Software Architecture Practices

John Klein, PhD

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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.

DM20-0821

Software Architecture Practices

Architecture Evaluation

- Holistic
- Single Quality Attribute Focus

Model-based Engineering

- Architecture-centric Virtual Integration

Post-Evaluation

- Implementation Conformance

CMU SEI is a DoD Federally Funded Research and Development Center

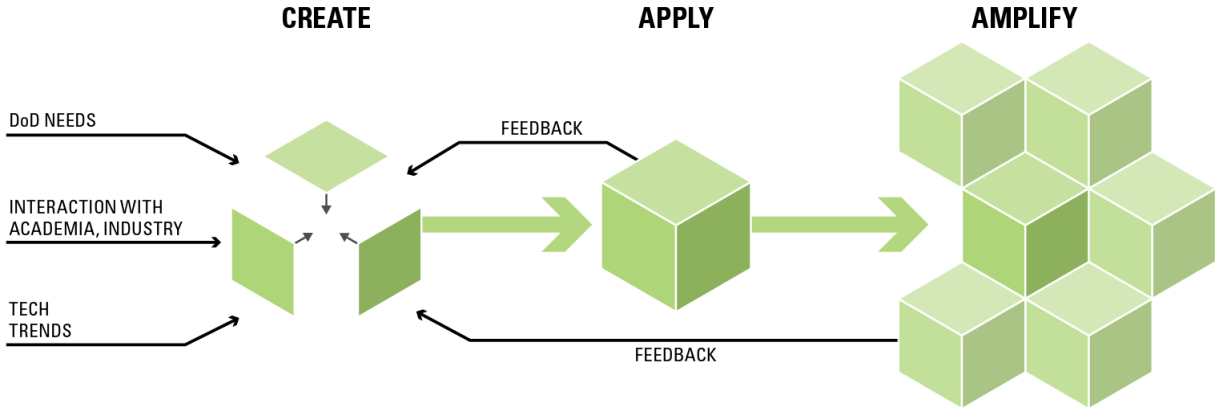
- Our mission: Engineering and securing software
- Established in 1984 at Carnegie Mellon University
- ~635 employees
- Offices in Pittsburgh and DC, with locations near customer facilities in MA, TX, and CA
- ~$145M in annual funding (~$20M USD(R&E) 6.2 and 6.3 Line funding)

SEI: Execution Strategy

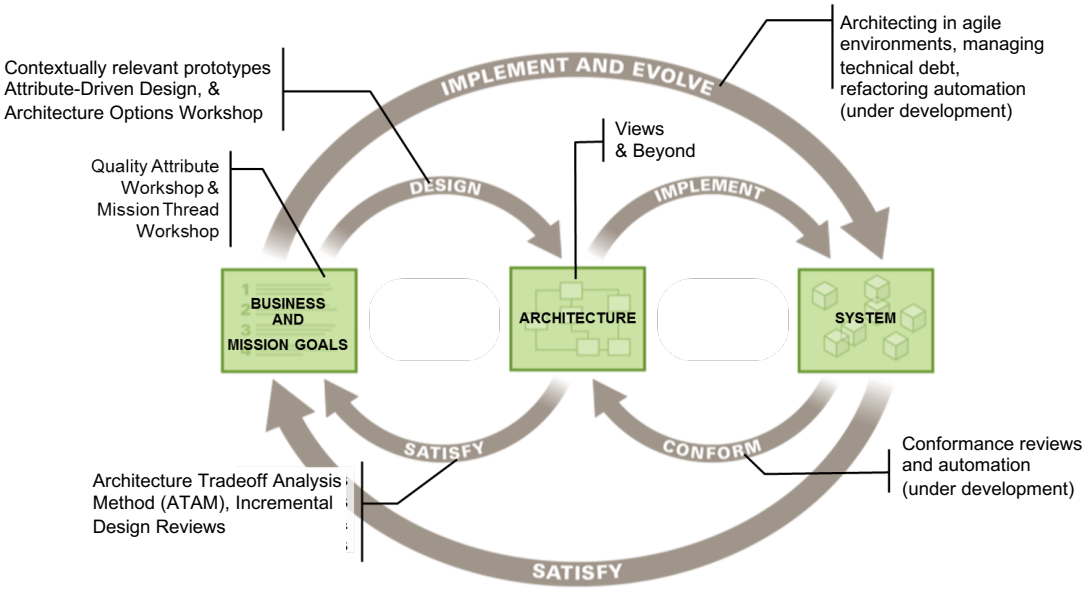
R&D work executed in collaboration with academia, labs, industry

Applying work with customers and stakeholders to solicit feedback

Mature technologies disseminated / transitioned / taught



Establishing a Discipline of Software Architecture



Range of methods and practices applicable at different points in the development lifecycle.

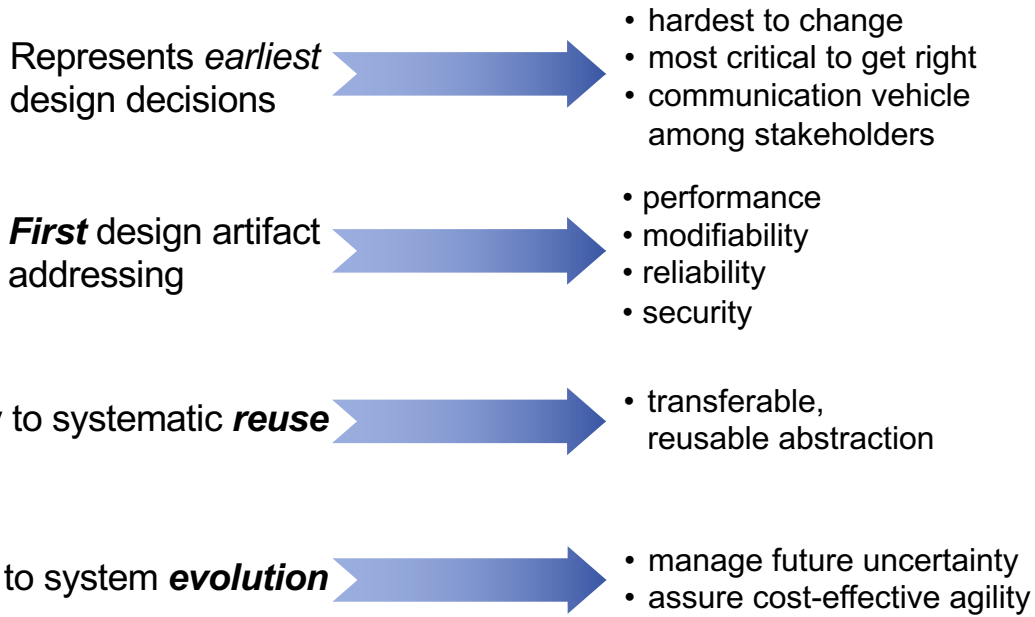
- Domains of expertise include IT, C2, tactical, and health informatics
- Technology expertise includes IoT, big data, digital twin, cloud, and machine learning

10+ courses, available in a mix of public, on-site, and eLearning options.

3 professional certificates.

A collection of books for wide dissemination.

Why is Software Architecture Important?



The **right architecture** paves the way for system **success**.
The **wrong architecture** usually spells some form of **disaster**.

Architecture-related Activities

Architecture-related activities include the following:

- creating the **business case** for the system
- understanding the **requirements**
- **creating and/or selecting** the architecture
- **documenting and communicating** the architecture
- **analyzing or evaluating** the architecture
- setting up the appropriate **tests and measures** against the architecture
- **implementing** the system based on the architecture
- ensuring that the implementation **conforms** to the architecture
- **evolving** the architecture so that it **continues to meet business and mission goals**

Architecture-related activities run throughout the iterative and incremental development life-cycle.

Architecture-related Activities

Architecture-related activities include the following:

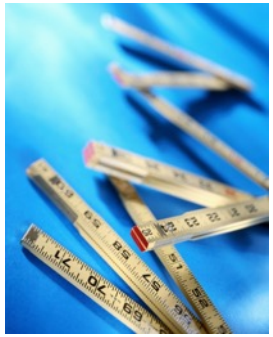
- creating the **business case** for the system
- understanding the **requirements**
- **creating and/or selecting** the architecture
- **documenting and communicating** the architecture
- **analyzing or evaluating** the architecture
- setting up the appropriate **tests and measures** against the architecture
- **implementing** the system based on the architecture
- ensuring that the implementation **conforms** to the architecture
- **evolving** the architecture so that it **continues to meet business and mission goals**

Architecture-related activities run throughout the iterative and incremental development life-cycle.

Software Architecture Practices

Holistic Architecture Evaluation Using Quality Attribute Scenarios

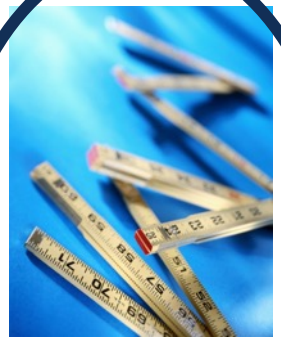
Things to Establish for Architecture Evaluation



An objective measure

An understanding of the architecture

The analysis



An objective measure

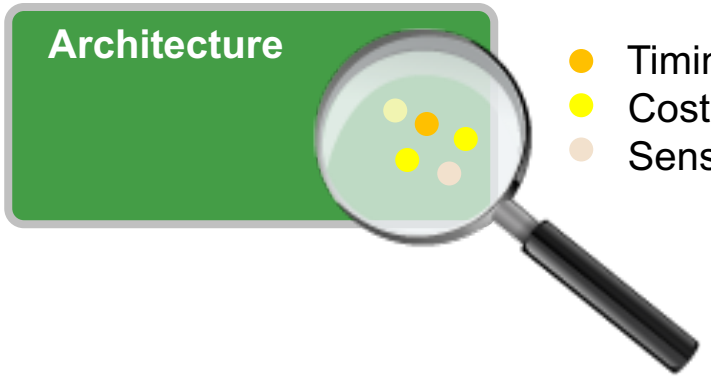
An understanding of the architecture

The analysis

Building the Yardstick

Without a yardstick to measure, every architecture is good or bad.

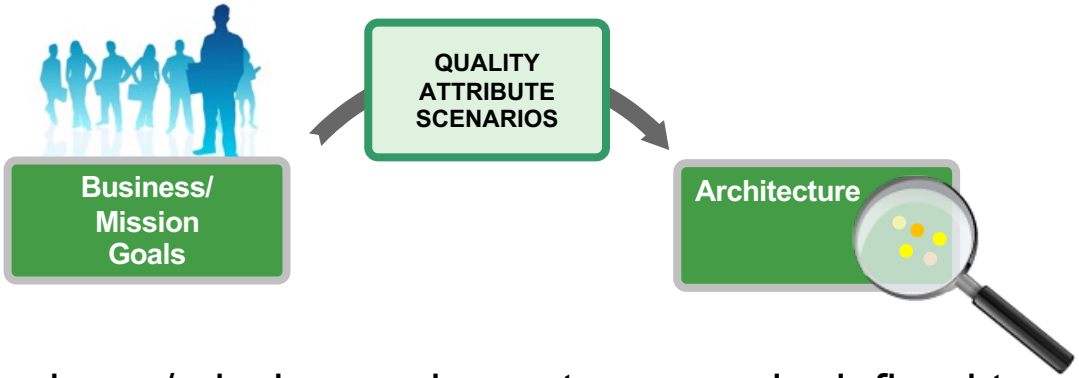
Will the designed system solution meet the requirements for business or mission success?



- Timing
- Cost of change
- Sensitive data

We need requirements for these quality attributes

Quality Attribute Scenarios



In most cases business/mission goals are too vaguely defined to actually be able to measure their achievement

Quality attribute scenarios bridge goals and architecture quality attribute requirements

An architecture that satisfies its functional and quality attribute requirements will produce a successful business or mission outcome

The Yardstick

Example scenarios:

- A customer requires a new auction algorithm. A developer implements and integrates that function within one day of effort.
- An error occurs in a fielded system. The system recovers from the error without manual intervention within 5 seconds.

Analysis

The architecture evaluation process has to answer the questions:

- Do the quality attribute properties of the identified components support the given quality attribute scenario sufficiently?
- Is the negative impact of the chosen architecture approaches on other quality attribute scenarios acceptable?

If the answers are documented, then the examining the documentation is sufficient for the evaluation.

If not, the architect has to provide the answers in an interview.

The evidence provided must be sufficient to convince quality attribute experts and the stakeholders.

The ATAM

The SEI developed the Architecture Tradeoff Analysis Method (ATAM) and has applied it to architectures for systems of wide-ranging sizes and domains.

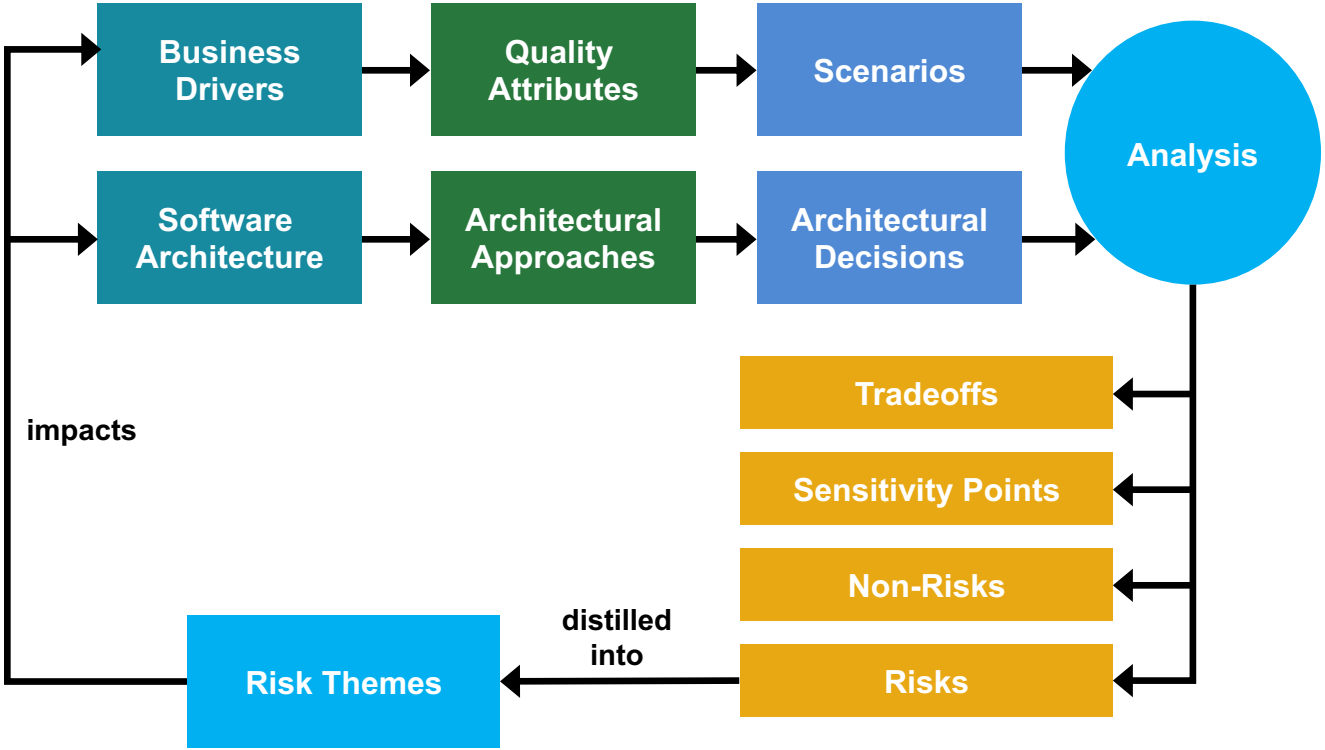
The purpose of the ATAM is to assess the consequences of architectural decisions in light of quality attribute requirements and business goals.

The ATAM brings together three groups in an evaluation:

1. a trained evaluation team
2. an architecture's "decision makers" (architect, senior designers, project managers, customers)
3. representatives of the architecture's stakeholders

   (users, integrators, operators, certifiers, …)

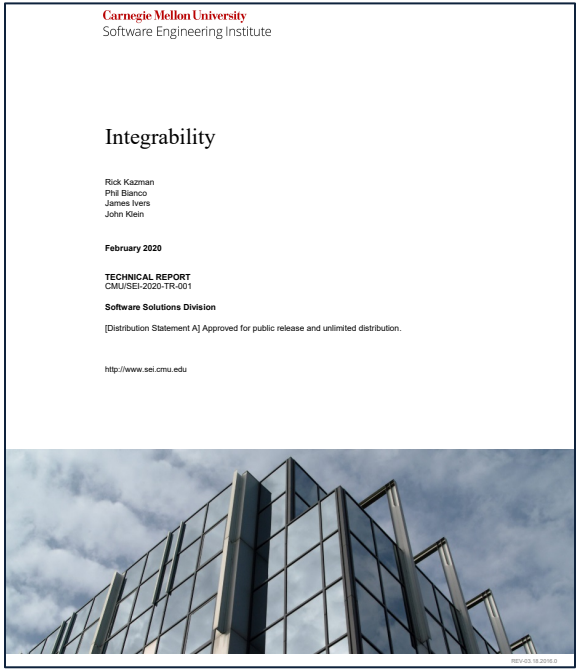
Conceptual Flow of the ATAM



Software Architecture Practices

Single Quality Attribute-based Architecture Evaluation

Focusing on a Single Quality Attribute



Series of technical reports, one per quality attribute

Use to construct requirements and validate architecture against requirements

Outline for each report

- Definitions
- Measures
- Scenario templates and examples
- Mechanisms – patterns and tactics
- Analysis methods – questionnaires, checklists, measurement
- Evaluation playbook

http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=637375

Quality Attribute-based Evaluation Playbook

Phase	Step
Preparation	Step 1–Collect artifacts
Orientation	Step 2–Identify the mechanisms used to satisfy the requirement
	Step 3–Locate the mechanisms in the architecture
	Step 4–Identify derived decisions and special cases
Evaluation	Step 5–Assess requirement satisfaction
	Step 6–Assess impact on other quality attribute requirements
	Step 7–Assess the cost/benefit of the architecture approach

Guide experts to perform repeatable analysis

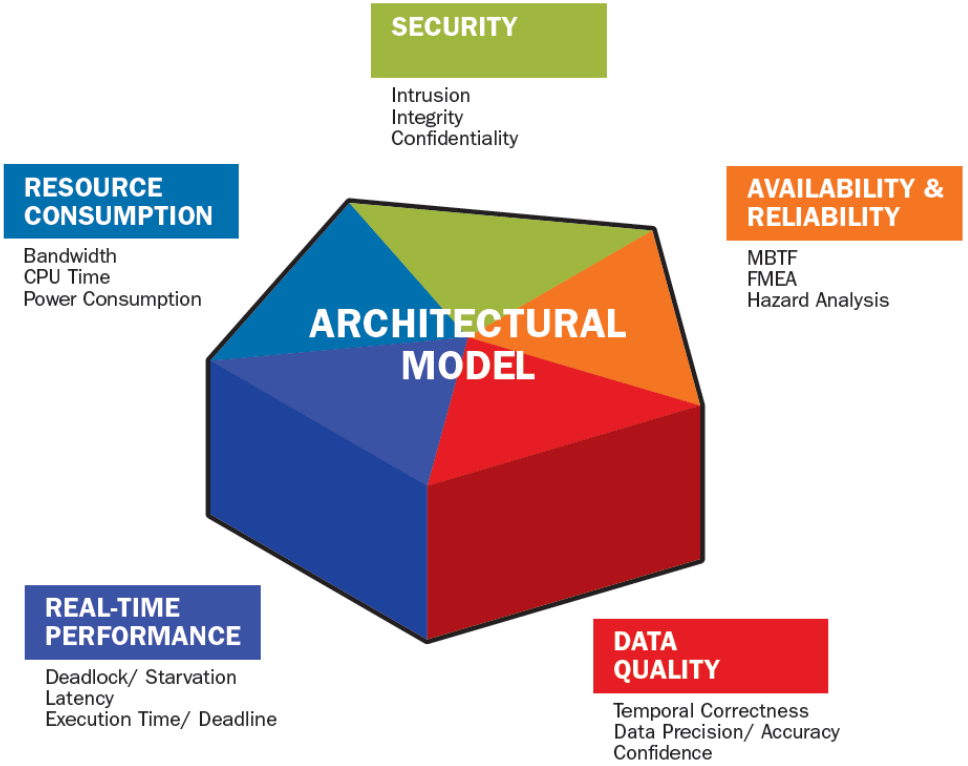
Applicable at any point in the development lifecycle

Applies checklists, questionnaires, and measurements

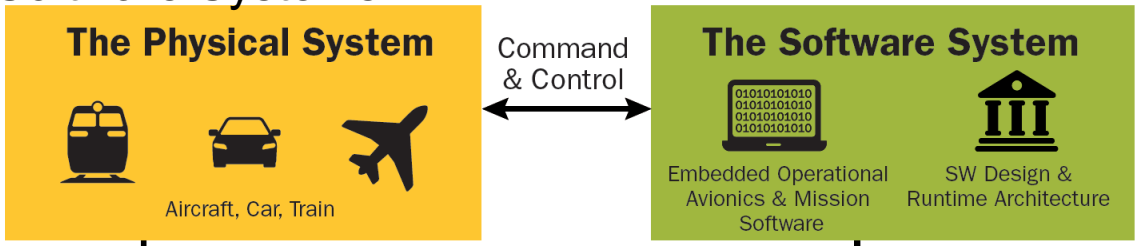
Software Architecture Practices

Model-based Engineering and the Architecture Analysis & Design Language (AADL)

Analyzable Architecture Models Discover System Level Issues Early in Development



Architecture Analysis & Design Language (AADL) Standard Targets Embedded Software Systems



SAE International
AS 5506 Standard Suite

Standards provide long-term industry-wide solutions to support multi-organization model-based engineering

In 2008 Aerospace industry initiative chose AADL over SysML and other notations as it specifically addresses embedded software systems

AADL captures mission and safety critical embedded software system architectures in virtually integrated analyzable models to discover system level problems early and construct implementations from verified models

SAE International AADL Standard Suite (AS-5506 series)

Core AADL language standard [V1 2004, V2 2012, V2.2 2017]

- Focused on embedded software system modeling, analysis, and generation
- Strongly typed language with well-defined semantics for execution of threads, processes on partitions and processor, sampled/queued communication, modes, end to end flows
- Textual and graphical notation
- Revision V3 in progress: interface composition, system configuration, binding, type system unification

Standardized AADL Annex Extensions

- Error Model language for safety, reliability, security analysis [2006, 2015]
- ARINC653 extension for partitioned architectures [2011, 2015]
- Behavior Specification Language for modes and interaction behavior [2011, 2017]
- Data Modeling extension for interfacing with data models (UML, ASN.1, …) [2011]
- AADL Runtime System & Code Generation [2006, 2015]

AADL Annexes in Progress

- Network Specification Annex
- Cyber Security Annex
- FACE Annex
- Requirements Definition and Assurance Annex
- Synchronous System Specification Annex

Architecture-centric Virtual Integration (ACVIP) with AADL

Finding Problems Early (AMRDEC/SEI)

- Summary: 6 Week Virtual Integration on CH47 using AADL
- Result: Identified 20 major integration issues early
- Benefit: Avoided 12-month delay on 24-month program

CH47 Chinook

Commercial Aircraft Industry Addresses Embedded Software System Challenge

- Summary: Industry Consortium Matures Virtual System Integration
- Result**:** Proof of virtual integration concept in 2008-09 led to ten year maturation commitment
- Benefit: ROI estimate $2B savings on $10B aircraft through 33% early detection

Transforming procurement (Joint Multi-Role)

- Summary: Industry/DoD mission system architecture demonstrations using ACVIP
- Result: Pre-integration fault identification
- Benefit: 10X reduction integration test cost

Improving System Security (DARPA / AFRL)

- AADL applied to Unmanned Aerial Vehicles & Autonomous Truck
- Result: AADL models enforced security policies and were used to auto build the system
- Benefit: Combined with formal methods verification, prevented security intrusion by a red team

High Assurance Cyber Military Systems (HACMS)

Unmanned Quadcopter

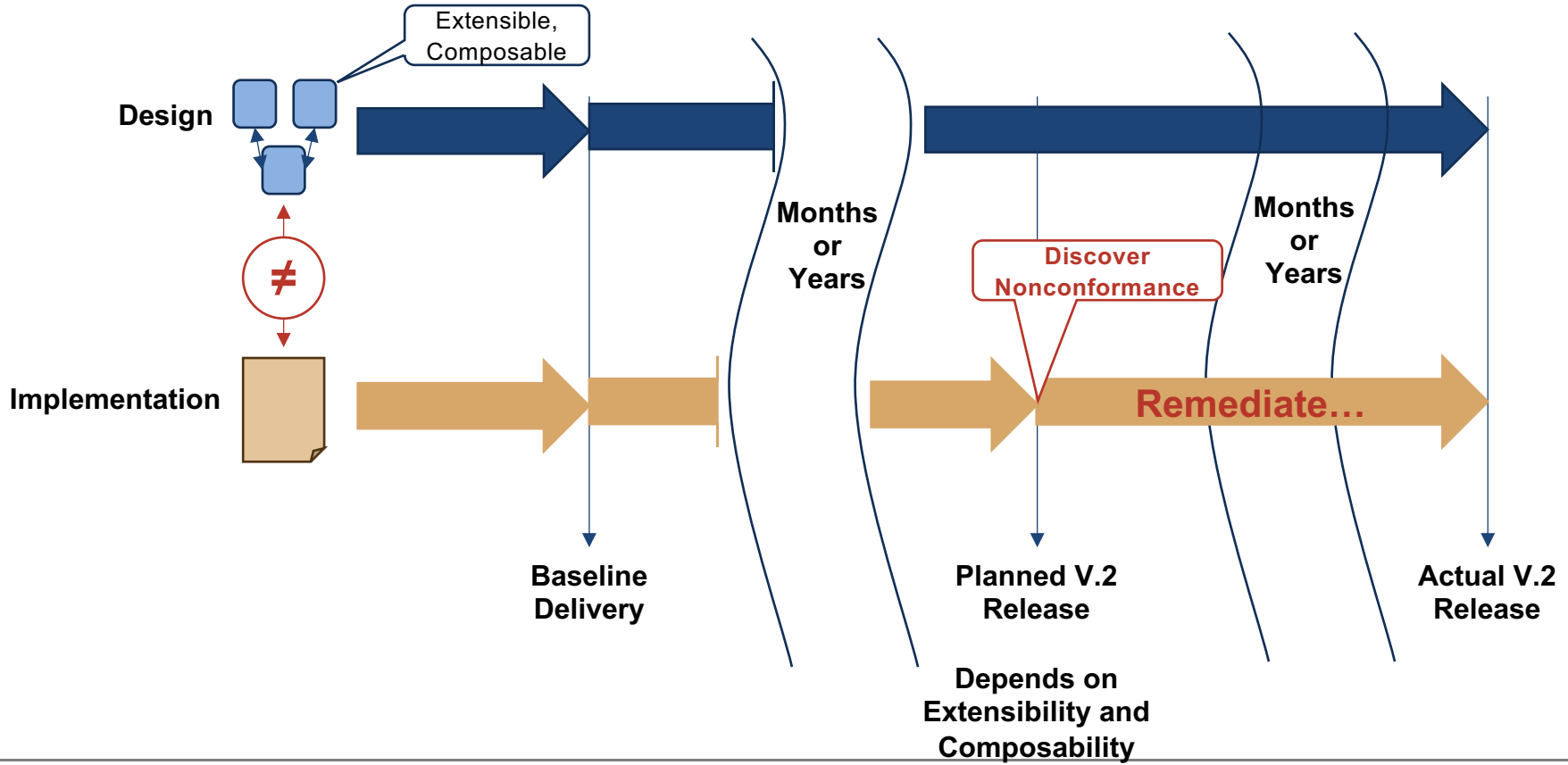
TARDEC Autonomous Truck

Unmanned Little Bird

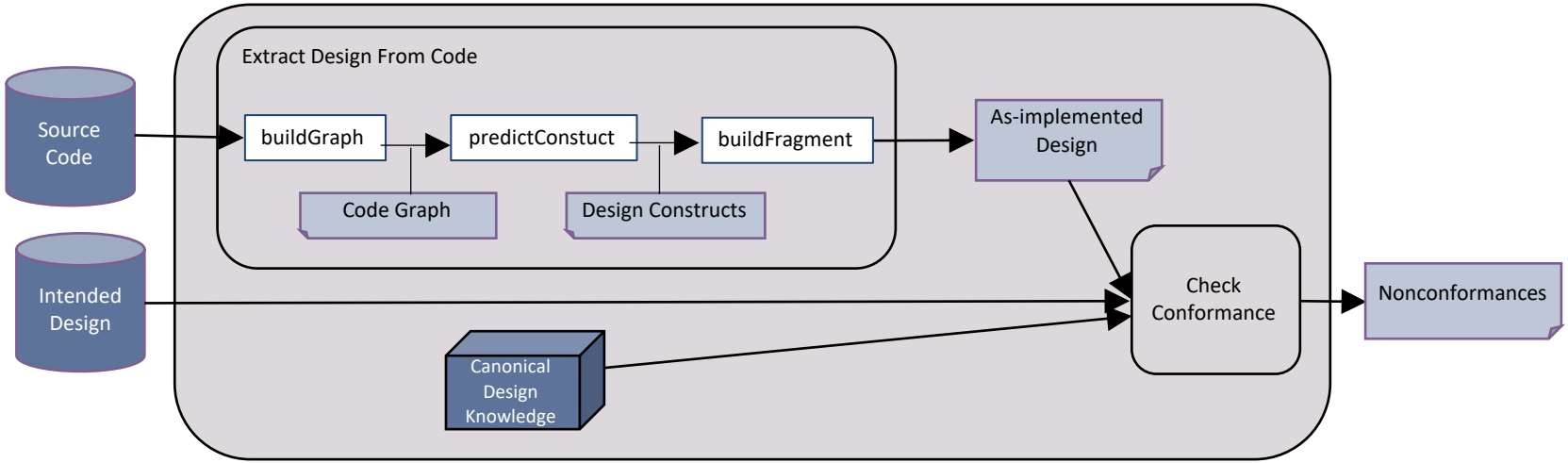
Software Architecture Practices

Assessing Implementation Conformance to Architecture Design

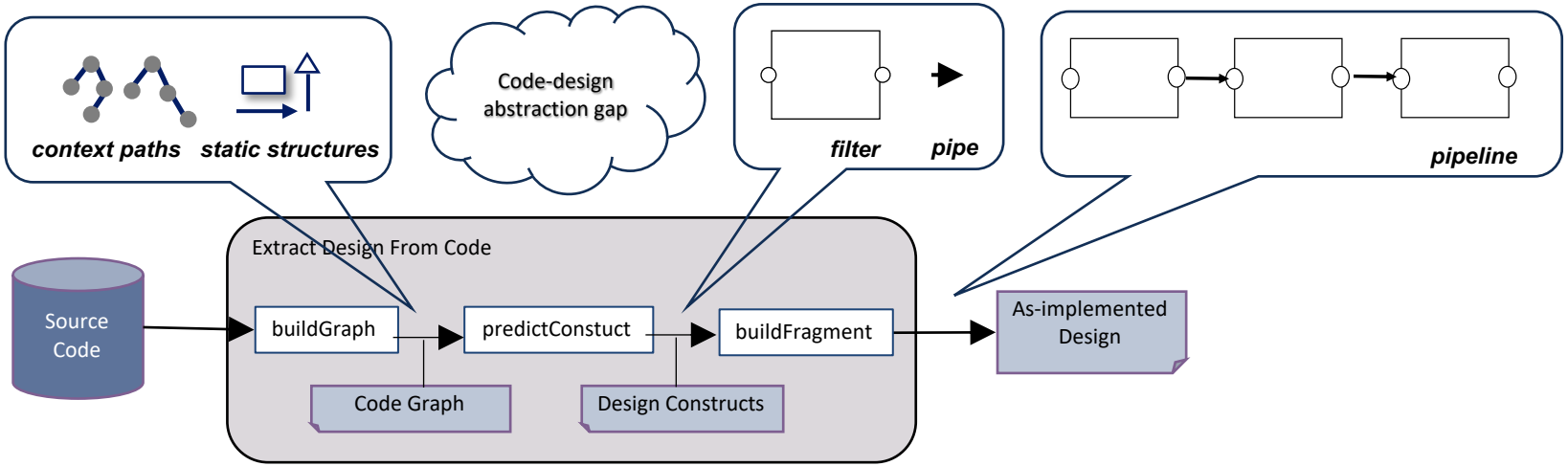
Software Nonconformance Problem



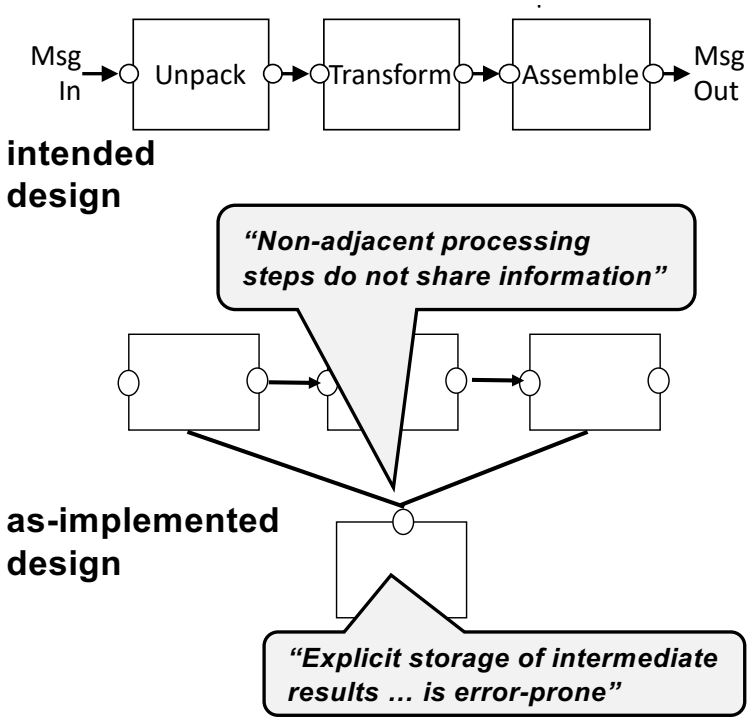
Building on Code Analysis, Software Architecture, Machine Learning, and Continuous Integration



Code-Design Abstraction Gap



Check Conformance

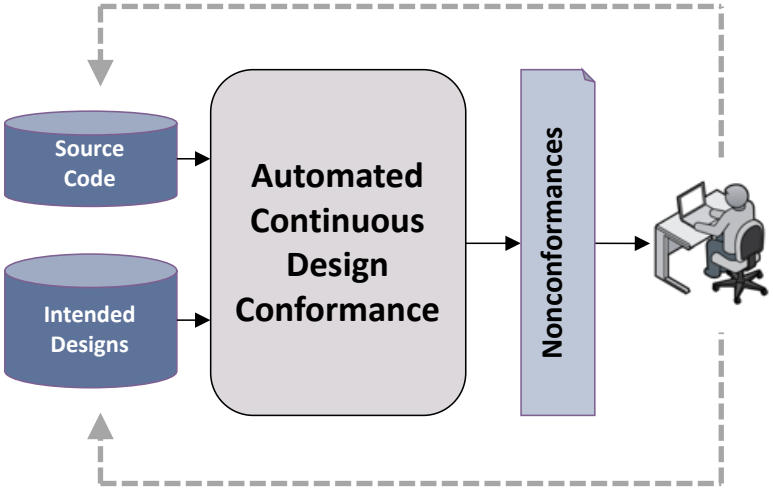


Design fragments represented as graphs enable automatic detection

Detect nonconformances

- check graph of extracted design fragment for agreement with intended design fragment to locate inconsistencies
- augment with checks against canonical design knowledge relevant to design fragment

Continuous Integration Workflow



Detecting nonconformances produces greatest value when issues are exposed close to the time of injection

Integrate with Jenkins CI tool to enable an empirical evaluation of the use of automation

Use developer feedback in rating each nonconformance to improve results

- *correctness* – improve design extraction by providing new labeled data
- *significance* – improve adaptive filtering by capturing context-specific rules and exceptions to intended design

Software Architecture Practices

Wrap-up

Closing Thoughts

Evaluate your architecture early and often

- Low ceremony peer review
- High ceremony SARB review

Tailor evaluation approach based on…

- Precision of quality attribute requirements – order-of-magnitude can be good enough to make architecture tradeoffs
- Depth and breadth of architecture representation – models, box-and-line diagrams, whiteboard sketches
- Confidence needed in the evaluation findings – e.g., early design tradeoff vs. critical availability risk

Consider model-based architecture approaches for continuous evaluation of some types of quality attribute requirements

”The code is the truth…” – the implementation has to conform to the architecture

For more information

John Klein, PhD
Senior Member of the Technical Staff
Software Engineering Institute

jklein@sei.cmu.edu

https://resources.sei.cmu.edu/library/author.cfm?authorID=3271

David Scherb
Business Development Manager
Software Engineering Institute

dscherb@sei.cmu.edu