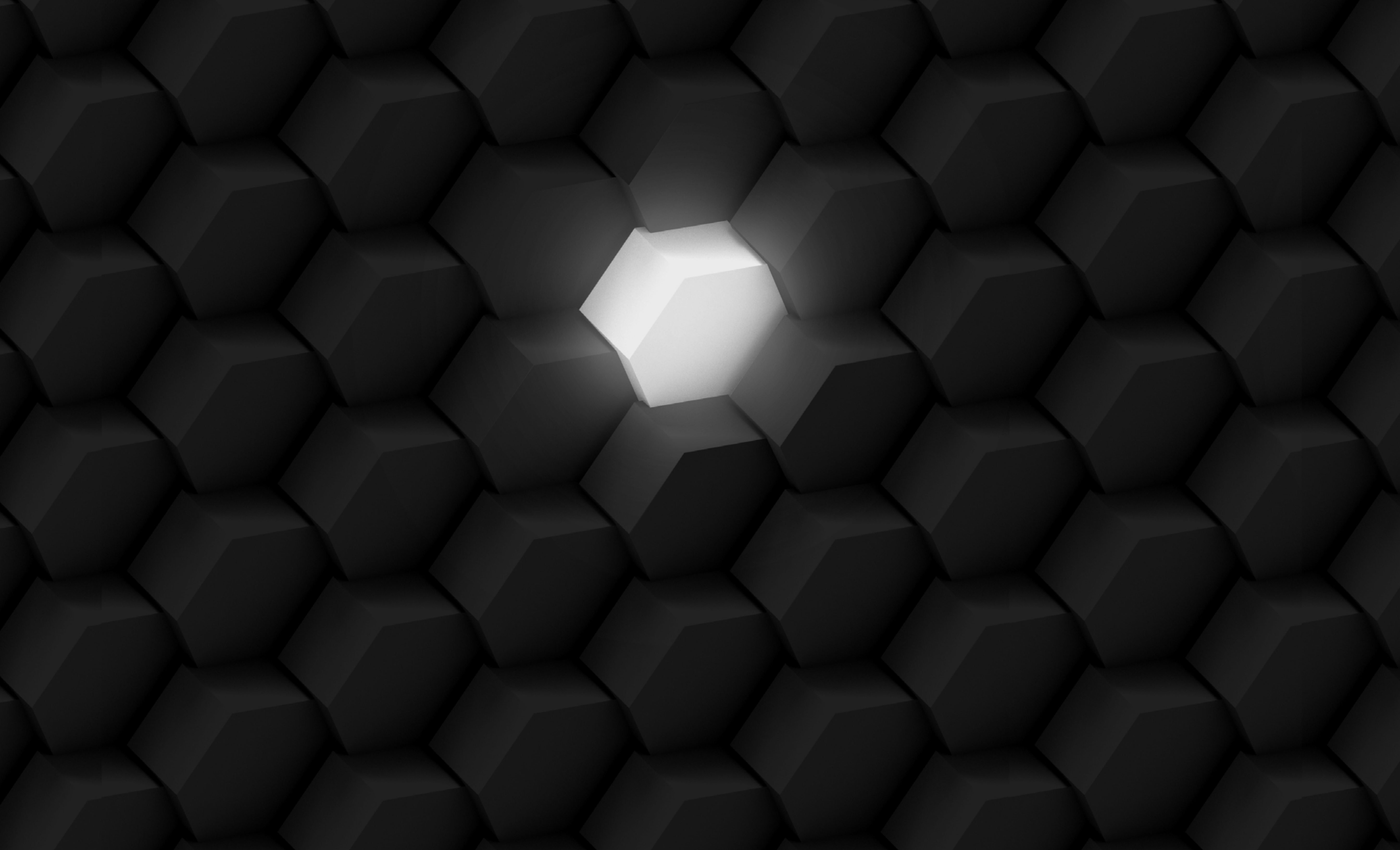
IN CONTEXT: THIS FY2020 PROJECT

- aligns with the CMU SEI technical objective to 1) be trustworthy in construction and implementation and resilient in the face of operational uncertainties, including known and yet unseen adversary capabilities, and 2) bring capabilities that make new missions possible or improve the likelihood of success of existing ones

SECTION 2

Formalize the Development, Integration, and Use of Models

CMU SEI seeks to 1) improve the fidelity and expressiveness of languages, models, and tools that allow the specification of software systems, and 2) ensure the ability to create these formalisms for new activities or extract them from legacy systems where critical documentation, development artifacts, source code, or formal descriptions may not exist.



Integrated Safety and Security Engineering for Mission-Critical Systems

Problem

Software increasingly dominates safety- and mission-critical system development. Issues are discovered long after they are created.

Solutions

Our three-year project aims to make systems safer and more secure by enabling early discovery of system-level issues through virtual integration and incremental analytical assurance. This project consists of four efforts, all of which use the Architecture Analysis and Design Language (AADL), an SEI-created, internationally standardized language for designing software-centric critical systems.

Security Requirements

A new security annex to AADL and verification plugins

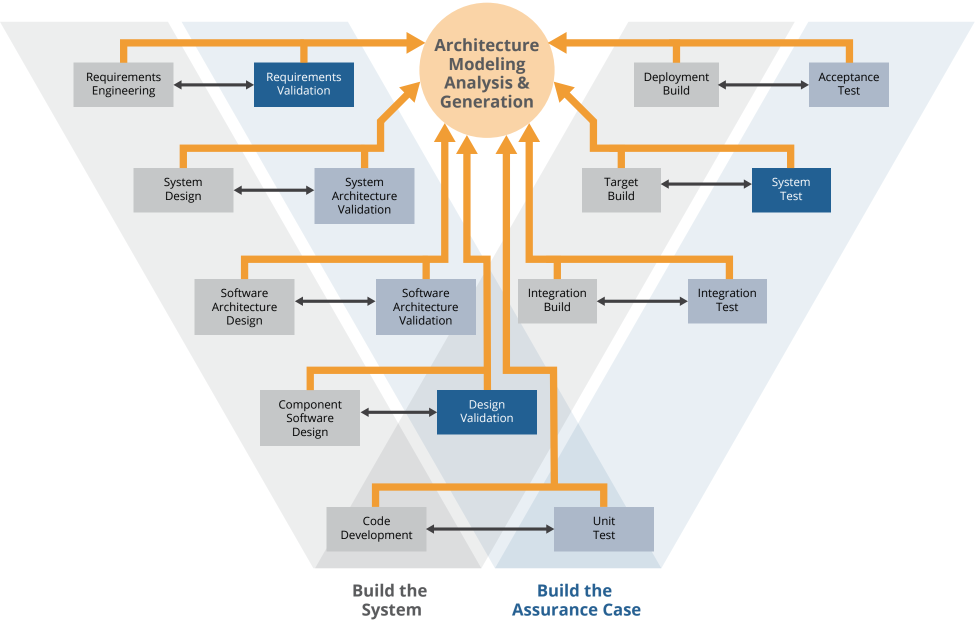
We developed an extension to AADL that enables system designers to describe how their system meets security goals by, for example, encrypting information or dealing with private keys. We also developed tools to verify that a system conforms to various policies, and we are publishing papers and documentation on how to use them.

Reusable Safety Patterns

A collection of patterns expressed using AADL

We proposed a library of safety design patterns that capture key safety architecture fragments. Each pattern is described using AADL, complemented by a machine-readable description of applicable error scenarios, a behavioral description of the nominal case, and a verification plan defined using custom tooling and AGREE / Resolute (tooling developed by Collins Aerospace). These formalizations are AADL implementations of existing patterns, and they equip system architects with modeling techniques and verification methods that are adaptable to various domains.

We're making it easier to **specify, design,** and **assure** critical systems that are safer and more secure.



Safety and Security Across the System Development Lifecycle

Architecture-Supported Audit Processor

A collection of system viewpoints for certification authorities

Performing a hazard analysis is a common way of examining a system for safety or security issues. This effort integrates a number of sources of system information—system architecture, error behavior, Kansas State's AWAS technology, and more—into a set of dynamic reports. The Architecture-Supported Audit Processor (ASAP) will allow system analysts to query interesting portions of a system's architecture interactively, rather than read only what an analysis format specifies.

[Off-]Nominal Behavior

Unified behavioral description

There are several ways to specify behavior in AADL, depending on what is being specified: (nominal) component behavior, off-nominal (i.e., erroneous) behavior, or mode-transition semantics. We produced a proposal to unify behavior specifications, which will make the language simpler and enable more powerful automated analyses.

AADL has been used in a variety of safety-critical domains, including medical devices, automotive components, and military and commercial aviation.

Principal Investigator

DR. SAM PROCTER
Senior Architecture Researcher

Carnegie Mellon University
Software Engineering Institute

INTEGRATED SAFETY AND SECURITY ENGINEERING FOR MISSION-CRITICAL SYSTEMS

Critical systems must be both safe from inadvertent harm and secure from malicious actors. However, safety and security practices have historically evolved in isolation. Safety-critical systems, such as aircraft and medical devices, were long considered standalone systems without security concerns. Security communities, on the other hand, have focused on information security and cybersecurity. Mechanisms such as partitioning, redundancy, and encryption are often deployed solely from a safety or security perspective, resulting in over-provisioning and conflicts between mechanisms. Despite the recognition that this disconnect is harmful, there is limited understanding of the interactions between safety and security. [Friedberg 2017]

To combat this lack of understanding, we are developing an integrated safety and security engineering approach based on system theory and supported by an AADL-based workbench. This approach

- unifies safety and security analysis through a formalized taxonomy that is used to drive system verification via fault-injection and simulation
- provides a design framework to combine safety and security mechanisms into a more robust and resilient system architecture through continuous analytic verification
- ensures traceability by linking machine-readable requirements to the tests that verify them and the system elements that implement them

In the Joint Multi-Role Rotorcraft (JMR) program, contractor teams are piloting Architecture-Centric Virtual Integration Practice (ACVIP) as a key technology on a mission-critical system architecture. Our ongoing partnership with JMR provides an excellent transition pathway for our research results and influences the Army's Future Vertical Lift (FVL) program.

The following individuals also provided key contributions to this work:

- CMU SEI: Peter Feiler, Dave Gluch, Aaron Greenhouse, Jerome Hugues, Lutz Wrage, and Joe Seibel
- Kansas State University: John Hatcliff, Eugene Vasserman, Robby, Hari Thiagarajan, and Jason Belt

IN CONTEXT: THIS FY2018–20 PROJECT

- extends AADL with a standardized security-documentation format, builds example safety and security patterns into the OSATE toolbench, and uses novel program-slicing technology developed by our partners at Kansas State University
- aligns with the SEI's technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet-unseen adversary capabilities

Untangling the Knot

Enabling Rapid Software Evolution

Problem

To quickly deliver new capabilities and take advantage of new technologies, DoD needs the ability to efficiently restructure software for common scenarios like:

- migrating a capability to the cloud
- harvesting software for reuse
- containerizing software

One recent anecdote estimates the effort to isolate a capability from the platform at 14,000 staff hours just for development.

Solution

Create an automated assistant that rapidly refactors software to support software isolation goals that enable software evolution.

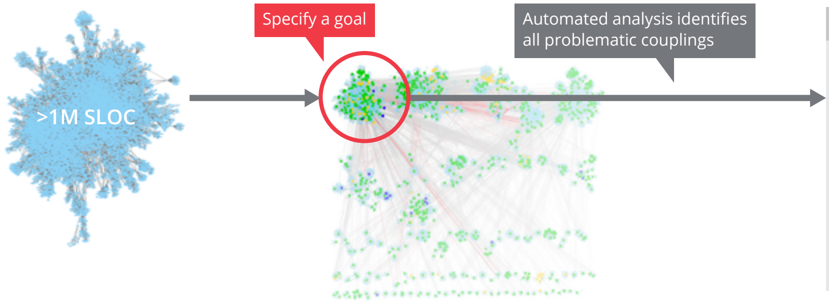
- Allows users to specify project-specific goals.
- Uses genetic algorithms to recommend refactorings.
- Navigates multiple, competing objectives.

Intended Outcomes (FY19–21)

- Refactoring recommendations outperform those based only on quality metrics, reducing problematic couplings by at least 75%.
- Our automation reduces the time to restructure software to 1/3 of the time compared to manual effort.

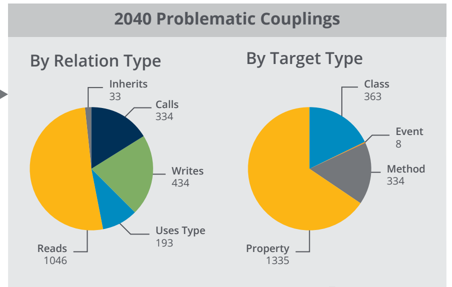
Read more about our vision:
J. Ivers, I. Ozkaya, R. L. Nord, C. Seifried, **Next Generation Automated Software Evolution: Refactoring at Scale**. 2020. *28th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20). ACM, Virtual Event, USA.*

Automated refactoring can improve the structure of existing software in **1/3 of the time** it takes to manually refactor.



Specify a goal

Automated analysis identifies all problematic couplings



Select Objectives

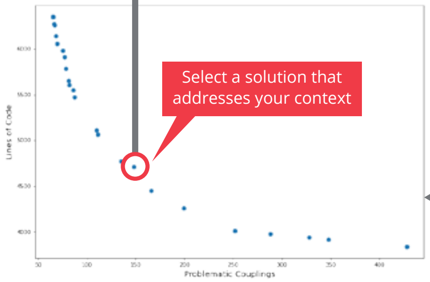
- minimize problematic couplings
- minimize code changes
- maximize code quality
- …

Analysis reduces to unique targets

Target Type	PC Count	# Unique Targets
Class	363	15
Event	8	1
Method	334	51
Property	1335	77
	2040	**144**

Out of 1M+ SLOC, changes should focus on **only 24 classes**

Prototype uses a multi-objective genetic algorithm to generate a set of Pareto optimal solutions (recommendations)



Select a solution that addresses your context

```
Best solution:
Fitness = 33
Step 1: MoveStaticProperty (Duplicati.Server.Strings.Program.PortablemodeCommandDe-
scription, Duplicati.Server.Program)
Step 2: MoveClass (Duplicati.Library.AutoUpdater.AutoUpdateSettings)
Step 3: MoveClass (Duplicati.Library.Utility.WorkerThread<>)
Step 4: MoveInterface (Duplicati.Server.Serialization.Interface.ISchedule)
Step 5: MoveInterface (Duplicati.Server.Serialization.Interface.IBackup)
Step 6: MoveInterface (Duplicati.Server.Serialization.Interface.ISetting)
Step 7: MoveClass (Duplicati.Server.Strings.Program)
Step 8: MoveClass (Duplicati.Server.Database.Backup)
Step 9: MoveClass (Duplicati.Library.Localization.Short.LC)
Step 10: MoveClass (Duplicati.Server.Database.Notification)
Step 11: MoveClass (Duplicati.Server.WebServer.IndexHtmlHandler)
Step 12: MoveClass (Duplicati.Server.WebServer.RESTMethods.RequestInfo)
Step 13: MoveClass (Duplicati.Server.Database.TempFile)
Step 14: MoveClass (Duplicati.Server.WebServer.BodyWriter)
Step 15: MoveClass (Duplicati.Library.Interface.CommandLineArgument)
Step 16: MoveInterface (Duplicati.Library.Interface.ICommandLineArgument)
Step 17: MoveClass (Duplicati.Server.EventPollNotify)
Step 18: MoveClass (Duplicati.Library.Utility.Utility)
Step 19: MoveClass (Duplicati.Library.Common.Platform)
Step 20: MoveClass (Duplicati.Server.LiveControls)
Step 21: MoveClass (Duplicati.Library.Interface.Strings.DataTypes)
Step 22: MoveClass (Duplicati.Library.Utility.Strings.Utility)
Step 23: MoveInterface (Duplicati.Server.Serialization.Interface.IFilter)
Step 24: MoveInterface (Duplicati.Library.Localization.ILocalizationService)
Step 25: MoveClass (Duplicati.Server.Database.Schedule)
Step 26: MoveInterface (Duplicati.Server.WebServer.RESTMethods.IRESTMethodPOST)
Step 27: MoveClass (Duplicati.Library.Utility.Sizeparser)
Step 28: MoveStaticMethod (Duplicati.Library.Utility.Strings.Sizeparser.InvalidSizeVal-
ueError, Duplicati.Library.Utility.Sizeparser)
Step 29: MoveStaticMethod (Duplicati.Library.Utility.Timeparser.ParseTimeSpan,
Duplicati.Server.Database.Connection)
Step 30: MoveClass (Duplicati.Library.Interface.UserInformationException)
Step 31: MoveClass (Duplicati.Library.Interface.Strings.CommandLineArgument)
Step 32: MoveClass (Duplicati.Server.UpdatePollThread)
Step 33: MoveClass (Duplicati.Library.AutoUpdater.UpdateInfo)
Step 34: MoveClass (Duplicati.Server.Strings.Server)
Step 35: MoveClass (Duplicati.Library.Common.IO.Util)
Step 36: MoveInterface (Duplicati.Library.Utility.IFilter)
Step 37: MoveStaticProperty (Duplicati.Library.AutoUpdater.UpdaterManager.InstalledBas-
eDir, Duplicati.Server.Program)
Step 38: MoveInterface (Duplicati.Library.Common.IO.ISystemIO)
Step 39: MoveStaticField (Duplicati.Library.AutoUpdater.UpdaterManager.BaseVersion,
Duplicati.Library.AutoUpdater.AutoUpdateSettings)
Step 40:  MoveClass (Duplicati.Server.Serialization.SettingsCreator)
```

Our prototype can help with common evolution scenarios:

Scenario
Gather data to assess the difficulty associated with project-specific goals as input to funding decisions.

Maturity
Available now (TRL 4)

Expected Results
Enumeration of problematic couplings, their locations, and types potentially impacted by proposed change as data to inform cost estimates.

Scenario
Compare the difficulty of different refactoring approaches.

Maturity
Available now (TRL 4)

Expected Results
Enumeration of problematic couplings, their locations, and types potentially impacted by proposed change as data to inform cost estimates.

Scenario
Automatically refactor software to isolate software and speed its evolution.

Maturity
Ready for pilot application in 3–6 months

Expected Results
Recommended refactorings that enable the proposed change address multiple criteria.

Contact us at info@sei.cmu.edu if you are interested in partnering with us.

Principal Investigator

MR. JAMES IVERS
Principal Engineer

Carnegie Mellon University
Software Engineering Institute

CHRIS SEIFRIED
Associate Engineer

Carnegie Mellon University
Software Engineering Institute

CARLY JONES
Data Analytics Intern

Carnegie Mellon University
Software Engineering Institute

UNTANGLING THE KNOT: ENABLING RAPID SOFTWARE EVOLUTION

Software-reliant systems need to evolve over time to meet new requirements and take advantage of new technology. However, all too often the structure of software becomes too complicated to allow rapid and cost-effective improvements. This challenge is common in long-lived DoD systems and not uncommon even in newer systems, which makes isolating a collection of functionality for use in a new context, or clean replacement by an improved version, difficult. Software refactoring can facilitate such changes, but can require tens of thousands of staff hours.

This project aims to use AI techniques to create software engineering automation to recommend a set of refactorings that isolates functionality from its tangle of system dependencies. We aim to reduce the time required for this kind of architecture refactoring by two-thirds. In one DoD example, a contractor estimated 14 thousand hours of software development work alone (excluding integration and testing) to isolate a mission capability from the underlying hardware platform. If successful, our work would reduce the development time required to less than 5 thousand hours.

Our solution combines advances in search-based software engineering with static code analysis and refactoring knowledge. It is unique in its focus on mission-relevant goals as opposed to improving general software metrics. This goal is incorporated in genetic algorithms through fitness functions that guide the search to solutions for the project-specific goal. The search algorithm relies on a representation derived from static code analysis and uses formalizations of refactorings as operators to apply during search.

This work has broad implications for moving existing software to modern architectures and infrastructures, such as service-based, microservice, cloud environments, and containers. It also addresses a pervasive research challenge in improving automated support for architecture refactoring tasks.

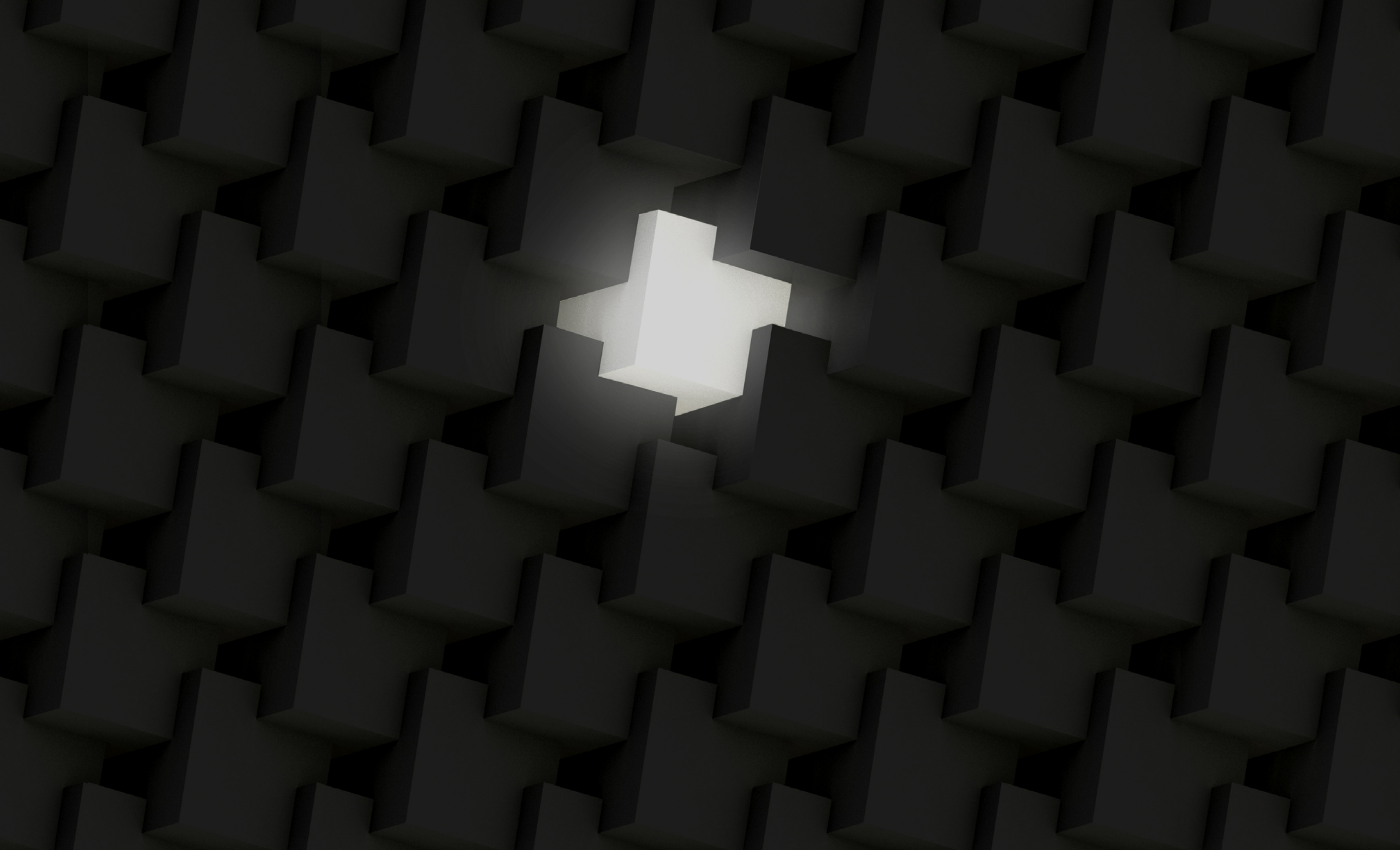
IN CONTEXT: THIS FY2019–21 PROJECT

- builds on prior DoD line-funded research in software architecture analysis, static code analysis, and identifying technical debt
- aligns with the CMU SEI technical objective to make software delivery timely so that the cadence of acquisition, delivery, and fielding is responsive to and anticipatory of the operational tempo of DoD warfighters
- addresses a widespread, recurring need in software organizations: as requirements and technology are never frozen in time, the need to adapt working software to new contexts is likely to remain a common need across many software systems

SECTION 3

Codify Fully Integrated CI/CD Practices

CMU SEI seeks to fully integrate continuous integration/ continuous delivery (CI/CD) process across the entire acquisition lifecycle to give the DoD reduced cost, traceability through the acquisition phases, and faster deployment of incremental capability.



TwinOps

Digital Twins Meet DevOps

Introduction

Cyber-Physical Systems (CPS) exhibit multiple engineering, verification and validation (V&V), and testing challenges. In this project, we aimed at reducing the time to get first test results by leveraging state-of-the-art system and software engineering approaches.

TwinOps explored the interplay between three core technologies:

- *Model-Based Engineering (MBE)*: model-based engineering relies on models as first-class abstraction of a system to support engineering activities;
- *DevOps:* an organizational effort to support continuous delivery of software through a better coupling between (Dev)elopment and (Op)erations activities;
- *Digital Twins*: an infrastructure to support system monitoring and diagnosis in real-time and enable continuous system improvement.

Achievements

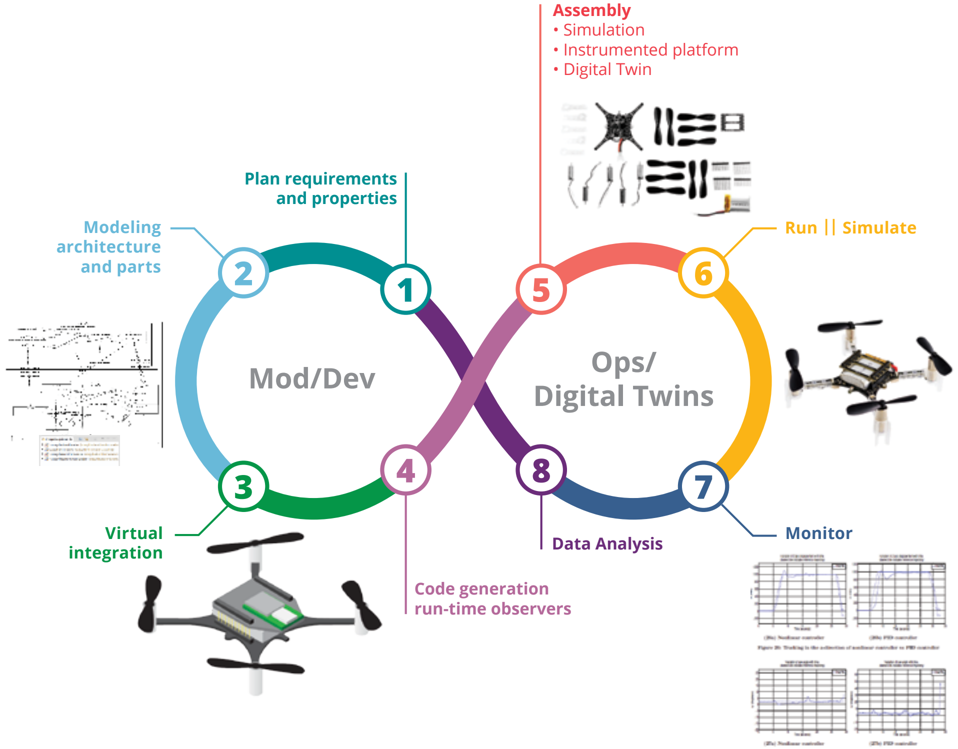
SEI delivers a ModDevOps exemplar

> ModDevOps extends DevOps through MBE and its V&V and code generation capabilities. We demonstrate how MBE enables rapid system prototyping through a DevOps cycle.

SEI enhances analysis and testing process for systems architects who build software-intensive CPS with the *TwinOps* process

> TwinOps builds on ModDevOps and Digital Twins to collect data on a system at runtime, and compare it to other engineering artifacts: model simulation and analysis. This comparison enables rapid system diagnosis.

ModDevOps adds Model-Based early V&V and code generation to DevOps automation.



Assembly
- Simulation
- Instrumented platform
- Digital Twin

Plan requirements and properties

Modeling architecture and parts

Mod/Dev

Ops/ Digital Twins

Run || Simulate

Virtual integration

Code generation run-time observers

Data Analysis

Monitor

Approach

ModDevOps is defined as an abstract process using OMG SysML. This captures the key steps of the process as a collection of use cases, block diagrams, and activities.

⇒ Each project will adapt ModDevOps to its own problem/solution spaces

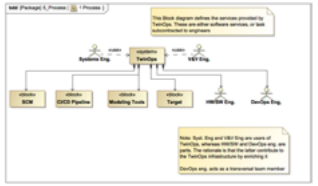
TwinOps is an instance of ModDevOps tailored for CPS. It combines

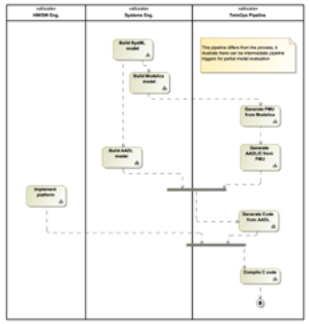
- AADL modeling for CPS architecture
- Simulink or C for the functional code
- Modelica for modeling the environment

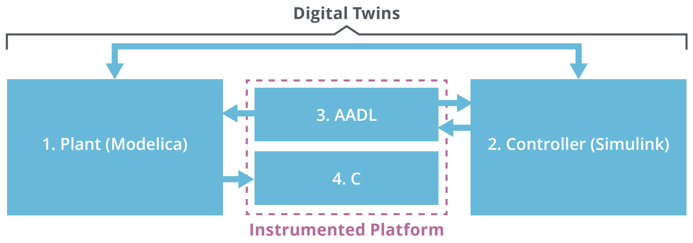
The definition of the process as SysML models guides engineering phases:

- Orchestrate modeling, code generation, and compilation
- Continuous integration/continuous deployment used to deploy the system on the target, using Azure IoT cloud-based solutions

Code generation from model enables multiple scenarios: deployment on target and digital twins to support various operating scenarios.







Digital Twins

1. Plant (Modelica)

3. AADL

4. C

2. Controller (Simulink)

Instrumented Platform

Principal Investigator

DR. JEROME HUGUES
Senior Architecture
Researcher

Carnegie Mellon University
Software Engineering Institute

TWINOPS: DIGITAL TWINS MEETS DEVOPS

The engineering of cyber-physical systems (CPS) requires a large set of expertise to capture the system requirements and derive a correct solution. Key issues, such as sensor timing jitter, bias, or imprecise component characterization (in the functional, timing, or safety viewpoints) are still only discovered during testing or after the system has been deployed. Recent accidents involving airliners and autonomous vehicles were in part caused by imprecise characterizations of system behavior, causing a significant and costly rework of the software. Model-based engineering (MBE) and DevOps aim to efficiently deliver software with increased quality. In this project, we have proposed new ways to combine them.

Model-based engineering relies on models as first-class artifacts to analyze, simulate, and ultimately generate parts of a system. DevOps focuses on software engineering activities, from early development to integration, and then improvement through the monitoring of the system at runtime. We claim these can be efficiently combined to improve the engineering process of CPS.

LENS TwinOps proposes a process that unifies MBE, digital twins, and DevOps practice in a uniform workflow. TwinOps leverages several best practices in MBE and DevOps for the engineering cyber-physical systems. We illustrate our contribution using a digital twins case study to illustrate TwinOps benefits, combining AADL and Modelica models, and an IoT platform.

This project extends our line of research aimed to improve both the state of the art and the state of practice of designing and analyzing cyber-physical systems. Through the Architectural Analysis Design Language (AADL) and the Architecture Centric Virtual Integration Process (ACVIP), we addressed both system and software concerns (safety, security, performance, and code generation). Analytical frameworks based on AADL evaluate system integrability prior to the performance of actual integration testing activities.

The following SEI researchers also provided key contributions to this work: Anton Hristozov, John Hudak, and Joe Yankel

IN CONTEXT: THIS FY2020 PROJECT

- builds on the foundations of digital twins and DevOps as well as on prior research on AADL and the Open Source AADL Tool Environment (OSATE)
- aligns with the CMU SEI technical objectives to 1) bring capabilities through software that make new missions possible or improve the likelihood of success of existing ones and to be trustworthy in construction and implementation, and 2) be resilient in the face of operational uncertainties, including known and yet-unseen adversary capabilities

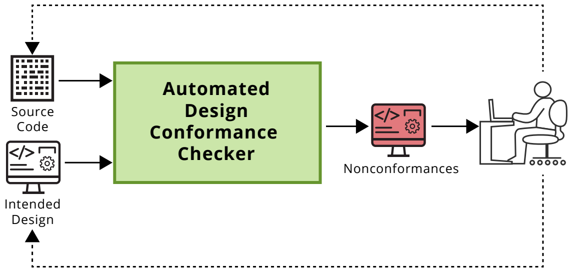
Automated Design Conformance during Continuous Integration

Problem

Code often does not conform to designs, undermining properties such as extensibility and composability. Late detection increases cost and delays delivering capability to the field.

Solution

Use code analysis, software architecture knowledge, and machine learning to automatically extract design as implemented in the code and check conformance with the intended design.

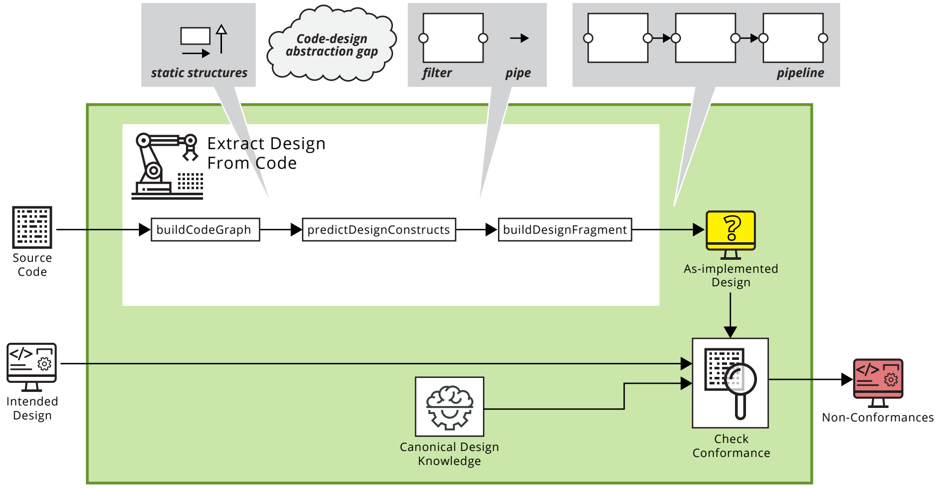


Intended Impact (FY20–22)

- Recommendations correctly identify nonconformance and detect at the commit that introduces nonconformance.
- Automation enables early detection and allows remediation before the violation gets "baked in" to the implementation.
- Detection of nonconformances allows program managers to hold developers (contractor or organic) accountable.

Read more about our approach:
Nord (2020). *Using Machine Learning to Detect Design Patterns,* SEI Blog.

An **automated design conformance checker** integrated into a continuous integration workflow will reduce time to detect violations from months or years to hours.

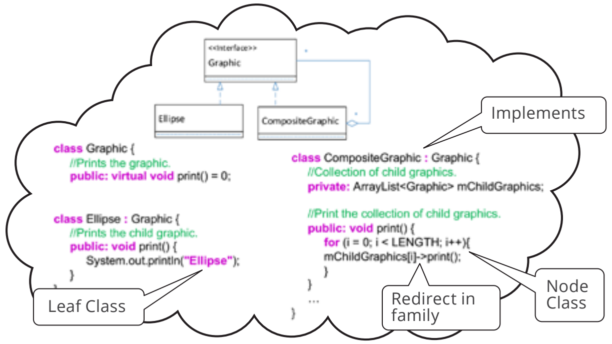


Automated Design Conformance Checker

Approach

Our solution builds on code analysis, software architecture, machine learning, and continuous integration**.** We ingest a software repository and **build a graph** representation of the code structure based on code analysis. We apply machine learning to bridge the abstraction gap to **extract design constructs** from the code. We **build the design fragments** that comprise the as-implemented design. The as-implemented design can then be **checked for conformance** against the intended design at each code commit during **continuous integration.**

The central research of this project uses **machine learning** to extract features by recognizing abstractions commonly used in software architecture in C++ source code.



```
class Graphic {
    //Prints the graphic.
    public: virtual void print() = 0;

class Ellipse : Graphic {
    //Prints the child graphic.
    public: void print() {
        System.out.println("Ellipse");
    }
}
```

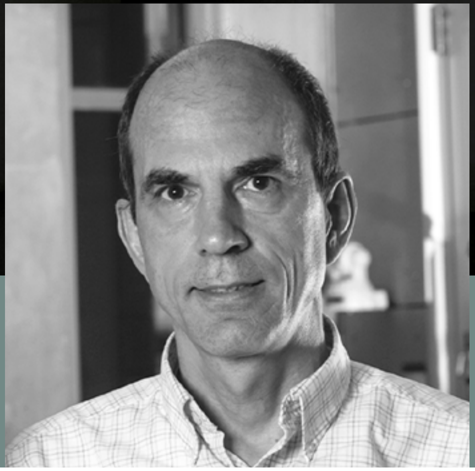
```
class CompositeGraphic : Graphic {
    //Collection of child graphics.
    private: ArrayList<Graphic> mChildGraphics;

    //Print the collection of child graphics.
    public: void print() {
        for (i = 0; i < LENGTH; i++){
        mChildGraphics[i]->print();
        }
    }
    ...
}
```

Code-Design Abstraction Gap

Feature engineering is key to extracting design and bridging the gap. Structural and behavioral features link elements (e.g., classes) though relations (e.g., inheritance, method call).

Our prototype advances the state of the art in applying machine learning to software engineering tasks and aligns with SEI strategic focus areas of timely and trustworthy software by introducing automation into the development and acquisition lifecycle.



Principal Investigator

DR. ROBERT (ROD) NORD

Principal Member of the Technical Staff

Carnegie Mellon University Software Engineering Institute

AUTOMATED DESIGN CONFORMANCE DURING CONTINUOUS INTEGRATION

To reduce the time needed to field capabilities and to lower lifecycle costs, the DoD has instructed program managers to consider a modular open systems approach (MOSA). MOSA promotes extensibility and composability of platforms through technical standards such as the Future Airborne Capability Environment (FACE). However, a gap exists in verifying whether implemented capabilities satisfy the design constraints of a reference architecture such as FACE.

This project is creating an automated conformance checker that can be integrated into the continuous integration workflow to detect and report nonconformances in hours instead of the months or years that it takes to discover these problems today. This technology will correctly identify design nonconformances with precision greater than 90%.

Our solution builds on code analysis, software architecture, machine learning, and continuous integration. The central research of this project is using machine learning to recognize abstractions commonly used in software architecture in C++ source code. We are focusing on detecting nonconformance with design approaches that are essential to achieving the goals of MOSA and common platforms: communication over distributed interfaces, isolation and encapsulation of functionality, and separation of concerns.

The conformance checker will benefit developers and program managers. Developers can detect problems continuously and near the time when they are introduced, allowing faster and more economical realignment of implementation and design. Program managers can hold developers (contractor or organic) accountable for delivering sustainable systems.

IN CONTEXT: THIS FY2020–22 PROJECT

- advances the state of the art in applying ML to software engineering tasks
- aligns with CMU SEI’s strategic focus areas of timely and trustworthy software by introducing automation into the development and acquisition lifecycle

Causal Models for Software Cost Prediction & Control (SCOPE)

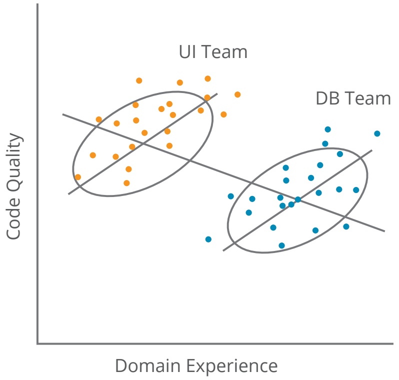
How can we control costs in software development and sustainment? We are collaborating with other researchers to apply causal learning to learn how.

DoD Problem

- DoD leadership needs to understand why software costs so much.
- DoD program offices need to know where to intervene to control software costs.

Why Causal Learning?

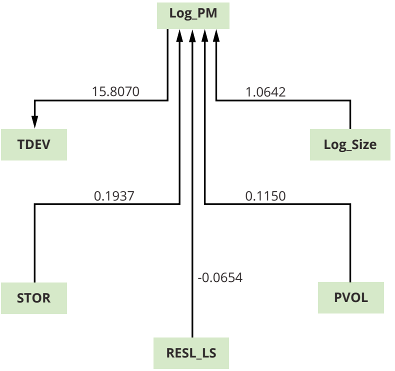
To reduce costs, the causes of an outcome (good or bad) need to be considered. Correlations are insufficient in part due to Simpson's Paradox. For example, in the figure below, if you did not segment your data by team (User Interface [UI] and Database [DB]), you might conclude that increasing domain experience reduces code quality (downward line); however, within each team, it's clear that the opposite is true (two upward lines). Causal learning identifies when factors such as team membership explain away (or mediate) correlations, and it works for much more complicated data sets too.



Simpson's Paradox as Applied to UI/DB Data

Causal learning reduces costs.

Recent Results



COCOMO® II Mini-Cost Estimation Model

COCOMO® II – Effort Drivers

Size (SLOC), Team Cohesion, Platform Volatility, Reliability, Storage Constraints, Time Constraints, Product Complexity, Process Maturity, Architecture/Risk Resolution (RESL)

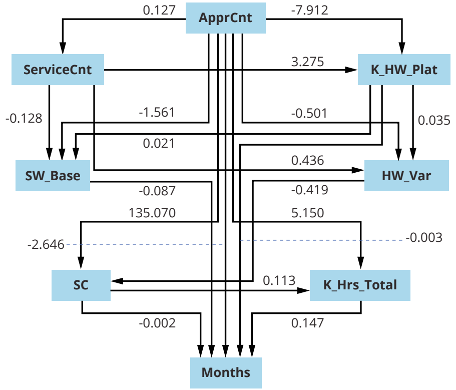
COCOMO® II – Schedule Drivers

Size (SLOC), Platform Experience, Schedule Constraint, and Effort

COSYSMO 3.0 – Effort Drivers

Size and Level of Service Requirements

After identifying which of over 40 factors directly drive costs, we used Tetrad to generate mini cost-estimation models that fit well. (In the figure, RESL_LS is the product of RESL and Log_Size.)



Consensus Graph for U.S. Army Software Sustainment

A U.S. Army Sustainment data set was segmented into **(Superdomain, ACAT Level)** pairs resulting in five sets of data to search and estimate. Splitting addressed high fan-out for common causes, which can lead to structures typical of Simpson's Paradox. A consensus graph (see above) was built from the resulting five **searched and** fitted models.

For consensus estimation, the data from individual searches was pooled with previously excluded data because of missing values. The resulting 337 releases were used to estimate the consensus graph using Mplus with Bootstrap in estimation.

There was no cherry picking or re-do's—this model is a direct out-of-the-box estimation, achieving good model fit on the first try.

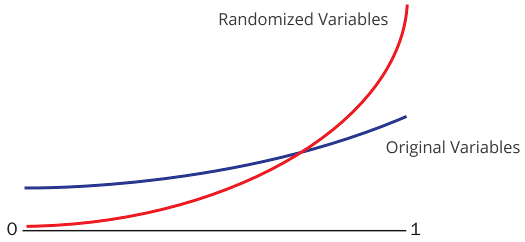
Acknowledgments

Our thanks to Anandi Hira and Jim Alstad of USC; and Cheryl Jones and her team at U.S. Army AFC-CCDC and DASA-CE.

Our Solution

Our approach to causal inference is **principled** (i.e., no cherry picking) and **robust** (to outliers). This approach is **especially useful** for small samples—when the number of cases is < 5-10 times the number of variables.

1. Inject **null variables** by appending an independently randomized copy of each original variable.
2. Search (FGES or PC with default settings) with Bootstrap to determine each edge's **Probability of No Edge (PNE)** across the search.
3. Set a **threshold (10th percentile)** among the edges involving a null variable. (Of edges involving a null variable, 90% have a PNE exceeding that threshold.) Then drop the null variables but apply this same threshold to determine which edges to keep among the original variables.



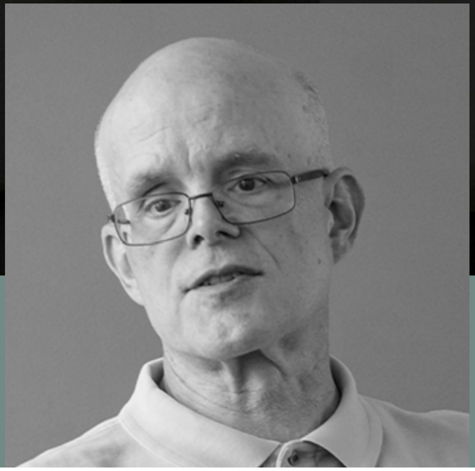
Probability of No Edge

Summary

Causal models offer better insight for program control than models based on correlation. Knowing which factors drive which program outcomes is essential to sustain the warfighter by providing high-quality, secure software in a timely and affordable manner.

For More Information

For more information, including causal analyses of other data sets, see our SCOPE Project website.



Principal Investigator

DR. MICHAEL KONRAD
Principal Researcher

Carnegie Mellon University
Software Engineering Institute

DR. WILLIAM NICHOLS
Senior Member of the Technical Staff

Carnegie Mellon University
Software Engineering Institute

INTEGRATED CAUSAL MODEL FOR SOFTWARE COST PREDICTION & CONTROL (SCOPE)

Correlation is not causation, and yet what we experience often confuses the two. This extends to software engineering research, where changes in project stakeholders, requirements, architecture, solution approach, personnel, and development platform and practices are shown to correlate with improved project outcomes; and yet, in reality, it might be only a few factors that directly drive project cost and schedule. Research in other fields (e.g., medicine) has shown causal models are superior to traditional statistical models because, by identifying truly causal factors, proactive control of a system or situation is possible.

How would we build a causal model for software project costs? Until recently, we did not have a way to obtain or validate causal models from primarily observational data, a challenge shared across nearly all systems and software engineering research, where randomized control trials are nearly impossible. The SCOPE project will apply recent advances in causal modeling algorithms and tools to project data to identify, measure, and test causality. [Glymour 2019]

In this project, which concluded at the end of FY20, we achieved the following:

- identified, from among 40 software engineering and systems engineering factors, which are more likely to improve program costs ("control knobs" for a program dashboard)
- completed the first stitching (and estimation) of a software engineering dataset, resulting in an integrated causal model covering multiple domains and acquisition category (ACAT) levels for determining the number of software changes released
- identified cognitive fog and system behavior stability as causes of program failure (from among 30 measures of project complexity)
- concluded that communication misbehaviors increase the amount of Common Vulnerabilities & Exposures (CVE) remediation effort, not just immediately, but for a longer time (open source projects)
- concluded that variation within programmers is approximately the same as between programmers; thus, rather than trying to hire the 10X programmer, organizations would do better to invest in training, processes, platforms and tools
- completed updates to Quantifying Uncertainty in Early Lifecycle Costs (QUELCE) for improved cost estimation
- transitioned causal discovery capability to the U.S. Army and University of Southern California cost estimation researchers
- developed a methodology for applying causal discovery to small datasets to improve robustness and reduce cherry picking of results

Thus, an immediate benefit of this work is the identification of causal factors that provide a basis for controlling program costs. A longer-term benefit is the use of causal models in negotiating software contracts, designing policy and incentives, and informing could/should cost and affordability efforts.

Key contributors to this project were team members Rhonda Brown, Michele Falce, Madelyn Glymour, Michael Konrad, Chris Miller, William Nichols, Bob Stoddard, Bryar Wassum, and Dave Zubrow.

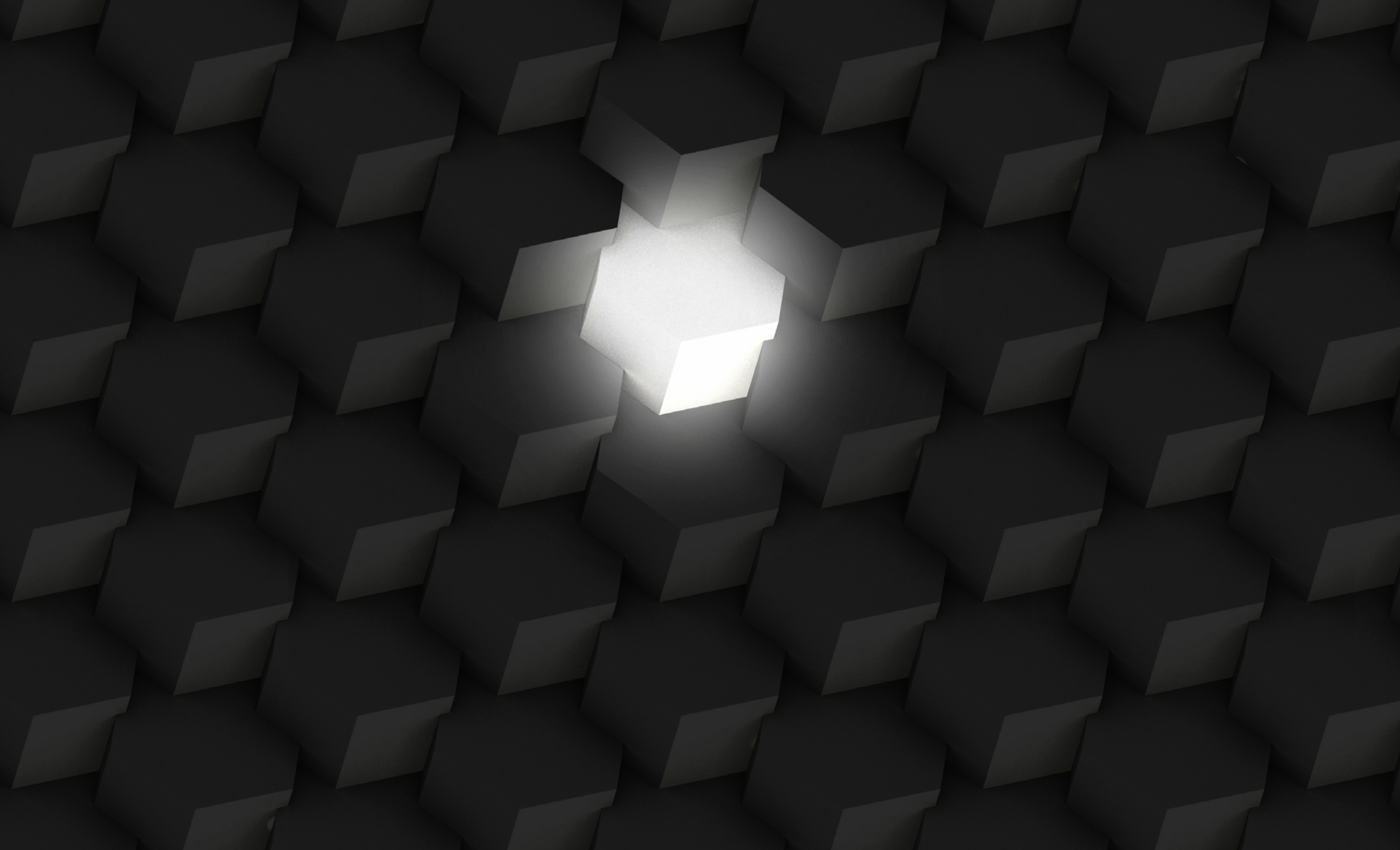
IN CONTEXT: THIS FY2018–20 PROJECT

- contributes to a longer-term research roadmap to build causal models for the software developer, software development team, organization, and acquirer
- aligns with the CMU SEI technical objective to make software affordable such that the cost of acquisition and operations, despite increased capability, is reduced and more predictable

SECTION 4

Improve Designed-In Resilience

CMU SEI seeks to increase the trustworthiness and confidence in DoD platforms through practices that imbue rigorous designed-in resilience properties.



Automated Code Repair to Ensure Memory Safety

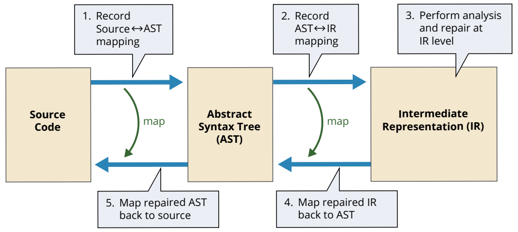
Problem

Software vulnerabilities constitute a major threat to DoD. Memory violations are among the most common and most severe types of vulnerabilities. Spatial memory vulnerabilities constitute 15% of CVEs in the NIST National Vulnerability Database and 24% of critical-severity CVEs.

Solution

We developed and implemented a technique to automatically repair source code to assure spatial memory safety. Our tool inserts code to abort the program (or call user-specified error-handling code) immediately before a memory violation would occur, preventing exploitation by attackers.

The main technique that we use (fat pointers) has been previously researched to repair code as part of the compilation process. Our work is novel in applying it as a source-code repair, which poses the difficulty of translating the repairs on the intermediate representation (IR) back to source code. The pipeline is shown below:


1. Record Source↔AST mapping
2. Record AST↔IR mapping
3. Perform analysis and repair at IR level
Source Code
map
Abstract Syntax Tree (AST)
map
Intermediate Representation (IR)
5. Map repaired AST back to source
4. Map repaired IR back to AST

Intended Impact

With further development, this technology can be used by DoD to ensure memory safety as part of all software projects with code written in memory-unsafe languages (such as C and C++).

We developed an **automated technique** to repair C source code to **eliminate memory-safety vulnerabilities**.

Figure 1(a): Original Source Code

```
#define BUF_SIZE 256
char nondet_char();

int main() {
    char* p = malloc(BUF_SIZE);
    char c;
    while ((c = nondet_char()) != 0) {
        *p = c;
        p = p + 1;
    }
    return 0;
}
```

Figure 1(b): Repaired Source Code

```
#include "fat_header.h"
#include "fat_stdlib.h"
#define BUF_SIZE 256
char nondet_char();

int main() {
    FatPtr_char p = fatmalloc_char(BUF_SIZE);
    char c;
    while ((c = nondet_char()) != 0) {
        *bound_check(p) = c;
        p = fatp_add(p, 1);
    }
    return 0;
}
```

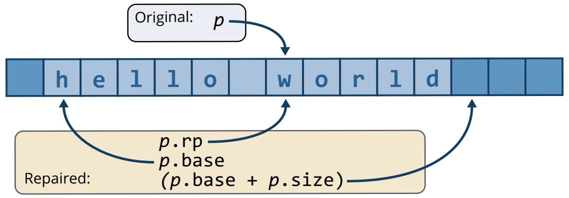
Results

The runtime overhead of our repair is around 50% on bzip2. Our DoD partners said this is too high for many of their use cases. Can we significantly reduce the overhead while still guaranteeing memory safety? Probably not, but automated repair is valuable even if it fixes only the likeliest bugs. To reduce the overhead time, we added an option to insert bounds checks only for memory accesses that are warned about by an external static analyzer. This **reduced the overhead to 6%** on bzip2.

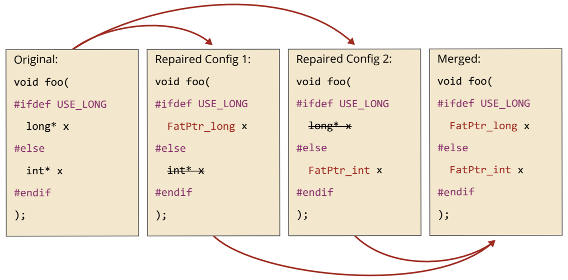
Ensuring spatial memory safety with fat pointers

Our tool replaces raw pointers with **fat pointers**, which are structs that include bounds information in addition to the pointer itself. Before dereferencing a fat pointer, a bounds check is performed. For each pointer type *T**, we define a new struct:

```
struct FatPtr_T {
  T*      rp;    /* raw pointer */
  char*   base;  /* of mem region */
  size_t  size;  /* in bytes */
};
```



To preserve compatibility with third-party binary libraries, we identify and refrain from fattening any pointers stored in heap memory that is reachable by external binary code. The C preprocessor can include or exclude pieces of C code depending on the configuration chosen at compile time. We repair configurations separately and merge the results:



Original:

```
void foo(
#ifdef USE_LONG
  long* x
#else
  int* x
#endif
);
```

Repaired Config 1:

```
void foo(
#ifdef USE_LONG
  FatPtr_long x
#else
  ~~int* x~~
#endif
);
```

Repaired Config 2:

```
void foo(
#ifdef USE_LONG
  ~~long* x~~
#else
  FatPtr_int x
#endif
);
```

Merged:

```
void foo(
#ifdef USE_LONG
  FatPtr_long x
#else
  FatPtr_int x
#endif
);
```

Principal Investigator

DR. WILL KLIEBER
Software Security Engineer

Carnegie Mellon University
Software Engineering Institute

AUTOMATED CODE REPAIR TO ENSURE MEMORY SAFETY

Software vulnerabilities constitute a major threat to the DoD, and memory violations are among the most common and most severe types of vulnerabilities. In recent years, spatial memory violations (e.g., buffer overflows) constituted 24% of critical-severity Common Vulnerabilities & Exposures in the NIST National Vulnerability Database.

We have designed and implemented a technique for automatically repairing all potential violations of spatial memory safety in source code. For this, we do not need to solve the challenging problem of distinguishing false alarms from true vulnerabilities: we can simply apply a repair to all potential memory-safety vulnerabilities, at a cost of runtime overhead. If the runtime overhead turns out to be too high, it can be reduced by limiting repairs to those lines of code that are flagged as likely vulnerabilities by an external static analyzer.

IN CONTEXT: THIS FY2018–20 PROJECT

- extends prior DoD line-funded research in automated repair of code for integer overflow and the inference of memory bounds
- is related to CMU SEI technical work into advancements based on the Pharos static binary analysis framework, vulnerability discovery, and code diversification to avoid detection of vulnerabilities by adversaries
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet unseen adversary capabilities

KalKi: High Assurance Software-Defined IoT Security

Problem

Despite the DoD's current use of Internet of Things (IoT) devices in supervisory control and data acquisition (SCADA) systems, and its interest in using such devices in tactical systems, adoption of IoT has been slow, mainly due to security concerns (e.g., reported vulnerabilities, untrusted supply chains). At the same time, the DoD recognizes the rapid pace at which the IoT commercial marketplace is evolving, and its urgency to embrace commodity technologies to match its adversaries.

Solution

Move part of security enforcement to the network to enable the integration of IoT devices into DoD systems, even if the IoT devices are not fully trusted or configurable, by creating an IoT security platform that is provably resilient to a collection of prescribed threats.

The "Software-Defined" Aspect

Use software-defined networking (SDN) and network function virtualization (NFV) to create a highly dynamic IoT security platform.

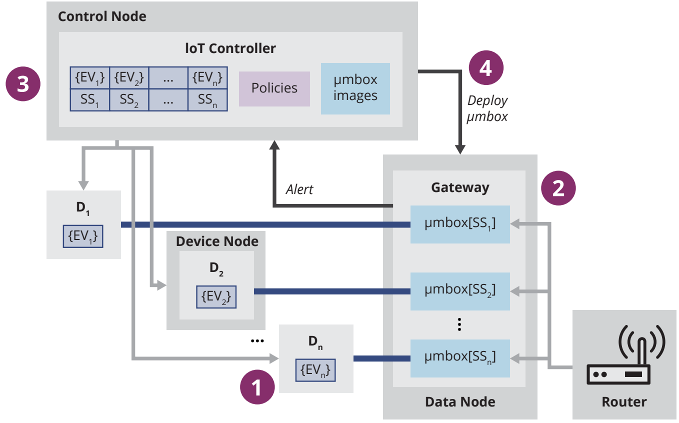
The "High Assurance" Aspect

Use the open-source uber eXtensible Micro-Hypervisor Framework (uberXMHF) to develop secure extensions that enforce security properties of critical elements of the software-defined IoT security platform at runtime, on commodity platforms.

KalKi IoT Security Platform Features

- Has flexible policies to define states, transitions and actions.
- Can protect from both cyber and kinetic attacks.
- Uses different network defenses for each device and state.
- Adapts to device-specific vulnerabilities or limitations.

The KalKi IoT Security Platform **enables the integration of IoT devices** into DoD systems, even if the IoT devices are **not fully trusted** or configurable.



1. Each IoT Device D senses/controls a set of environment variables EV
2. Network traffic to/from each device is tunneled through µmboxes that implement the desired network defense for the current system state
   D_1 µmbox[SS_1] = Firewall
   D_2 µmbox[SS_2] = IPS, …
3. IoT Controller maintains a shared statespace composed of {EV} and security state (SS) for each device
   SS= {Normal, Suspicious, Attack}
4. Changes in the shared statespace are evaluated by policies and may result in the deployment of new µmbox(es)

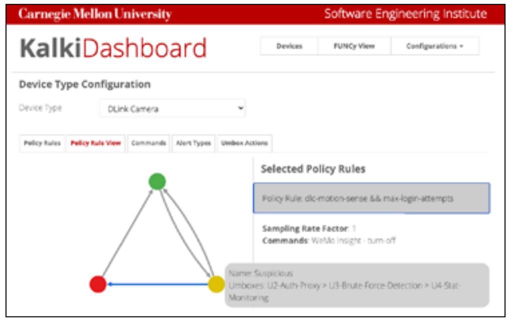
Security sensitive areas of the system are protected by the uberXMHF extensible and performant micro-hypervisor framework that provides three key runtime capabilities:

a. isolation,
b. mediation,
c. attestation.

- The micro-hypervisor verifies the integrity of the µmbox images when they are loaded, to ensure that each device has the correct network defenses.
- Signing network packets ensures that they are routed through the proper µmboxes for each specific device in the Data Node.

Year 3 Highlights

1. The new version of the platform prototype using docker containers showed significant performance and scalability improvements—threat reaction time is 3 seconds (90% improvement) with support for up to 125 connected devices (80% improvement).
2. User interface improvements to the Dashboard UI significantly reduce the time and complexity of adding new IoT devices, especially with respect to policy definition.



Device Type Configuration

Device Type: DLink Camera

Selected Policy Rules

Policy Rule: dlc-motion-sense && max-login-attempts

Sampling Rate Factor: 1
Commands: WeMo Insight - turn-off

Name: Suspicious
Umboxes: U2-Auth-Proxy > U3-Brute-Force-Detection > U4-Stat-Monitoring

3. Architecture changes enable the system to adapt to different network layouts and to be deployed on low-cost hardware such as a Raspberry Pi.
4. We created a formal model of our security architecture using the Alloy modeling language and successfully validated its designed-in resilience properties.
5. We demonstrated that the architecture provides intrinsic security against a broad spectrum of attacks, including nine published attacks against such software-defined architectures.
6. Kalki code is available as open-source on Github to invite the community to test or adapt the platform. KalKi platform: https://github.com/SEI-TAS/kalki-node-setup/wiki ; uberXMHF microhypervisor: https://uberxmhf.org/

Principal Investigator

MR. SEBASTIAN ECHEVERRIA
Member of the Technical Staff/Senior Engineer

Carnegie Mellon University Software Engineering Institute

INVESTIGATING THE FEASIBILITY OF HIGH-ASSURANCE SOFTWARE-DEFINED IOT SECURITY

Despite its use of Internet of Things (IoT) devices in supervisory control and data acquisition (SCADA) systems and its interest in using such devices in tactical systems, the DoD has been slow to adopt IoT. In particular, the DoD is reluctant to use commodity IoT devices, especially in tactical systems, because of untrusted supply chains and a growing amount of reported vulnerabilities in these devices. At the same time, the DoD recognizes the rapid pace at which the IoT commercial marketplace is evolving and its urgency to embrace commodity technologies, to match its adversaries.

Our proposed solution moves part of security enforcement to the network to enable the integration of IoT devices into DoD systems, even if the IoT devices are not fully trusted or configurable, by creating an IoT security infrastructure that is provably resilient to a collection of prescribed threats. It uses

- software-defined networking (SDN) and network function virtualization (NFV) to create a highly dynamic IoT security framework
- überSpark (a framework for building secure software stacks) to incrementally develop and verify security properties of elements of the software-defined IoT security infrastructure [Vasudevan 2016]

IN CONTEXT: THIS FY2018–20 PROJECT

- builds on prior CMU SEI technical work in the mobile communication and computing needs of edge users and the authentication and authorization for IoT devices
- draws from our collaboration with CMU researchers and sponsored engagements to reduce risk through architecture analysis
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties

Using All Processor Cores While Being Confident about Timing

Today, almost all computers use multicore processors. Unfortunately, satisfying hard real-time requirements of software executing on such computers is challenging because the timing depends on how resources in the memory system are shared, and this information is typically not publicly available. This project addresses this problem.

Multicore processors

Today, almost all computers use multicore processors. These computers have many processor cores such that one program can execute on one processor core and another program can execute on another processor core simultaneously (true parallelism). Typically, processor cores share memory. In today's memory system, a large number of resources are used to make memory accesses faster in general but, unfortunately, also make execution time more unpredictable and dependent on execution of other programs (because these other programs use shared resources in the memory system). A simplified view of a multicore processor with the memory system is shown in Figure 1.

Embedded real-time cyber-physical systems

These systems are pervasive in society in general, as shown by the fact that 99% of all processors produced are used in embedded systems. In many of these systems, computing the correct result is not enough; it is also necessary to compute the correct result at the right time.

These methods assume that **one knows** the resources in the memory system; unfortunately, most chip vendors do not make this information available.

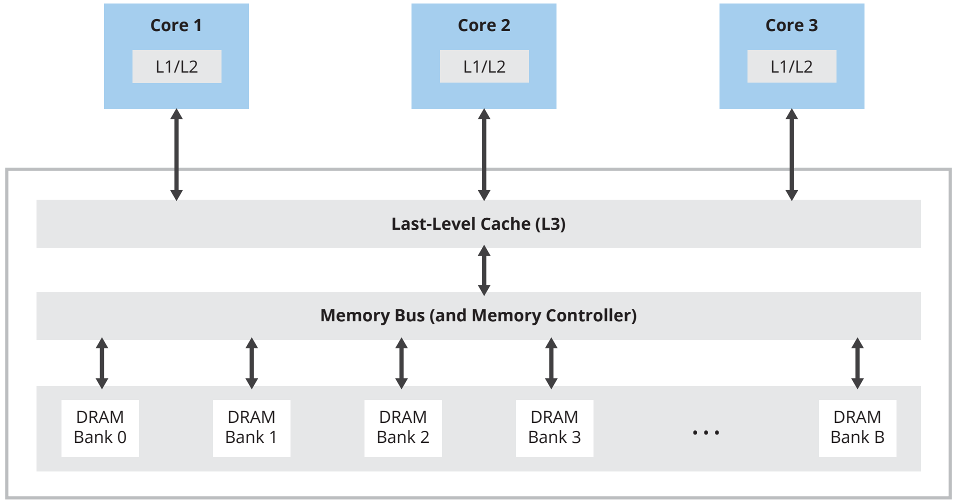


Figure 1: A simplified view of a multicore processor with shared memory

Department of Defense (DoD)

Embedded real-time cyber-physical systems are pervasive in the DoD. Because of the importance of achieving predictable timing, it is common for practitioners to disable all processor cores except one (hence making a multicore processor behave as a single processor system). The importance of timing was recently stressed by AMRDEC's S3I director [1]:

> *"The trick there, when you're processing flight critical information, it has to be a deterministic environment, meaning we know exactly where a piece of data is going to be exactly when we need to—no room for error,"* [Jeff] Langhout says. *"On a multi-core processor there's a lot of sharing going on across the cores, so right now we're not able to do that."*

Current solutions

The current state of the art makes solutions available for managing contention for resources in the memory system and for analyzing the impact of this contention on timing for the case that we know the resources in the memory system.

Problem addressed

In this project, we have addressed the problem of verifying timing of software executing on a multicore processor assuming that we do not know the resources in the memory system.

Results

We have developed a preliminary method—see Andersson, B. et al., "Schedulability Analysis of Tasks with Co-Runner-Dependent Execution Times," ACM Transactions on Embedded Computing Systems, 2018.

[1] "Army still working on multi-core processor for UH-60V," May 2017, Available at https://www.flightglobal.com/news/articles/army-still-working-on-multi-core-processor-for-uh-6-436895/

Principal Investigator

DR. BJORN ANDERSSON
Member of the Technical
Staff/Principal Researcher

Carnegie Mellon University
Software Engineering Institute

USING ALL PROCESSOR CORES WHILE BEING CONFIDENT ABOUT TIMING

Complex, cyber-physical DoD systems, such as aircraft, depend on correct timing to properly and reliably execute crucial sensing, computing, and actuation functions. Any timing failure can have disastrous consequences—a large unexpected delay translating sensor data into actuation can cause system instability and loss of control. What's more, the complexity of today's DoD systems has increased the demand for use of multicore processors, because unicore chips are either unavailable or not up to the task. However, concerns about timing have led to the practice of disabling all processor cores except one.

In this project, we aim to develop a solution to overcome this obstacle. This is a difficult challenge, because timing is determined by many shared resources in the memory system (including cache, memory banks, and memory bus) with complex arbitration mechanisms, some of which are undocumented. The goal of our research is to demonstrate multicore timing confidence by achieving the following sub-objectives:

- Verification. Develop a method for timing verification that does not depend directly on undocumented design qualities and quantities.
- Parameter extraction. Develop a method for obtaining values for parameters in the model of a software system suited for the timing verification procedure mentioned above.
- Configuration. Develop a configuration procedure (such as assigning threads to processor cores or assigning priorities to threads) that takes a model as input and produces a configuration for which the verification will succeed (if such a configuration exists).

IN CONTEXT: THIS FY2019–20 PROJECT

- builds on prior DoD line-funded research and sponsored work on timing verification of undocumented multicore, verifying distributed adaptive real-time systems, high-confidence cyber-physical systems, and real-time scheduling for multicore architectures
- aligns with the CMU SEI technical objective to bring capabilities through software that make new missions possible or improve the likelihood of success of existing ones

Rapid Certifiable Trust

Fielding new technologies is essential to **preserve defense superiority.** However, this is only possible if these technologies can be **validated for safety**.

Challenges for Validation

- Increasingly complex systems
- Changing behavior at runtime (e.g., machine learning)
- Interactions with physical world (e.g., vehicles)
  - Must have correct value
  - Occur at right time (i.e., before crash)

Methods

Formal automatic verification

- **Scalable**
  - Unverified components
  - Monitored and enforced by verified components
  - Verified components protected from unverified components
- **Verified**
  - Physics: verify reaction of physical model (e.g., physical vehicle)
  - Logic: correct value with correct protection
  - Timing: occurs at the right time
- **Protect** verified components

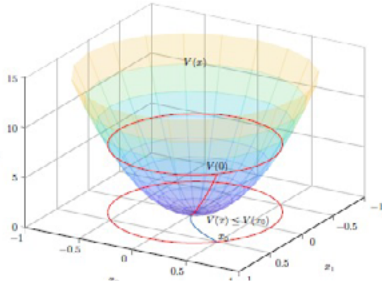
Results

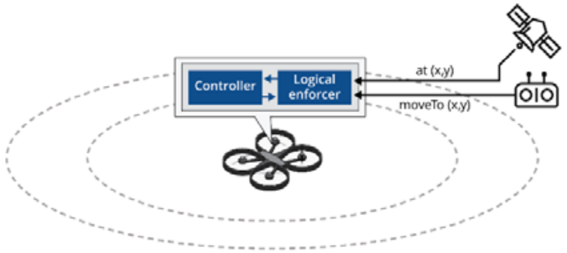
Real-time Mixed-Trust Computation

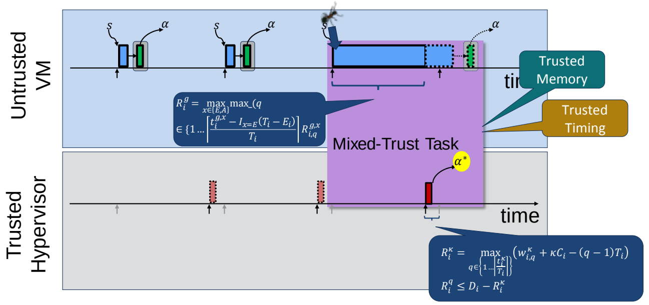
- Verified protection mechanism (micro-hypervisor: uber XMHF)
- Timing verification of combined trusted/untrusted (mixed-trust)
- Physics verification of enforcement

Preserve safety by verifying only a small part of the system. **Assure trust** by protecting the verified part.

Trust = Verified + Protected







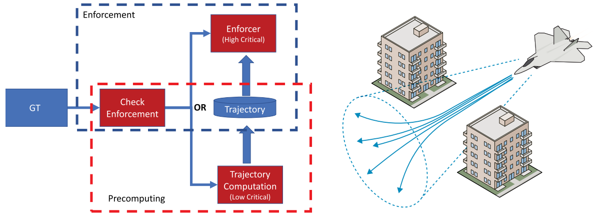
$$R_i^g = \max_{x\in\{E,A\}} \max\_(q \in \{1 \ldots \left\lceil \frac{t_i^{g,x} - I_{x=E}(T_i - E_i)}{T_i} \right\rceil R_{i,q}^{g,x}$$

$$R_i^\kappa = \max_{q\in\left\{1\ldots\left\lceil\frac{t_i^\kappa}{T_i}\right\rceil\right\}} \left(w_{i,q}^\kappa + \kappa C_i - (q-1)T_i\right)$$

$$R_i^q \le D_i - R_i^\kappa$$

NEW RESULTS

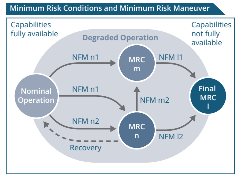
Predictive Mixed-Trust Scheduling

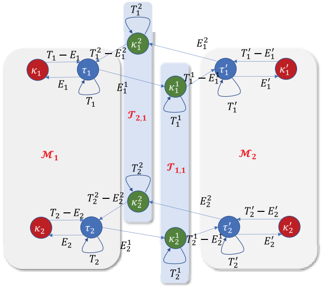


Balance trajectory production/consumption: $G_i^d(I_i - 1) - S_i^e \geq 0$

+ Response: $R_i^p = \kappa C_i^p + \sum \left\lceil \frac{R_i^p}{T_j} \right\rceil \kappa C_j^p - \left\lfloor \frac{R_i^p}{I_j T_j} \right\rfloor \left(\kappa C_j^p - \kappa C_j^e\right)$

Resilient Mixed-Trust Autonomy Scheduling


Minimum Risk Conditions and Minimum Risk Maneuver
Capabilities fully available
Capabilities not fully available
Degraded Operation
Nominal Operation
NFM n1
NFM n1
NFM n2
MRC m
MRC n
NFM m2
NFM l1
NFM l2
Final MRC l
Recovery



$$J(g_j) = \max_{v_{j,q}\in V_j} (D_{j,q} - C_{j,q})$$

$$rf_{\pi_j}^{v_{i,k}}(t) \coloneqq \max\{e(\pi_j') | \pi_j' \; is \; prefix \; of \; \pi_j \; and \; end(\pi_j', v_{i,k})\}$$

$$end(\pi, v_{i,k}) = \begin{cases} p(\pi) \le t & if \; v_{i,k} \; is \; non - preemptive \\ p(\pi) < t & otherwise \end{cases}$$

$$MI(v_{i,k}) = P(v_{i,k}) + \sum_{g_j \in hp(i)} rf_{\pi^{g_j}}^{v_{i,k}} (MI(v_{i,k}) + J(g_i))$$

Principal Investigator

DR. DIONISIO DE NIZ
Technical Director, Assuring
Cyber-Physical Systems

Carnegie Mellon University
Software Engineering Institute

RAPID CERTIFIABLE TRUST

The DoD recognizes the need to field new cyber-physical systems (CPS) capabilities at an increasingly rapid pace, which is why it maintains a number of initiatives on rapid deployment. The demand for more rapid deployment, however, creates a need for verification techniques that can adapt to a faster deployment cadence, especially for CPS that are too big for traditional verification techniques and/or involve unpredictable aspects, such as machine learning.

The goal of Rapid Certifiable Trust is to reduce the deployment time of CPS by reducing the overall development and assurance times. We will do this by enabling the use of unverified commodity software components (e.g., open source drone piloting software) guarded by verified enforcers that guarantee the containment of unsafe component behavior. We are developing compositional verification techniques to allow us to use multiple enforced components minimizing and automatically removing conflicting enforcer assumptions (e.g., reducing a plane's airspeed to avoid crash while increasing airspeed to prevent stalling). These techniques will allow us to assure full-scale systems, even if most of their functionality is implemented by unverified components. Our objective is to develop enforcement verification techniques that scale to at least 10 enforced controllers.

IN CONTEXT: THIS FY19–21 PROJECT

- builds on line-funded work on Certifiable Distributed Runtime Assurance, the goal of which was to facilitate confident and rapid deployment of autonomous distributed real-time systems (DRTS) operating in uncertain and contested environments
- seeks to verify software-reliant systems that interact with physical processes (e.g., aircraft) to which existing verification technology does not scale
- will develop enforcing algorithms to identify unsafe control actions and replace them with safe actions
- drones are used to validate our approach in the SEI's drone lab
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties
- also aligns with the CMU SEI technical objective to make software delivery timely so that the cadence of acquisition, delivery, and fielding is responsive to and anticipatory of the operational tempo of DoD warfighters

Rapid Adjudication of Static Analysis Meta-Alerts During Continuous Integration (CI)

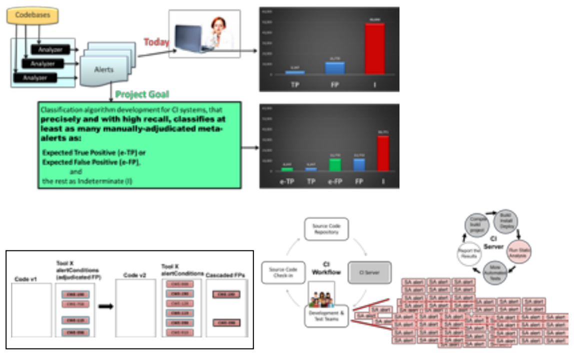
Problem

Manual adjudication of static analysis meta-alerts requires too much effort in short CI build and PR-approval time frames to address many (if any) of them. This problem is technically challenging. Developing a new static analysis to ***precisely*** match flaws in different version of Java or C++ code requires language-specific algorithms, and the matching must be fast to work in a CI/CD system. Also, when cascading is ***imprecise***, mis-labeled data worsens classifier performance, and no effective systems exist that use automated classifiers for multiple static analysis tools in a CI system.

Solution

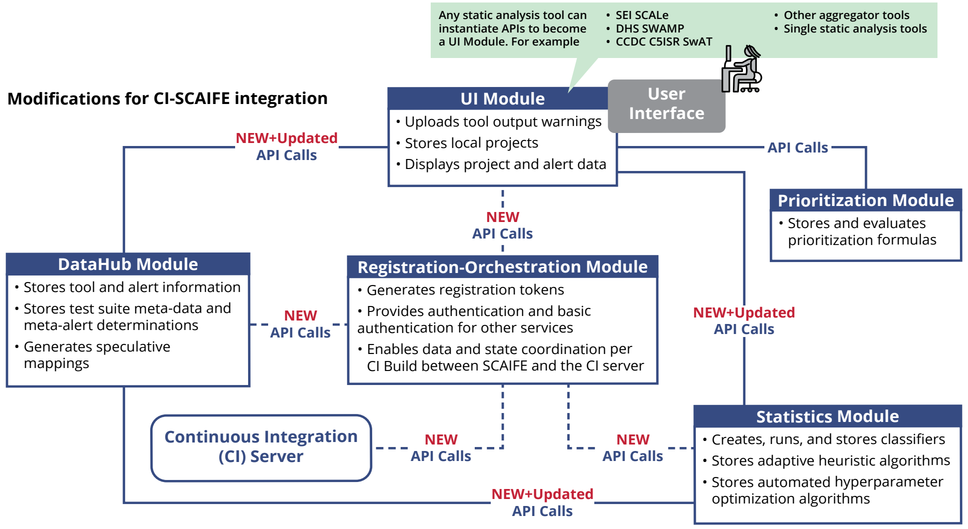
The solution involves (1) a system that supports classification integrated with CI, and builds on the SCAIFE API and implementation we developed for an extensible architecture that supports classification, and (2) precise cascading algorithms for C++ code.

We (1) designed a model for integrated SCAIFE-CI systems, including SCAIFE changes, performance measures, and new classifier features; (2) implemented parts of the design (collaborators tested and reviewed subsequent versions); (3) performed an experiment using diff-based (imprecise) cascading and generated data for comparison to precise cascading. Future plans are to develop a precise cascading algorithm, improve classifiers, and fully integrate them.


Codebases
Analyzer
Analyzer
Analyzer
Alerts
Today
TP
FP
I
Project Goal
Classification algorithm development for CI systems, that
precisely and with high recall, classifies at least as many manually-adjudicated meta-alerts as:
Expected True Positive (e-TP) or
Expected False Positive (e-FP),
and
the rest as Indeterminate (I)
e-TP
TP
e-FP
FP
I
Code v1
Tool X alertConditions (adjudicated FP)
Code v2
Tool X alertConditions
Cascaded FPs
Source Code Repository
Source Code Check-in
CI Workflow
CI Server
Development & Test Teams

To **overcome barriers to using automated classifiers during CI**, we designed a system **that enables classification to be used in CI builds, including cascading adjudications.**

Modifications for CI-SCAIFE integration



Any static analysis tool can instantiate APIs to become a UI Module. For example

- SEI SCALe
- DHS SWAMP
- CCDC C5ISR SwAT
- Other aggregator tools
- Single static analysis tools

UI Module
- Uploads tool output warnings
- Stores local projects
- Displays project and alert data

Prioritization Module
- Stores and evaluates prioritization formulas

DataHub Module
- Stores tool and alert information
- Stores test suite meta-data and meta-alert determinations
- Generates speculative mappings

Registration-Orchestration Module
- Generates registration tokens
- Provides authentication and basic authentication for other services
- Enables data and state coordination per CI Build between SCAIFE and the CI server

Continuous Integration (CI) Server

Statistics Module
- Creates, runs, and stores classifiers
- Stores adaptive heuristic algorithms
- Stores automated hyperparameter optimization algorithms

FY20 Code and API Artifacts

- (Sep 2020) SCAIFE System v 1.2.2 is released with significant CI-SCAIFE integration progress; it includes five APIs, an HTML manual, SCALe, and the rest of the software system. (collaborators)
- (Sep 2020) SCALe is released for SCALe v. r.6.2.2.2.A. (public)
- (Sep 2020) Five SCAIFE APIs are released. (collaborators, public)
- (Jul 2020) SCAIFE System v 1.1.1 is released with API modules and SCALe automation for CI-SCAIFE integration; the system includes separable SCALe v. r.6.1.1.1.A, five APIs, and an HTML manual. (collaborators)
- (Mar 2020) SCAIFE System v 1.0.0 is released with containers for CI-SCAIFE integration; the system includes a SCALe separable module, new APIs, and an HTML manual. (collaborators)
- (Feb 2020) SCAIFE API v 0.0.9-beta is published. (collaborators, GitHub)
- (Oct 2019) SCAIFE System Beta VM v 2.1 is released with a bill of materials. (collaborators)

FY20 Additional Artifacts

- (Sep 2020) Diff-based cascading experiment artifacts are produced.
- (Sep 2020) A SCAIFE/SCALe HTML manual is released for SCALe v r.7.0.0.0.A. (public, collaborators)
- (Jul 2020) “How to Instantiate SCAIFE API Calls” manual is released. (public)
- (Apr 2020) “Open Dataset RC_Data for Classifier Research” is published. (public)
- (Mar 2020) “How to Test and Review the SCAIFE System v 1.0.0 Release” manual is published. (collaborators)
- (Feb 2020) “SCAIFE API Version 0.0.9-Beta: Reviewer Roadmap” manual is published. (collaborators)

The team developed progressive versions of (1) a design for CI-classifier (CI-SCAIFE) integration and (2) an API definition. The team also implemented a system for modular classification with features to enable CI-integration and to measure performance.

Principal Investigator

DR. LORI FLYNN
Senior Software Security Engineer

Carnegie Mellon University Software Engineering Institute

RAPID ADJUDICATION OF STATIC ANALYSIS ALERTS DURING CONTINUOUS INTEGRATION

The DoD has directed a shift toward continuous integration/continuous deployment (CI/CD) to maintain a competitive edge. [McMurry 2018] It is currently standard to run automated unit, integration, and stress tests during CI builds, but static analysis (SA) tools are not always part of builds because CI time frames are too short. However, SA tools could detect code flaws that are cheaper to fix earlier in the development process during CI builds.

It is increasingly common to use multiple SA tools and combine their alerts to maximize the identification of potential security flaws. [Delaitre et al. 2018] However, current SA tools produce some false positive (FP) alerts that require humans to inspect the code and manually adjudicate true alerts vs. false. [Heckman 2011] We use the term alertCondition to designate an alert from a tool mapped to a member of an external taxonomy of conditions (code flaws); for instance, CWE-190 from the CWE taxonomy. If SA is used within CI, alertConditions could stop a build and force human adjudication of true positive (TP) vs. FP, which slows development but might net an acceptable tradeoff if the slowdown is limited and/or occasional. Furthermore, many previously adjudicated FP alerts reappear each time an SA tool is run on a subsequent code version.

To maintain development velocity, DoD organizations with a continuous authority to operate (ATO) process have been forced to make tradeoffs in their security development testing and evaluation processes. For example, one organization removed SA tools from the CI/CD process, substituting a more expensive, less agile, and later manual review. Another kept SA tools, but reduced their sensitivity and analyzed only a small subset of the alerts, which introduced false negatives. We take the latter approach as a starting point, our goal being to increase efficiency by automating this process.

This research project will use machine learning and semantic analysis of data generated during CI/CD to reduce the number of alerts requiring human adjudication by 50% in multiple SA tool deployments without slowing the development process. More specifically, this project will

- improve the state of the art in reducing false positives and integrating SA tools into CI/CD processes
- improve the state of the practice by delivering and validating a prototype system that implements the new algorithms and measures the effectiveness of the techniques

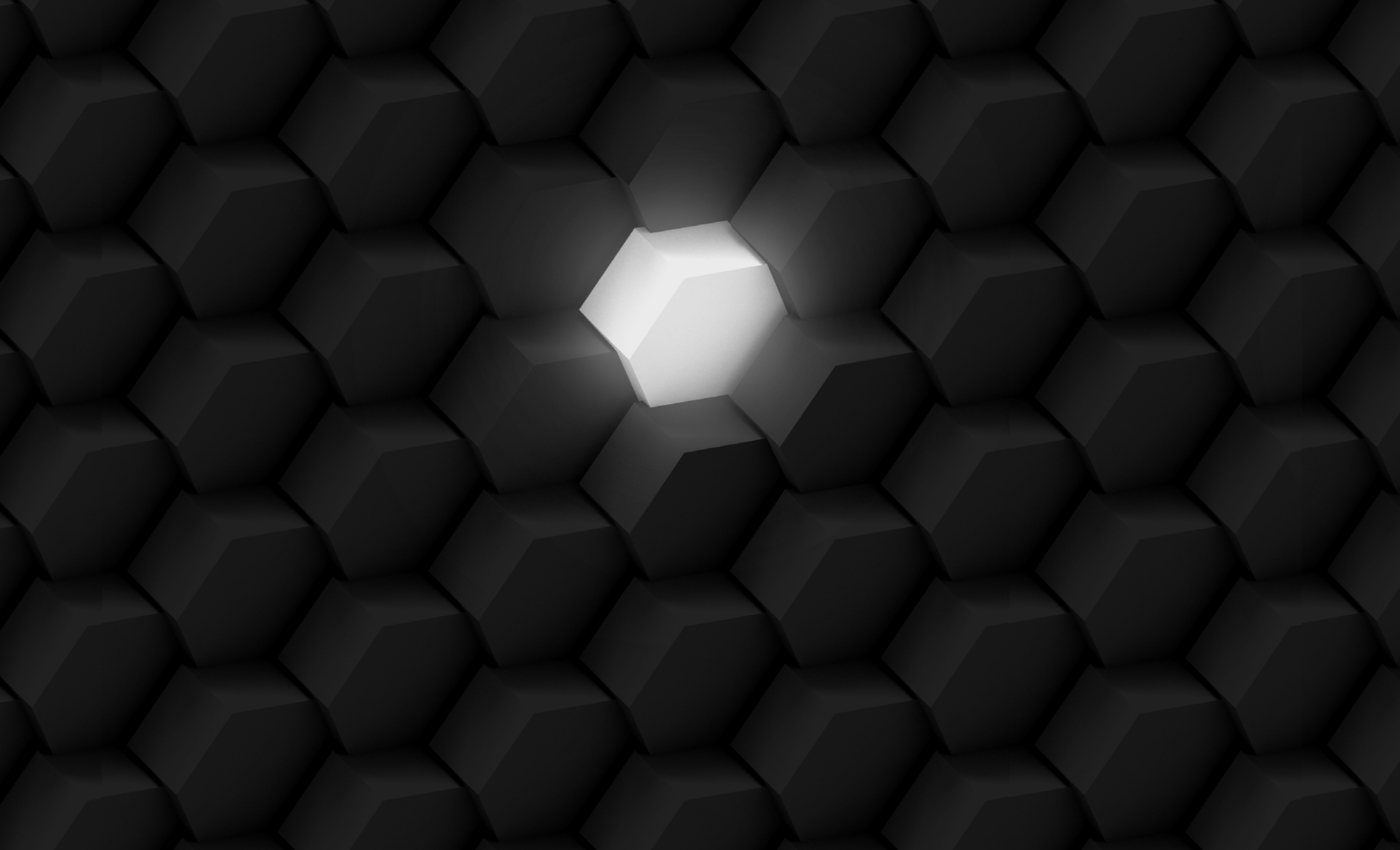
IN CONTEXT: THIS FY20–21 PROJECT

- builds on a number of previous projects, including “Rapid Construction of Accurate Automatic Alert Handling System: Model & Prototype” and “Running in the Cloud Without Breaking the Bank”
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties, including known and yet-unseen adversary capabilities

SECTION 5

Equip the Cyber and Information Operators with Dominant Tradecraft

CMU SEI seeks to move the human operator “out of the loop” by developing automation and autonomy in key cyber tradecraft areas needed for agile, risk-informed cyber response actions (for instance, malware analysis, forensics, situational awareness, adversary assessment and incident management).



Human Decision Making with AI Support

The Problem:
Time and again we've seen humans making poor choices while relying on (or ignoring) existing AI decision support systems. These failures have led several systems to be abandoned. Preliminary research indicates that a failure to communicate model output understandably may contribute to this problem, but it is currently unknown what the best practices in AI system design are that would alleviate it.

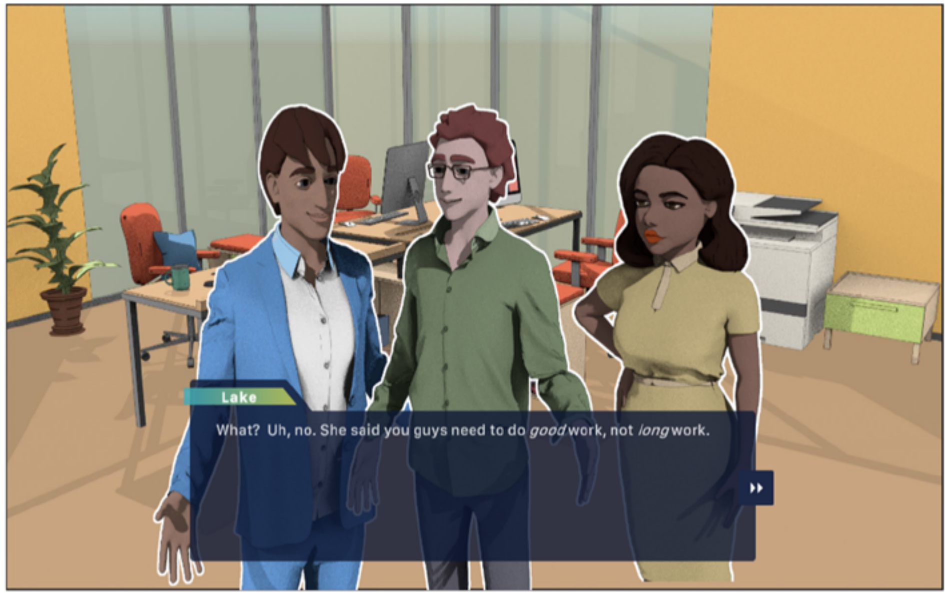
The Solution:
If you want to know what humans will do, you usually need to check what a human will do. Our goal is to collect data on real human decision making and use that data to determine appropriate best practices for AI system interface design within a chosen domain.

The Approach:
We created the Human-AI Decision Evaluation System (HADES). This test harness allows the collection of human decision making data on an arbitrarily large set of possible AI interfaces.

The optimal setting for collecting this data requires a human to repeatedly make the same type of decision over and over again, each time with slightly different information available. Such a task presented directly can quickly induce fatigue and disinterest in a subject. However, this repeated decision making is a common characteristic of games. The specific information available to a player may be modified from turn to turn, but the core game mechanics rarely change.

The Innovation:
Integrate HADES test harness into game environments to observe the effect of AI decision support systems on gameplay outcomes.

To test **human decision making**, you need to test **humans making decisions**.

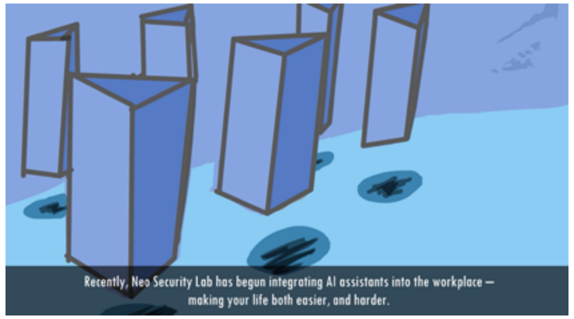


Lake

What? Uh, no. She said you guys need to do *good* work, not *long* work.

Special thanks to our collaborators, Dr. Jessica Hammer, Erik Harpstead, and the students of the CMU Entertainment Technology Center and CMU's OH!Lab, without which the testing of the HADES test harness would have been impossible.

Neo Security Lab: Student-developed game leveraging the HADES test harness



Recently, Neo Security Lab has begun integrating AI assistants into the workplace — making your life both easier, and harder.

Interface Features Tested

Explainability Variables	Contextual Variables
Input Visibility	Underlying Model Accuracy
Selected Features Visibility	Risk / Stakes of Decision
Threshold Types	Cost of Choices
Threshold Adjustability	Unmodelled Information
Confidence Measure Visibility	

HADES Capabilities

- Ability to simulate not-yet-implemented AI systems
  - Allows for data-driven system requirements development
- Slot-In capability for implemented AI systems
  - Useful for verification and validation (V&V) use case
- Standards-Compliant RESTful interface
- Support for multiple experimental designs

Principal Investigator

MR. ROTEM GUTTMAN
Cybersecurity Researcher

Carnegie Mellon University
Software Engineering Institute

HUMAN DECISION MAKING WITH AI SUPPORT

The field of artificial intelligence (AI) is still in its infancy, and vulnerabilities introduced by human-AI interactions are not well understood. Recent failures of systems using AI underscore this point. To best understand the (mis) use of AI-enabled systems, we must be able to collect data on how these systems will be used under a variety of circumstances. However, the current research literature is insufficient to provide actionable guidance to DoD AI-enabled decision support system designers.

Two distinct bodies of research literature bear on this problem: psychological research on human decision making and how humans interpret and react to uncertainty, and nascent research on explainable AI. The literature on explainable AI, including the majority of DARPA XAI projects, focuses on extracting information from an AI model. However, recent research has indicated that, for many individuals, this information is incomprehensible for the purpose of decision making. Extensive literature from the field of psychology and decision sciences documents how humans process information and make decisions in the face of uncertainty. Yet, to date, there has been little work integrating these disparate research threads.

This project aims to create the Human-AI Decision Evaluation System (HADES). HADES allows the investigation and evaluation of AI-assisted human decision making in a variety of simulation environments by exposing a standards-compliant API interface. This is a necessary first step to closing the gap in the literature.

Our test environment will focus on cybersecurity decision making, a domain of critical interest to the DoD. By aligning the test environment with those used in operational settings, we can ensure operational validity. Our criteria for success is an improvement in the average test subject's decision-making quality by at least 50% from their baseline performance.

IN CONTEXT: THIS FY2020 PROJECT

- allows for the testing of AI systems across the development cycle; critically, HADES allows the testing of proposed systems prior to their development to drive requirement setting, as well as verification and validation activities after system development is complete.
- contributes best practices to reduce risk and increase confidence in AI enabled mission support systems; enables testing of new tactics, techniques, and procedures (TTPs), and operations with AI-enabled mission support systems; enables better training for human-AI teaming; and enables testing of AI products for human-AI teaming at all stages of the software lifecycle
- aligns with the CMU SEI technical objective to reduce risk and increase confidence in cyber-enabled mission elements by defining and documenting best practices that align defense operators to mission metrics and through the invention of innovative training environments that allow mission rehearsal for new tactics, techniques, and procedures (TTPs) and operations

Recovering Meaningful Variable Names in Decompiled Code

Introduction

Conventional wisdom tells us that when a compiler transforms a program from source code to executable, some information is lost and cannot be recovered. For example, variable names are not included in a compiled executable, and have been assumed to be lost. Although state of the art decompilers can recover the presence of variables, they make attempt to recover their original names. Instead, they name the variables v1, v2, and so on. This is unfortunate since several studies have shown that programmers carefully select variable names to make the program easier to understand.

In this project, we showed that the conventional wisdom that variable names cannot be recovered is wrong. Specifically, we showed that variable names can largely be predicted based on the context of code in which they are used and accessed. We trained a neural network to predict variable names on a large corpus of C source code that we collected from GitHub.

Corpus

To generate our corpus, we scraped GitHub for projects written in C. We then automatically built 164,632 binaries from these projects and extracted 1,259,935 functions. For each function, we generated a corpus entry that consisted of the original source code with placeholder variables, as shown in the code figure to the right. Each corpus entry also included a mapping from placeholder variable to the original identifier in the source code and the decompiler’s identifier.

We can exactly predict **74.3%** of variable names in decompiled executable code by training a neural network on a large corpus of C source code from GitHub.

```
void *file_mmap(int v1|fd|fd, int v2|size|size)
{
void *ptr|ret|buf;
ptr|ret|buf = mmap (0, v2|size|size, 1, 2, v1|fd|fd, 0);
if (ptr|ret|buf == (void *) -1)
{ perror ("mmap"); exit(1); }

return ptr|ret|buf;
}
```

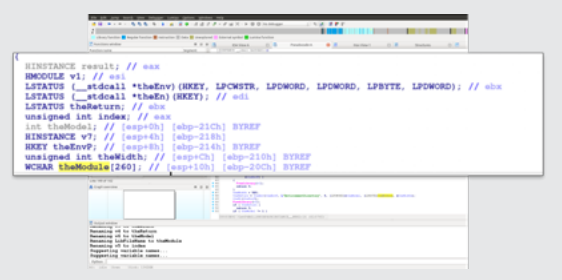
Key

🟥 Decompiled 🟩 Original 🟦 Recovered

Results

Experiment	Accuracy
Overall	74.3
Function in Training	85.5
Function not in Training	35.3

An important consideration when evaluating a solution based on machine learning such as ours is the construction of the training and testing sets. Each binary was randomly assigned to either the training or testing set. As in real reverse-engineering scenarios, library functions may be present in multiple binaries, and thus may be present in both the training and testing sets. To better understand the effect of this on our system, we partitioned our testing set into the set of functions that were also in the training set, and those that were not in the training set. As demonstrated in the table below, DIRE achieves 85.5% accuracy on functions it has been trained on, compared to 74.3% overall. On functions that it has not seen in training, it yields 35.3% accuracy.



```
{
  HINSTANCE result; // eax
  HMODULE v1; // esi
  LSTATUS (__stdcall *theEnv)(HKEY, LPCWSTR, LPDWORD, LPDWORD, LPBYTE, LPDWORD); // ebx
  LSTATUS (__stdcall *theEn)(HKEY); // edi
  LSTATUS theReturn; // ebx
  unsigned int index; // eax
  int theModel; // [esp+0h] [ebp-21Ch] BYREF
  HINSTANCE v7; // [esp+4h] [ebp-218h]
  HKEY theEnvP; // [esp+8h] [ebp-214h] BYREF
  unsigned int theWidth; // [esp+Ch] [ebp-210h] BYREF
  WCHAR theModule[260]; // [esp+10h] [ebp-20Ch] BYREF
```

```
Renaming v4 to theReturn
Renaming v6 to theModel
Renaming lpFileName to theModule
Renaming v5 to index
Suggesting variable names...
Suggesting variable names...
```

Plug-in for Hex-Rays decompiler showing recovered names.

Principal Investigator

MR. CORY COHEN
Member of the Technical Staff/Principal Researcher

Carnegie Mellon University Software Engineering Institute

DR. EDWARD SCHWARTZ
Member of the Technical Staff/Senior Researcher

Carnegie Mellon University Software Engineering Institute

ADVANCING CYBER OPERATOR TRADECRAFT THROUGH AUTOMATED STATIC BINARY ANALYSIS

Highly skilled Department of Defense (DoD) malware and vulnerability analysts must devote significant time to manual tasks. For several years, the SEI has been working on automated solutions to free up limited analyst resources for more meaningful work. Three SEI research threads highlight these efforts:

- recovering meaningful variable names in decompiled code
- program reachability for vulnerability and malware analysis
- improvements to object-oriented construct recovery using OOAnalyzer

Recovering Meaningful Variable Names in Decompiled Code

Understanding executable code is a challenge because the compilation process removes much of the source code information. Decompilers have been widely believed to be unable to recover meaningful variable names, which improve code understandability. To meet this challenge, we developed the Decompiled Identifier Renaming Engine (DIRE), a novel probabilistic technique for variable name recovery that uses lexical and structural information. We also developed a technique for generating corpora for training and evaluating models of decompiled code renaming, which we used to create a corpus of 164,632 unique x86-64 binaries generated from C projects mined from Github. Surprisingly, our results show that DIRE can predict variable names identical to the names in the original source code up to 74.3% of the time.

Program Reachability for Vulnerability and Malware Analysis

Manually coercing specific portions of executable code to run presents a number of challenges, such as determining the unknown input conditions required to trigger the desired behavior, eliminating non-determinism, and coping with missing dependencies complicate this effort. We developed capabilities within the CMU SEI's Pharos binary code analysis framework to address these challenges by identifying the specific program inputs and environments needed to reach an execution of interest to an analyst, which we call path finding. Finding paths in an executable can be especially useful for bypassing runtime anti-analysis checks in the code.

Improvements to Object-Oriented Construct Recovery Using OOAnalyzer

Object-oriented programs pose many challenges for reverse engineers and malware analysts. C++ classes are complex and hard to analyze at the machine code level. We've long sought to simplify the process of reverse engineering object-oriented code by creating tools such as OOAnalyzer, which automatically recovers C++-style classes from executables. OOAnalyzer can export its results to other reverse engineering frameworks, and we've enhanced our Pharos Binary Analysis Framework to import OOAnalyzer analysis into the recently released Ghidra reverse engineering (SRE) tool suite. Ghidra provides the analyst many useful reverse engineering services, including disassembly, function partitioning, decompilation, and various other types of program analyses.

IN CONTEXT: THIS FY2018–20 PROJECT

- extends DoD line-funded research and tool development for vulnerability and binary code analysis
- contributes to development and transition of Pharos binary code analysis framework
- aligns with the CMU SEI technical objective to make software trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties including known and yet-unseen adversary capabilities

Note: The illustrations on the following two pages describe additional threads related to this research.

Program Reachability for Vulnerability and Malware Analysis

Problem

Highly skilled Department of Defense (DoD) malware and vulnerability analysts currently spend significant amounts of time manually coercing specific portions of executable code to run.

Solution

Automate the analysis of binary code, choosing program inputs that will trigger specific behavior to reduce the time that DoD cyber personnel spend performing complex software analysis.

Approach

Use model checking techniques to identify these inputs and generate a simplified executable free of complex and convoluted dependencies that can be analyzed by existing code analysis tools.

Intended Impact (FY18–20)

Improve the DoD's ability to measure and monitor the advancement of path-reachability research, especially as Ghidra decompilation quality improve improves.

Testing Method

A total of 91 test programs were compiled for three optimization levels and two architectures. Each test attempted to find a path from a starting location to a reachable goal and an unreachable goal. If both answers were correct, the test passed. The test timeout was 30 minutes.

2,184 test configurations found several successful approaches, but **none** that **consistently outperformed** the others, suggesting a needed hybrid **approach**.

Pharos Function Summaries
SEI's Pharos binary analysis framework computes symbolic function summaries by symbolically executing binary code. This technique converts these function summaries into light-weight constraints. The conversion uses a simple model of memory that is very efficient, but is known to be incorrect in the presence of interprocedural reasoning.

Weakest Precondition
The weakest precondition algorithm finds the weakest constraints on the program input that are required for the program to terminate successfully. We force execution to the desired program locations by adding assertions. This technique uses an array encoding of memory, which is precise but expensive to reason about. It also cannot reason generally about loops.

Property Directed Reachability
Property Directed Reachability (PDR) is a technique used in source code software model checking. It iteratively generates an inductive invariant to prove that the target code is unreachable, and it uses counter-examples to refine the invariant, so it can prove targets are unreachable even when there are loops. It uses the same array encoding of memory as the previous technique.

Ghidra + Seahorn
This technique uses the NSA's Ghidra decompiler to raise the executable code to a C-like language rather than trying to express the binary semantics directly. The Seahorn software model checker is then used to check reachability using PDR. Because it operates on a source code representation, the encoding is very different than the other PDR approach.

Test Case Configuration		Pharos Function Summaries			Weakest Precondition			Property Directed Reachability			Ghidra + Seahorn		
Optimized	Arch	Fail	Timeout	Pass	Fail	Timeout	Pass	Fail	Timeout	Pass	Fail	Timeout	Pass
None	32-bit	🟥 55	🟩 2	🟥 34	🟨 16	🟩 2	🟩 73	🟩 3	🟥 29	🟧 59	🟧 21	🟧 7	🟨 63
None	64-bit	🟥 47	🟩 0	🟥 44	🟨 15	🟨 3	🟩 73	🟩 2	🟥 36	🟧 53	🟧 28	🟧 2	🟨 61
Medium	32-bit	🟥 40	🟩 0	🟥 51	🟨 9	🟨 3	🟩 79	🟩 1	🟥 13	🟨 77	🟧 12	🟧 7	🟧 72
Medium	64-bit	🟥 53	🟩 0	🟥 38	🟨 9	🟨 4	🟩 78	🟩 1	🟥 17	🟨 73	🟧 21	🟧 6	🟧 64
High	32-bit	🟥 50	🟩 0	🟥 41	🟨 6	🟨 2	🟩 83	🟩 1	🟥 12	🟨 78	🟧 18	🟧 7	🟧 66
High	64-bit	🟥 32	🟩 1	🟥 58	🟨 28	🟨 3	🟩 60	🟩 2	🟥 16	🟨 73	🟧 32	🟧 5	🟧 54
Total		🟥 **257**	🟩 **3**	🟥 **266**	🟨 **83**	🟨 **17**	🟩 **446**	🟩 **10**	🟥 **123**	🟨 **413**	🟧 **132**	🟧 **34**	🟧 **380**

Key

- 🟩 Best result
- 🟨 Second best result
- 🟧 Third best result
- 🟥 Worst result

Pharos Function Summaries: This approach is very fast, but its imprecision results in a large number of failures and a small number of passing tests.

Weakest Precondition: This well-known approach is still the benchmark to beat. It performs well but has significant deficiencies when analyzing code with loops.

Property Directed Reachability: This approach is very accurate but has severe performance problems in the binary domain due the array memory model, which is not necessary at the source code level.

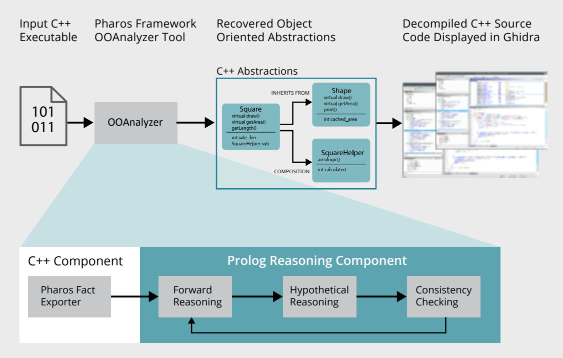
Ghidra + Seahorn: PDR can be fast when using a source code representation. Unfortunately, decompilation can fail in myriad ways, and this accounts for the majority of failures for this approach.

Improvements to Object-Oriented Construct Recovery Using OOAnalyzer

Problem

OOAnalyzer is the state of the art in automatically recovering object-oriented abstractions to assist reverse engineers in malware analysis, vulnerability analysis, and software assurance. First published at the ACM Conference on Computer and Communications Security, OOAnalyzer uses novel techniques to reason in the presence of uncertainty, which is unavoidable in this type of analysis. This feature is heavily dependent on OOAnalyzer's Prolog-based implementation. Unfortunately, early versions of OOAnalyzer were too slow to scale to the large and complex programs used in the DoD.

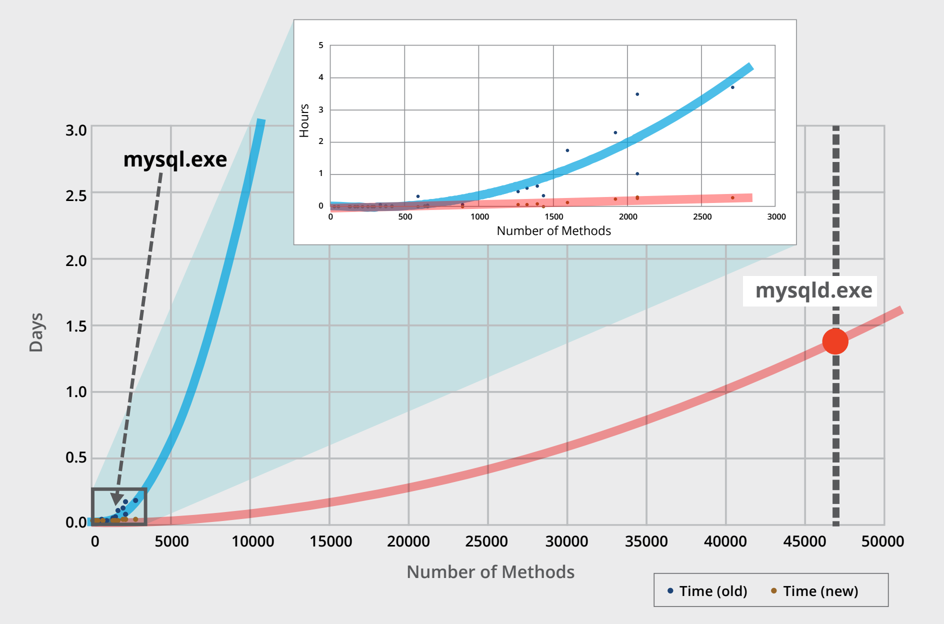
OOAnalyzer Design Overview



Solution

- We worked with the developer of SWI Prolog to create novel profiling and debugging tools for Prolog.
- Many problems were simple to fix once the problem was identified using new tools.
- Unfortunately, we identified systemic issues related to the Prolog tabling optimization.
- We avoided these issues with a new technique and are working with the SWI developers on a general solution.

OOAnalyzer was too slow to be used on the programs that the DoD needs it for the most. It is now **50x** faster and can analyze large programs.



Before and After Data

Program	# Class	# Method	Time (Old)	Time (New)	Improvement
x3c	6	28	0:00:01	0:00:01	0.6x
Malware d597bee8	19	133	0:00:04	0:00:04	0.0x
Malware 0faaa3d3	21	135	0:00:05	0:00:07	-0.3x
optionparser	11	56	0:00:05	0:00:01	3.8x
MySQL connection.dll	43	166	0:00:07	0:00:04	0.7x
Malware cfa69fff	39	182	0:00:08	0:00:09	-0.1x
light-pop3-smtp	44	290	0:00:21	0:00:14	0.5x
Malware 29be5a33	19	130	0:00:24	0:00:05	3.7x
CImg	29	220	0:00:52	0:00:11	3.6x
MySQL ha_example.dll	21	256	0:01:04	0:00:16	3.1x
Firefox	141	638	0:01:47	0:01:30	0.2x
PicoHttpD	95	656	0:03:38	0:00:37	4.9x
Malware 6098cb7c	55	339	0:03:54	0:00:15	**14.5x**
Malware 67b9be3c	400	2072	2:42:19	0:17:31	**8.3x**
MySQL cfg_editor.exe	190	1270	3:27:50	0:03:53	**52.6x**
MySQL libmysql.dll	200	1327	4:22:55	0:04:04	**63.7x**
Malware f101c05e	169	1601	4:25:34	0:07:17	**35.5x**
MySQL mysql.exe	202	1395	4:34:49	0:04:37	**58.5x**
MySQL upgrade.exe	333	2069	11:34:56	0:15:30	**43.8x**
Malware 628053dc	207	1920	11:46:38	0:14:16	**48.5x**
Malware deb6a7a1	283	2712	17:33:52	0:17:15	**60.1x**

REFERENCES

[Amershi et al. 2019] Amershi, S.; Begel, A.; Bird, C.; Deline, R.; Gall, H.; Kamar, E.; Nagappan, N.; Nushi, B.; and Zimmermann, T. Software Engineering for Machine Learning: A Case Study. Pages 291-300. In Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '19). Montreal, Canada. May 2019. URL: **https://www.microsoft.com/en-us/research/uploads/prod/2019/03/amershi-icse-2019_Software_Engineering_for_Machine_Learning.pdf.**

[Delaitre et al. 2018] Delaitre, Aurelien M.; Stivalet, Bertrand C.; Black, Paul E.; Okun, Vadim; Cohen, Terry S.; and Ribeiro, Athos. SATE V Report: Ten Years of Static Analysis Tool Expositions. No. Special Publication (NIST SP)-500-326. 2018.

[Ghelani 2019] Ghelani, S. ML Models—Prototype to Production. Towards Data Science. October 20, 2019 [accessed]. **https://towardsdatascience.com/ml-models-prototype-to-production-6bfe47973123.**

[Glymour et al. 2019] Glymour, Clark; Zhang, Kun; and Spirtes, Peter. Review of causal discovery methods based on graphical models. Frontiers in Genetics. Volume 10. June 4, 2019. Page 524. **https://www.frontiersin.org/articles/10.3389/fgene.2019.00524/full.**

[Heckman 2011] Heckman, Sarah and Williams, Laurie. A systematic literature review of actionable alert identification techniques for automated static code analysis. *Information and Software Technology.* Volume 53. Number 4. Pages 363–387. April 2011.

[McMurray et al. 2018] McMurry, Robert D. and Roper, William B. Establishment of Air Force Program Executive Officer (PEO) Digital. Memorandum for all AFPEOs. Washington, D.C., Department of the Air Force. August 29, 2018. **https://www.hanscomreps.org/wp-content/uploads/2018/09/20180829-PEO-Digital-Establishment-Memo-Signed.pdf.**

[Ransbotham et al. 2017] Ransbotham, S.; Kiron, D.; Gerbert, P.; and Reeves, M. Reshaping business with artificial intelligence: Closing the gap between ambition and action. MIT Sloan Management Review. Volume 59. Number 1. September 6, 2017.

[Sculley et al. 2015] Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; and Dennison, D. Hidden Technical Debt in Machine Learning Systems. Pages 2503-2511. In Advances in Neural Information Processing Systems. 2015. **http://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf.**

[Seligman 2016] Seligman, Lara. Interview: Air Force Chief Scientist Dr. Greg Zacharias. Defense News. February 20, 2016. **http://www.defensenews.com/story/defense/policy-budget/leaders/interviews/2016/02/20/interview-air-force-chief-scientist-dr-greg-zacharias/80424570/.**

[Vasudevan 2016] Vasudevan, Amit; Chaki, Sagar; Maniatis, Petros; Jia, Limin; & Datta, Anupam. überspark: Enforcing verifiable object abstractions for automated compositional security analysis of a hypervisor. 87-104. *Proc. of the 25th USENIX Security Symposium.* August 2016.

WE CAN HELP YOU MAKE A DIFFERENCE

The SEI works with the DoD, government, industry, and academia to help organizations in all these sectors make a positive difference in a rapidly evolving world. How can we help you make a difference? We encourage you to contact us at **info@sei.cmu.edu**. To learn more about how we work with organizations in your sector, visit **sei.cmu.edu/about/work-with-us/index.cfm**.

COPYRIGHT

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

DM20-1015

About Us

The Software Engineering Institute (SEI) at Carnegie Mellon University is a Federally Funded Research and Development Center (FFRDC)—a nonprofit, public–private partnership that conducts research for the United States government. One of only 10 FFRDCs sponsored by the U.S. Department of Defense (DoD), the SEI conducts R&D in software engineering, systems engineering, cybersecurity, and many other areas of computing, working to introduce private-sector innovations into government.

As the only FFRDC sponsored by the DoD that is also authorized to work with organizations outside of the DoD, the SEI is unique. We work with partners throughout the U.S. government, the private sector, and academia. These partnerships enable us to take innovations from concept to practice, closing the gap between research and use.

Contact Us

Carnegie Mellon University
Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

412.268.5800 | 888.201.4479
sei.cmu.edu | info@sei.cmu.edu