Sandia National Laboratories

Exceptional service in the national interest

ASC DEVOPS RESEARCH

FY25 Efforts and Trilinos

Roscoe A. Bartlett, Anderson Chauphan,
Reed M. Milewicz, Jim M. Willenbring

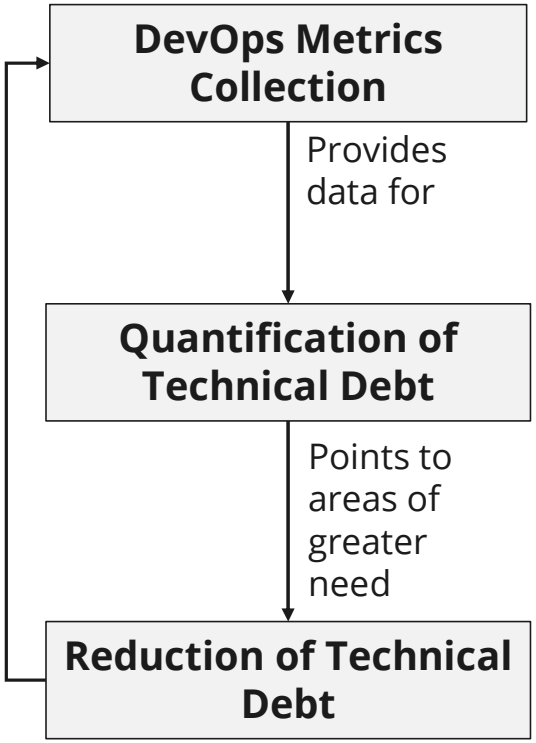
Software Engineering & Research
Department 1424

Trilinos Users Group Meeting 2024, Developers Day

October 24, 2025

SAND2024-14392C

ASC DEVOPS RESEARCH (ASCDOR): BOTTOM LINE UP FRONT



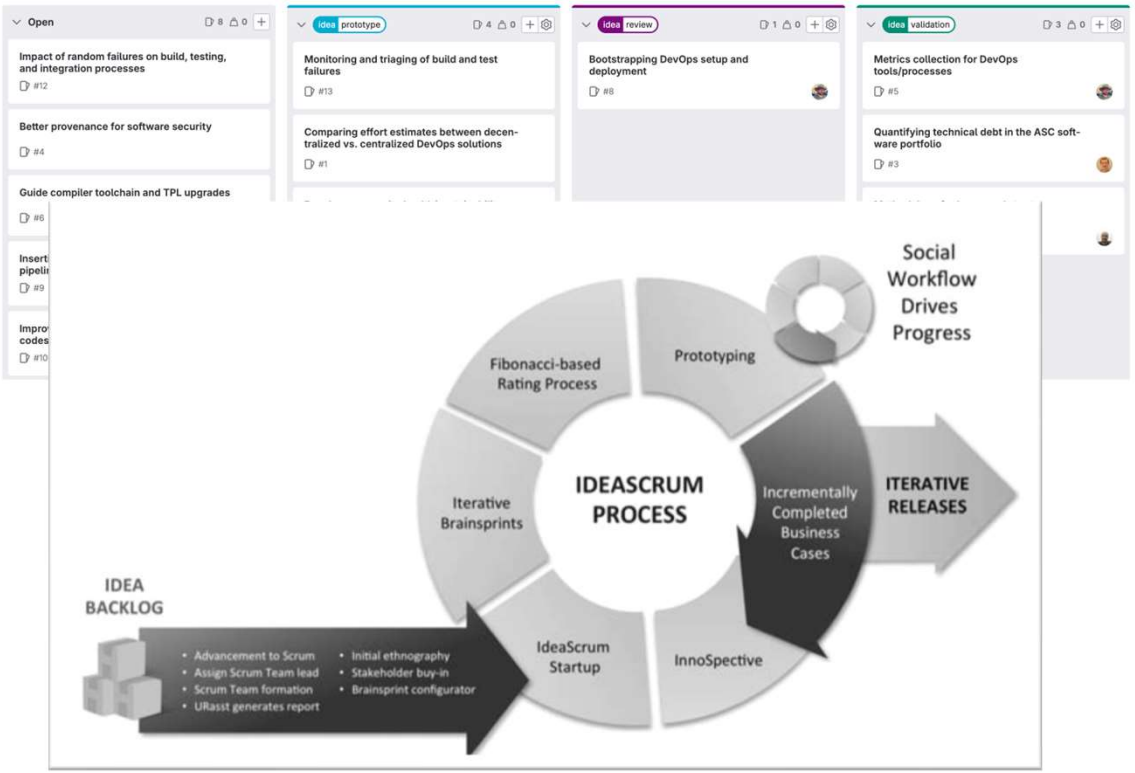
R&D strategies to reduce technical debt based on metrics and analysis

- **What We Want To Do**: Develop novel data-driven, AI-assisted approaches to reduce existing and future ASC software technical debt so that we can improve speed, productivity, agility, and quality in ASC software development.
- **Why We Need To Do It**: Technical debt leads to high operational costs, unreliability, and a lack of agility. To move faster as a lab, we must improve how we develop and manage our software.
- **How We Will Do It**: We propose a three-pronged effort between late FY24 – FY27 focused on (1) metrics collection for DevOps tools and processes, (2) approaches for identifying and quantifying technical debt, and (3) research strategies to reduce identified technical debt.

OUR METHODOLOGY

We iteratively developed a wide variety of proposal topics which could be of value to ASC DevOps

- Used an Agile Innovation approach (IdeaScrum) to identify, develop, and refine research proposals.
- We conducted interviews key ASC DevOps stakeholders to gather insights and requirements.
- Based on these discussions, we generated a broad range of research ideas. Each idea was then developed into user stories, which were iteratively refined through cycles of prototyping, review, and validation.



PROPOSAL IDEA DEVELOPMENT

- Measuring the Impact of Random Failures
- Better Provenance for Software Security
- Guide Compiler Toolchain and TPL Upgrades
- Inserting Security Tools into DevOps Pipelines
- Metrics Collection for DevOps Tools/Processes
- Simplifying Configuration Management
- Enabling and Improving Continuous Deployment
- Improving Requirements Traceability with LLMs
- Methodology for Large-Scale Test Management

- Complex Test Rejuvenation for Heterogeneous Architectures
- Monitoring and Triage of Build and Test Failures
- Quantifying Technical Debt in the ASC Software Portfolio
- Comparing Decentralized vs. Centralized DevOps Solutions
- Developing a Community Health/Sustainability Dashboard
- Better Developer Support for Package Managers
- Bootstrapping DevOps Setup and Deployment

BRO

IN WORKING WITH ASC DEVOPS STAKEHOLDERS, WE IDENTIFIED THE FOLLOWING NEEDS

A common overarching theme: the pursuit of efficiency, reliability, and continuous improvement through DevOps

Metrics Collection for DevOps Tools and Processes

- Developing a comprehensive suite of targeted metrics to measure the effectiveness of ASC DevOps solutions. We seek to provide actionable insights that can guide decision-making and align ASC practices with industry standards.

Identifying and Quantifying Technical Debt in the ASC Software Portfolio

- Developing approaches to assess technical debt within ASC software projects. We seek to quantify the impact of technical debt on development efficiency and point to areas where larger improvements are possible.

Methodology for Large-Scale Test Management

- Investigating methodologies and tools to reduce the cost of test suites while not sacrificing coverage. We seek enhancing the utility and efficiency of testing practices within ASC, enabling teams to better leverage DevOps automation.

Impact

Enhanced Observability → Informed Decision-Making → Strategic Management of ASC Software Assets

Slide 5

BR0 Is this really true? How is anything we are proposing going to "enable teams to better leverage DevOps automation"?

Bartlett, Roscoe, 2024-04-24T13:17:13.981

WHAT WE MEAN BY TECHNICAL DEBT

Moving at the speed of mission has historically required making compromises between near-term needs and long-term sustainability.

In software-intensive systems, **technical debt** is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability.[Avgeriou]

Cognitive Overload

Computational Overhead

Low intra-module cohesion, high inter-module coupling, long/confusing functions, high complexity, poor design/poor naming, no comments or incorrect comments, other "code smells", etc ....

Build-related: Long build times (e.g. deeply templated C++ code, implicit template instantiation, deep stacks of included header files), Long rebuild times (e.g. most changes need to be made to header files which propagate to all build targets), high RAM usage per target not allowing using all the cores on a node to build

Test run technical debt: No/poor unit or system tests to allow development and testing pieces of code in isolation, long test runtimes, multiple cores or processes per test, high usage of expensive/scarce accelerator resources (e.g. GPUs)

[Avgeriou] Avgeriou, Paris; Kruchten, Philippe; Ozkaya, Ipek; Carolyn, Seaman (2016). "Managing Technical Debt in Software Engineering (Dagstuhl seminar 16162)". Dagstuhl Reports. 6 (4).

WHERE WE ARE TODAY, WHERE WE ARE GOING

Managing Debt and Sustainability Matters for ASC

- "The ASC program lacks an integrated approach for leveraging and strategically managing [environments and common practices… thwarting] co-development, efficiency and **sustainability** objectives." — ASC DevOps TLT Charter[1]
- "A significant focus needs to be on **developing sustainable solutions to lower the recurring debt**." — ASC DevOps Project Charter[2]

How We Address Debt Today

- **Manual work** refactoring code and tests
  - Requires large amounts of valuable developer time, may require specialized domain knowledge, etc.
  - Many existing tools for refactoring aren't available for large-scale C++.

How We Will Break Through

As is done in industry, data collected from DevOps pipelines and continuous practices can reveal **opportunities for improvement[3].**

Emerging AI4SE technologies can **accelerate debt reduction tasks**.

[1] TLT Charter. https://wiki.sandia.gov/display/ADT/TLT+Charter. Accessed 3 June 2024.
[2] ASC DevOps Project DevOps Charter. https://wiki.sandia.gov/display/DevOps/ASC+DevOps+Resources. April 2018.
[3] Lunde, Bjørn Arild and Colomo-Palacios, Ricardo. "Continuous practices and technical debt: a systematic literature review." *2020 20th International Conference on Computational Science and Its Applications (ICCSA)*. IEEE, 2020.

METRICS => QUANTIFY TECH DEBT => REDUCE TECH DEBT
FOCUS ON COMPUTATIONAL OVERHEAD TECH DEBT?

DevOps Metrics Collection

Provides data for

Quantification of Technical Debt

Points to areas of greater need

Reduction of Technical Debt

R&D strategies to reduce technical debt based on metrics and analysis

Research Test Suite Cost Reduction?

- Select reduced test suite while minimizing loss of coverage?
- Partition test suite and run at different intervals?
- Reduce cost of existing system-level tests?
- **Factor out code and add unit tests and move most testing to unit tests?** *
- ???

Research (Re)Build Cost Reduction?

- More implicit template instantiation?
- More information hiding (e.g. pImpl)?
- Usage of C++20 modules?
- **Factor out code to allow most development and (re)builds of unit tests?** *
- ???

Refactored using

Refactored using

Process to achieve balance

Legacy Software Change Algorithm (LSCA)

AI IMPACT ON LEGACY SOFTWARE (CODE) CHANGE ALGORITHM OVERVIEW

Large suite of expensive set of system-level tests

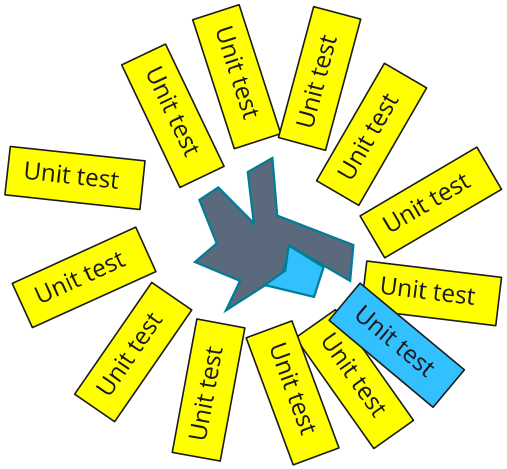
(**NOT shown to scale!**)

Full code base

2. Break dependencies
Minimal refactors to allow extraction of code into unit test harness
- **How to automate refactors**?
  - ⇒ Existing refactoring tools?
  - ⇒ LLVM/Clang AST with **AI**?

1. Analyze code to be extracted
1.a: Select code to be changed/extracted
1.b: Select control points
- ⇒ LLVM/Clang AST with **AI**?

1.c: Select sensing points
- ⇒ LLVM/Clang AST with **AI**?



Factored out code runnable in a unit test harness

3. Cover extracted code with characterization tests
- **How to select/generate test cases?**
  - ⇒ Capture from full test suite?
  - ⇒ **Code analysis & test generation using AI**?

4. Add new code and tests
4.a: Add new code and tests and/or change behavior (e.g. using TDD)
4.b: Refactor updated code to make simple and clean 😊

5. Propagate any changes to public APIs?
- ⇒ LLVM/Clang AST with **AI**?

Hard tasks in Red!

Research Questions in Green!

AI IMPACT ON LEGACY SOFTWARE CHANGE ALGORITHM IMPLEMENTATION: DEEP DIVE EXAMPLES

- Developing characterization tests for undertested regions of code can be automated or assisted by AI:
  - Generate new test case
  - Generate improved test case
  - Generate 85% complete test case

- AI has the potential to bring down the cost of hardening legacy production code, including impacting some of the most challenging steps of the legacy software change algorithm.

3. Cover extracted code with characterization tests
- How to select test cases?
  - ⇒ Capture from full test suite?
  - ⇒ **Code analysis/test generation using AI**?

BRO

Activities are individually impactful, and contribute towards eventual fully automated implementation of the legacy software change algorithm.

2. Break dependencies
Minimal refactors to allow extraction of code into unit test harness
- How to automate refactors?
  - ⇒ Existing refactoring tools?
  - ⇒ LLVM/Clang AST with **AI**?

- Breaking dependencies can be automated or assisted by AI
  - Refactoring and test improvement may evolve into a virtuous cycle
    - Improved testing provides confidence for more aggressive refactoring

Slide 10

BR0 NOTE: These could work together. An initial set of tests could be extracted from the running system-level tests and then a second LLM-based analysis tool could try to suggest new test cases that increase coverage. The former uses data that is used in the existing tests which has a lot of advantages for numerical code.
Bartlett, Roscoe, 2024-05-01T17:59:45.446

WHY US, WHY NOW

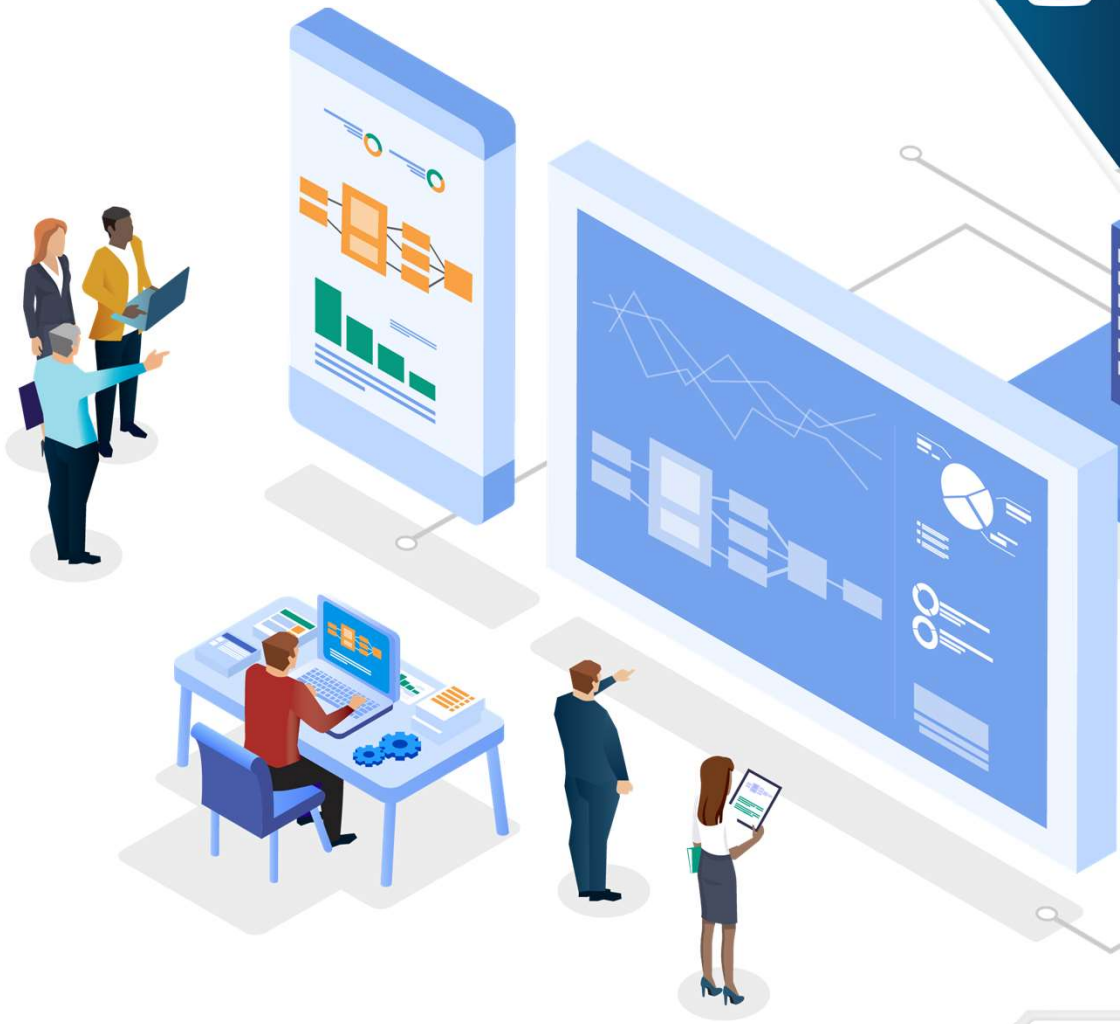
Vision for the Future

"Humans and AI are trustworthy collaborators that rapidly evolve systems based on programmer intent."[Carleton]

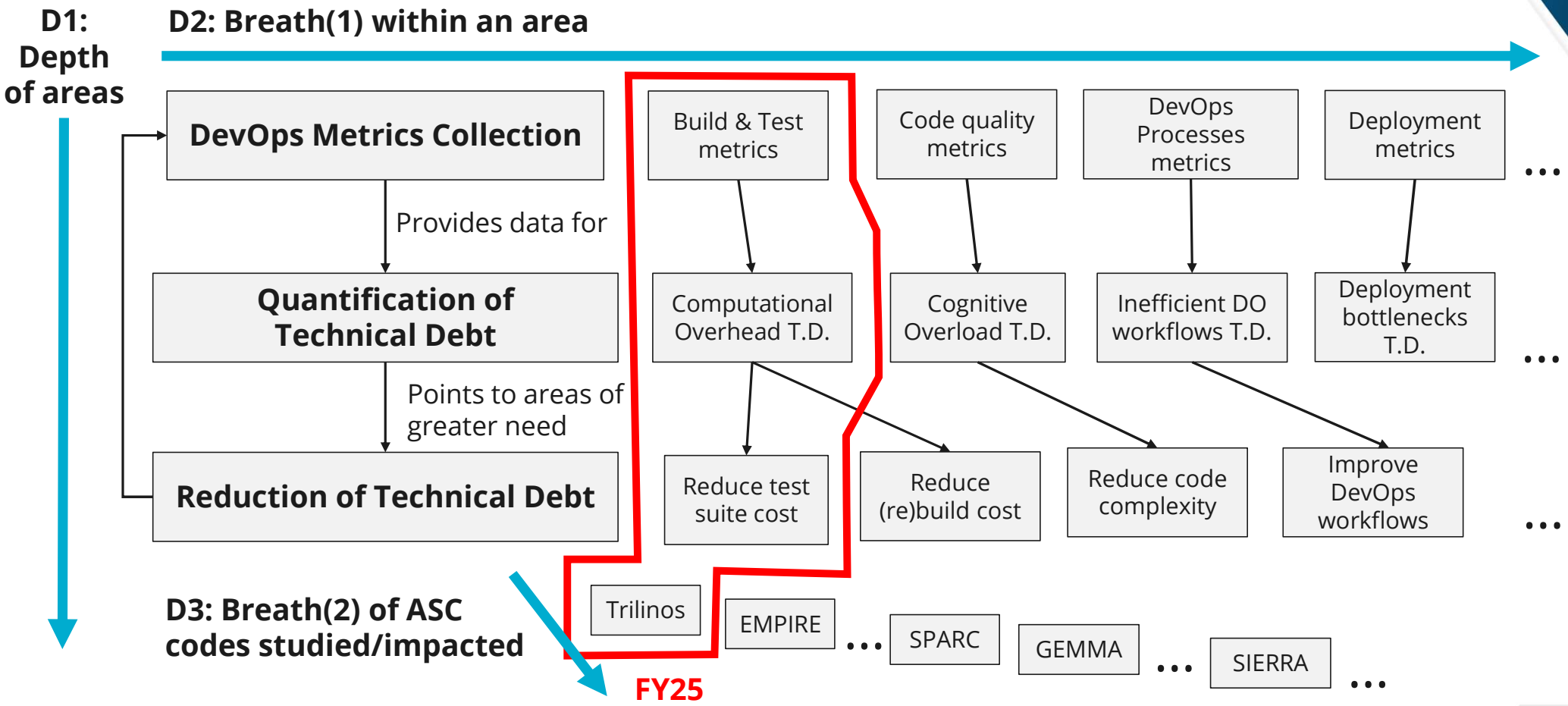
We must build a **foundation of expertise** in the deployment and practical application of emerging AI/ML technologies in DevOps and software development now so we can benefit from current and future advances in these technologies. Working directly with **AI/ML toolchains in real-world scenarios** is needed to better understand how to mature and tailor them to ASC's needs.

WHO CARES? IF YOU ARE SUCCESSFUL, WHAT DIFFERENCE DOES IT MAKE?

- **For Developers**: Faster development cycles, less time chasing bugs.
- **For Users**: New features and bug fixes appear faster, and less new bugs introduced.
- **For ASC Leadership**: More agility, better sustainability, lower computer resource usage to support development and testing, lower cost per new feature.



METRICS => QUANTIFY TECH DEBT => REDUCE TECH DEBT: MULTI-DIMENSIONAL SCALING BREATH VS DEPTH OF EFFORTS!



LOWER RISK, SHORT-TERM, LOWER-IMPACT EFFORTS VS. HIGHER RISK, LONGER-TERM, HIGHER-IMPACT EFFORTS

Lower-risk, shorter-term, less-invasive, lower-impact

- **Keep existing system-level tests as-is**:
  - Research selection of reduced test suite from existing tests while minimizing loss of coverage
  - Research partitioning of test suite run at different intervals/integration points
- **Reduce cost of existing system-level tests**:
  - Knobs: Reduce size of mesh, reduce number of iterations, start with closer initial guess, etc …
  - Research adjustment of knobs that maintains coverage but with lower test cost
- **Factor out code into unit test harness and add tests using Legacy Software Change Algorithm**:
  - Research usage of COTS refactoring tools on ASC-size codes
  - Research usage of LLVM/Clang AST for creation of custom C++ refactorings
  - Research extraction of characterization test cases from running full system-level tests
  - Research usage of AI tools to generate LLVM/Clang AST refactoring scripts for custom refactorings
  - Research usage of AI tools (using LLVM/Clang AST) to do refactorings to break dependencies and get code out into unit test harness
  - Research usage of AI/code-analysis tools to generate characterization test cases

Higher-risk, longer-term, more-invasive, higher impact

ASC DEVOPS RESEARCH - SCOPE & RESOURCES (2.25 FTE)

Develop novel AI-assisted approaches to reduce existing and future ASC software technical debt for decades to come => **Impact:** Improve speed, productivity, agility, and quality!

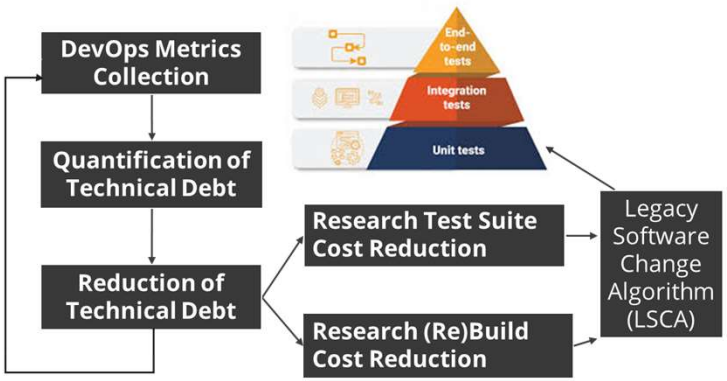
Key Project Plan Deliverables (FY25)

Q1: Select ASC codes and metrics for metrics collection activity.

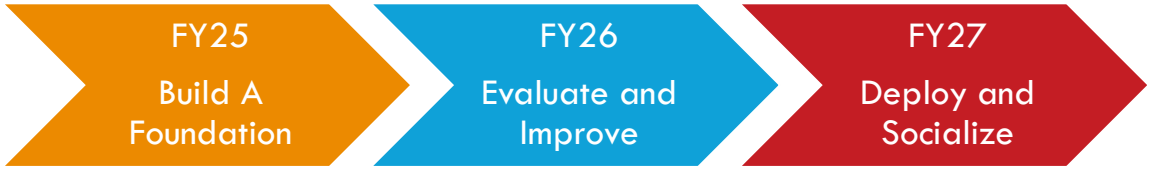
Q2: Gather selected metrics for selected codes.

Q3: Preliminary analysis of metrics collected.

Q4: Initial assessment of GenAI for test generation and refactoring.



ASC DevOps Research: AI-Assisted Tech Debt Quantification and Reduction		
Role	**Name**	**FTE**
PI, applied AI Research	Reed M. Milewicz	0.45
Applied AI/Test Reseach, RSE	Jim M. Wllenbring	0.45
Applied AI/LSCA Research, RSE	Roscoe A. Bartlett	0.45
RSE Support	Anderson Chauphan	0.50
ASC Code Team Support	TBD, multiple people	0.00
Interns/Post Docs	TBD	0.50
FY25 SUM		2.35

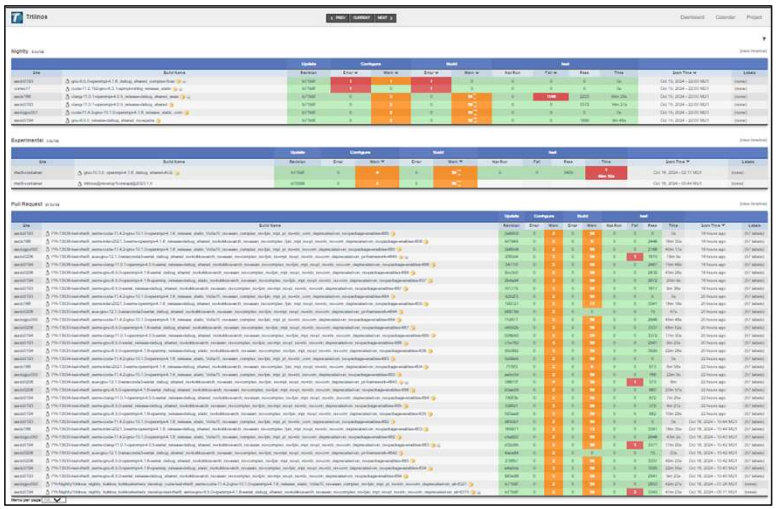


Whitepaper: "AI-Assisted Research to Reduce ASC Software Technical Debt"

FY25 SCOPE AND TRILINOS EFFORTS

FY25 EFFORTS: METRICS INFRASTRUCTURE

- **Kitware: Extending CMake/CTest/CDash for extended build and test metrics:**
  - ***Target-based build and test metrics***: start and end time-stamps, wall-clock time, max load, max RAM, size of file produced, …
  - ***Categorize build targets based on type:*** object files, library files, executables, others (e.g. custom commands)
  - ***Categorize build targets based on lib vs. test*** (e.g. Using target labels in CMake)
- **Kitware: Fixing/extending CTest/CDash coverage data collection/handling :**
  - Partition coverage between library and tests (e.g. Labels)
  - Filter and download coverage data



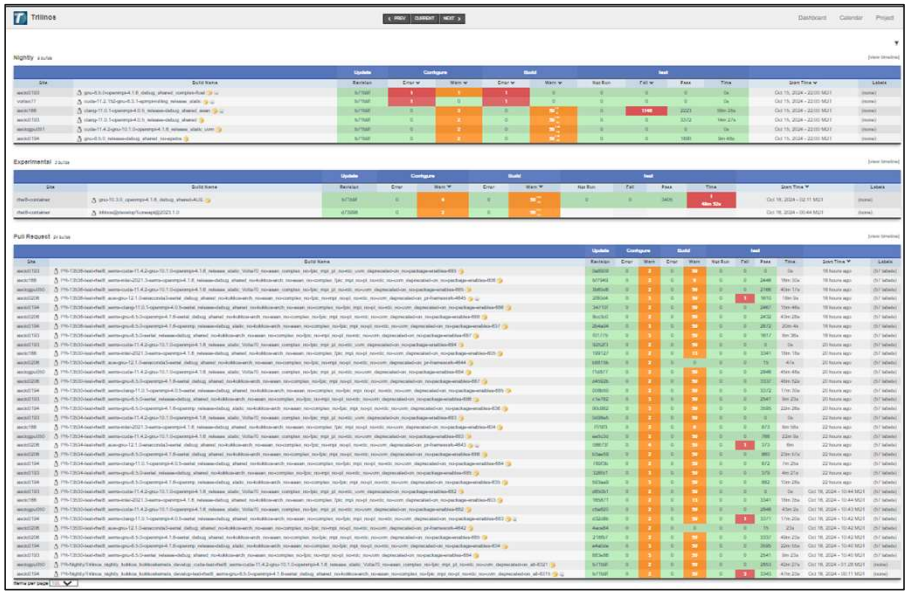
- **SNL ASCDOR: Develop tools to download, store, and analyze build and test data**
  - Tools to download build and test metrics data from CDash
  - DB for long-term storage of build and test metric data
  - Analysis tools for build and metric data, qualification of computational overhead TD

FY25 EFFORTS: METRICS AND TRILINOS?

- ***Turn on CMake/CTest build and test metrics in some Trilinos nightly builds:***
  - NOTE: Requires CMake/CTest 'master', dual submits to trilinos-cdash-qual.sandia.gov (running CDash 'master')
  - Update some official Trilinos PR and/or Nightly builds (some, most, all)?
  - or, ASCDOR team set up some new builds for this purpose?

- **Add matching Trilinos nightly coverage builds:**
  - Must submit to trilinos-cdash-qual.sandia.gov
  - Needed to direct testing/refactoring efforts

- **Additional build and test metrics builds and coverage builds:**
  - As needed, run additional Trilinos builds and submit to trilinos-cdash-qual for other build configurations



FY25 EFFORTS AND TRILINOS? QUANTIFCATION OF TECH DEBT

Quantification of Computational Overhead Technical Debt:

- Identify packages that consume the most build resources and show which targets in each package are the most expensive in each metric
- Identify the object files, libraries, executables, etc. that consume the most build resources (i.e. CPU time, RAM, generated file size, etc.)
- Identify packages and tests that consume the most resources for testing
- Compute cost of build and tests for Trilinos PR and Nightly builds
  - => Translate to $$ cost per build, per PR, per day, per year?

Computational Overhead Technical Debt

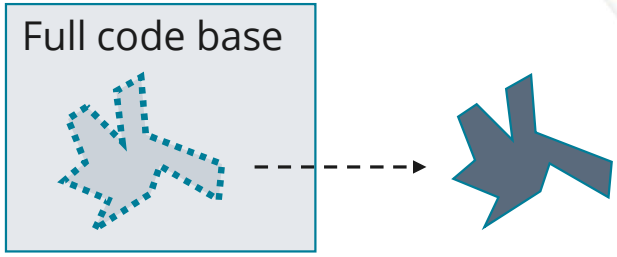
Selection of targeted Trilinos packages/builds/etc as initial targets for refactoring/testing research:

- Based on build data, test data, coverage, and interest from Trilinos package developers
  - **=>** select candidate packages/code to experiment with

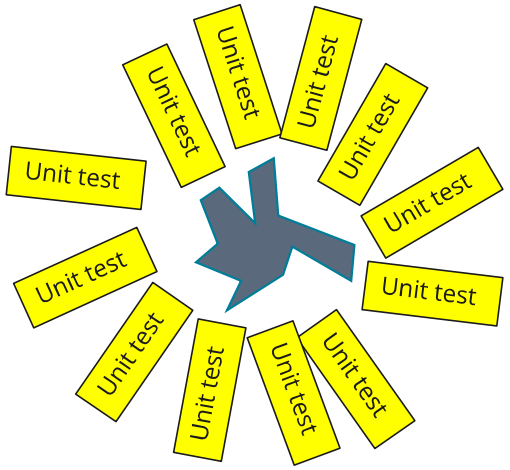
FY25 EFFORTS AND TRILINOS? TECH DEBT & LSCA RESEARCH

Research applying LLM/GenAI to Legacy Software Change Algorithm (LSCA) algorithm

- ***Research factoring code out into unit test harness:***
  - Perform manual refactorings (measure human time, create benchmarks)
  - Research application of automated refactorings with LLVM/Clang AST scripts
  - Research generation of LLVM/Clang AST scripts using GenAI
- ***Research generation of characterization tests:***
  - Manually add characterization tests (measure human time, create benchmarks)
  - Research generation of tests using GenAI, LLVM/Clang AST, …



2. Break dependencies
Minimal refactors to allow extraction of code into unit test harness
⇒ LLVM/Clang AST with **AI**?



3. Cover extracted code with characterization tests
=> Code analysis & test generation using AI?

TRILINOS DEVELOPER FEEDBACK?

Any general feedback/suggestions/concerns for this research effort?

Who is interested in collaborating/providing input to this effort for Trilinos?

EXTRAS

WHAT DEPARTMENT 1424 OFFERS ASC DEVOPS

1424 We research, develop, and deploy software capabilities to accelerate science and engineering.

1420 We lead and influence future computing paradigms to serve the nation.

1400 We advance frontiers of computing research and deliver innovative mission solutions.

Department 1424 provides R&D capabilities spanning fundamental and applied research to deployed and maintained software solutions. **We intend to act as a force multiplier** for ASC DevOps through pathfinding, prototyping, and proving out DevOps capabilities (tools, methodologies, etc.).

ASC DEVOPS RESEARCH – FY26, FY27, DEPLOYMENT OF CAPABILITIES

What happens in FY26?

- Research deeper into approaches that look promising from FY25 work
- ???

What happens in FY27?

- Continue to refine promising approaches identified in FY25 and FY26
- Circle back and investigate the other approaches from FY25 that were not worked in FY26.
- ???

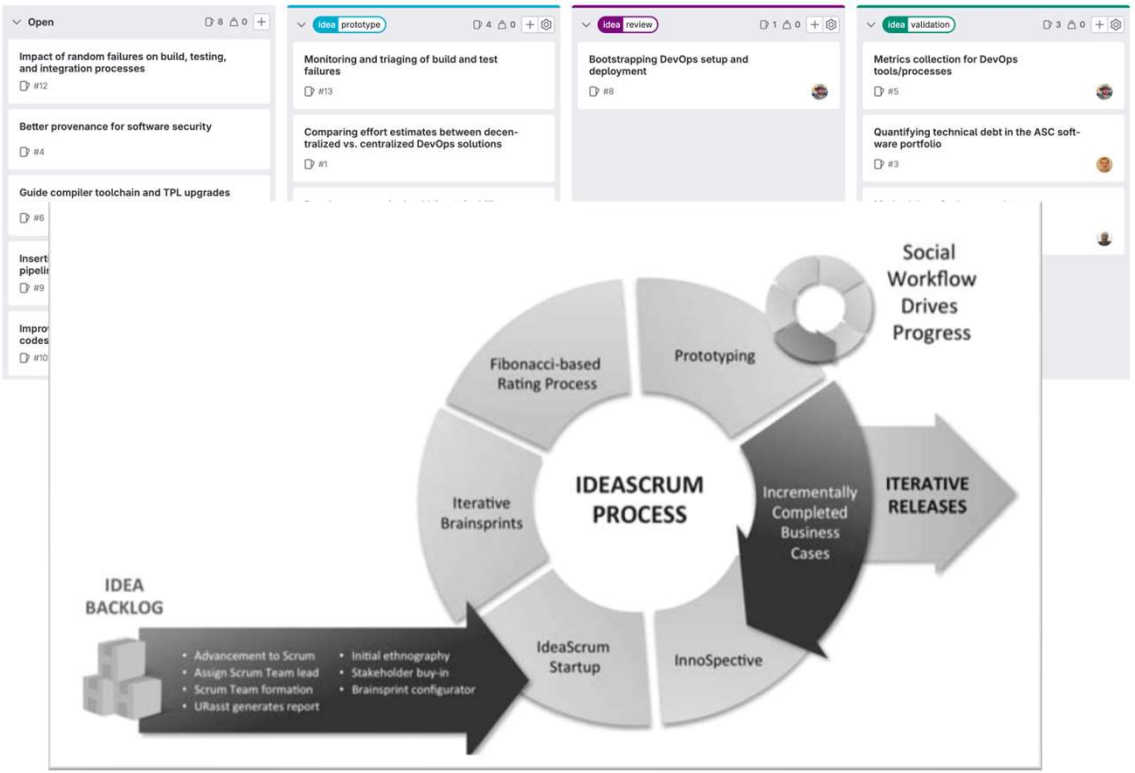
How will these approaches be refined and deployed to the ASC code teams?

- Transition them to and work with the ASC DevOps Raise the Bar team?
- ???

OUR METHODOLOGY

We iteratively developed a wide variety of proposal topics which could be of value to ASC DevOps

- Used an Agile Innovation approach (IdeaScrum) to identify, develop, and refine research proposals.
- We conducted interviews key ASC DevOps stakeholders to gather insights and requirements.
- Based on these discussions, we generated a broad range of research ideas. Each idea was then developed into user stories, which were iteratively refined through cycles of prototyping, review, and validation.



PROPOSAL IDEA DEVELOPMENT

16 Ideas Generated → **8** Selected for Prototyping and Elaboration → **4** Selected for Internal Review → **3** Selected for Validation

- Measuring the Impact of Random Failures
- Better Provenance for Software Security
- Guide Compiler Toolchain and TPL Upgrades
- Inserting Security Tools into DevOps Pipelines
- Metrics Collection for DevOps Tools/Processes
- Simplifying Configuration Management
- Enabling and Improving Continuous Deployment
- Improving Requirements Traceability with LLMs
- Methodology for Large-Scale Test Management
- Complex Test Rejuvenation for Heterogeneous Architectures
- Monitoring and Triage of Build and Test Failures
- Quantifying Technical Debt in the ASC Software Portfolio
- Comparing Decentralized vs. Centralized DevOps Solutions
- Developing a Community Health/Sustainability Dashboard
- Better Developer Support for Package Managers
- Bootstrapping DevOps Setup and Deployment

IDEAS SELECTED FOR FULL DEVELOPMENT

A common overarching theme: the pursuit of efficiency, reliability, and continuous improvement through DevOps

- **Metrics Collection for DevOps Tools/Processes**: Developing a comprehensive suite of targeted metrics to measure the effectiveness of ASC DevOps solutions. This initiative aims to provide actionable insights that can guide decision-making and align ASC practices with industry standards.
- **Identifying and Quantifying Technical Debt in the ASC Software Portfolio**: Creating a methodology to assess technical debt within ASC software projects. This research seeks to quantify the impact of technical debt on development efficiency and point to areas where larger improvements are possible.
- **Methodology for Large-Scale Test Management**: Investigating approaches and tools to reduce the cost of test suites while not sacrificing coverage. This ideas focus on enhancing the utility and efficiency of testing practices within ASC, enabling teams to better leverage DevOps automation.

Slide 27

BR0 Is this really true? How is anything we are proposing going to "enable teams to better leverage DevOps automation"?

Bartlett, Roscoe, 2024-04-24T13:17:13.981

THE BIG PICTURE: HOW THESE RESEARCH STORIES ARE RELATED

- **Enhanced Observability:** Establishing a comprehensive metrics collection framework is fundamental to gaining visibility into the DevOps lifecycle. This observability is crucial for understanding the current state of software development, deployment processes, and overall system health.
- **Informed Decision-Making**: The insights gained from increased observability enable organizations to identify areas for improvement that may be hindering efficiency, reliability, or scalability (e.g., technical debt).
- **Strategic Management of DevOps-related Assets:** With a clear understanding of the DevOps landscape (like through metrics) and identified areas of improvement (like technical debt), organizations can then strategically manage their DevOps-related assets (like test suites).

BR0

Slide 28

BR0 Is this not mostly just paraphrase of the bullet of the last slide?
Bartlett, Roscoe, 2024-04-24T13:19:15.634

BACKGROUND: CATEGORIZATION OF TECHNICAL DEBT

- **A) Cognitive Overload Technical Debt:**
  - Low intra-module cohesion, high inter-module coupling, long/confusing functions, high complexity, poor design/poor naming, no comments or incorrect comments, other "code smells", etc ....
- **B) Computational Overhead Technical Debt:**

Easier to measure and track with metrics!

  - **Build-related technical debt**:
    - Long build times (e.g. deeply templated C++ code, implicit template instantiation, deep stacks of included header files)
    - Long rebuild times (e.g. most changes need to be made to header files which propagate to all build targets)
    - High RAM usage per target not allowing using all the cores on a node to build
  - **Test run technical debt**:
    - No/poor unit or system tests to allow development and testing pieces of code in isolation
    - Long test runtimes, multiple cores or processes per test
    - High usage of expensive/scarce accelerator resources (e.g. GPUs)

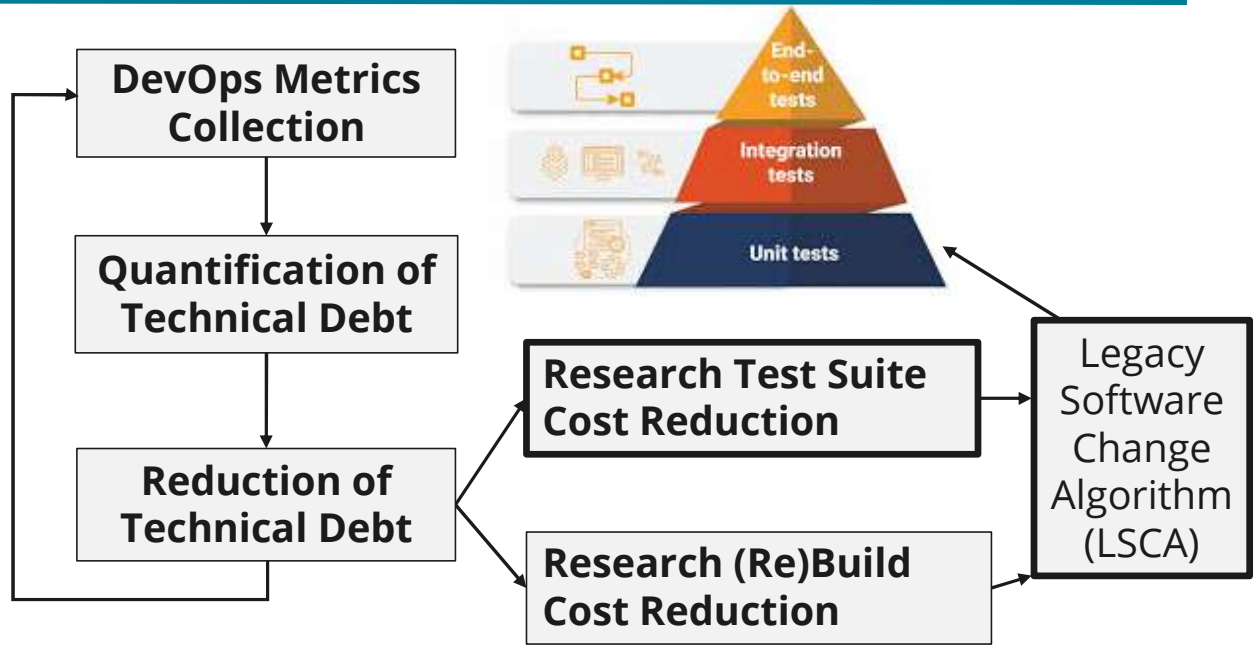
BRO

BOOTSTRAPPING THE ASC DEVOPS RESEARCH EFFORT

Idea-Scrum Process

16 Ideas Generated (Interviewed key stakeholders) → **8** Selected for Prototyping and Elaboration → **4** Selected for Internal Review → **3** Selected for Validation

1. **Metrics Collection for DevOps Tools/Processes**: Develop suite of metrics to provide insights and guide decision-making.
2. **Identifying and Quantifying Technical Debt in the ASC Software Portfolio**: Quantify the impact of technical debt on development efficiency to facilitate improvement.
3. **Methodology for Large-Scale Test Management**: Enhance efficiency of testing practices within ASC.



Goal: Develop novel AI-assisted approaches to reduce existing and future ASC software technical debt for decades to come => **Impact:** Improve speed, productivity, agility, and quality!

Slide 30

BR0 Perhaps we don't need this slide given the slide before this and the slide after this?
Bartlett, Roscoe, 2024-06-04T16:02:34.861

BR0 0 Or, this could be a summary slide towards the end? But given the number of slides that we have, perhaps we should cut this slide out?
Bartlett, Roscoe, 2024-06-04T16:06:36.794

ASC DEVOPS RESEARCH - SCOPE & RESOURCES (2.35 FTE)

Develop novel AI-assisted approaches to reduce existing and future ASC software technical debt for decades to come => **Impact:** Improve speed, productivity, agility, and quality!

- Metrics Collection for DevOps Tools/Processes
- Identifying and Quantifying Technical Debt
- Methodology for Large-Scale Test Management

Key Project Plan Deliverables (FY25)

Q1: Select ASC codes and metrics for metrics collection activity.

Q2: Gather selected metrics for selected codes.

Q3: Preliminary analysis of metrics collected.

Q4: Initial assessment of GenAI for test generation and refactoring.

ASC DevOps Research:
AI-Assisted Tech Debt Quantification and Reduction

Role	Name	FTE
PI, applied AI Research	Reed M. Milewicz	0.45
Applied AI/Test Reseach, RSE	Jim M. Wllenbring	0.45
Applied AI/LSCA Research, RSE	Roscoe A. Bartlett	0.45
RSE Support	Anderson Chauphan	0.50
ASC Code Team Support	TBD, multiple people	0.00
Interns/Post Docs	TBD	0.50
FY25 SUM		2.35

SYNERGIES WITH ASC DEVOPS RAISING THE BAR EFFORTS

- ASC DevOps Raising the Bar FY25 Testing Planning Team Support and Synergies
  - Proposing a test suite reduction effort
    - Smaller scope
  - References our plan to quantify and reduce technical debt
    - Stating this as a need for the Raising the Bar effort, but looking at ASC DevOps Research to propose and execute
  - General interest in metrics effort to motivate decisions