How to Address the Problem of Poorly-Defined Requirements in Software System Design

NOVEMBER 12, 2025

Dr. Lori Flynn
Lyndsi Hughes

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Copyright 2025 Carnegie Mellon University.

This material is based upon work supported by the Department of War under Air Force 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 opinions, findings, conclusions, and/or recommendations contained in this material are those of the author(s) and should not be construed as an official US 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.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

CERT®, ATAM® and Carnegie Mellon® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM25-1452

How to Address the Problem of Poorly-Defined Requirements in Software System Design

Engineering Functional Software is Hard

Carnegie Mellon University Software Engineering Institute

[Distribution Statement A] Approved for public release and unlimited distribution.

Engineering Quality Functional Software is Harder

Carnegie Mellon University Software Engineering Institute

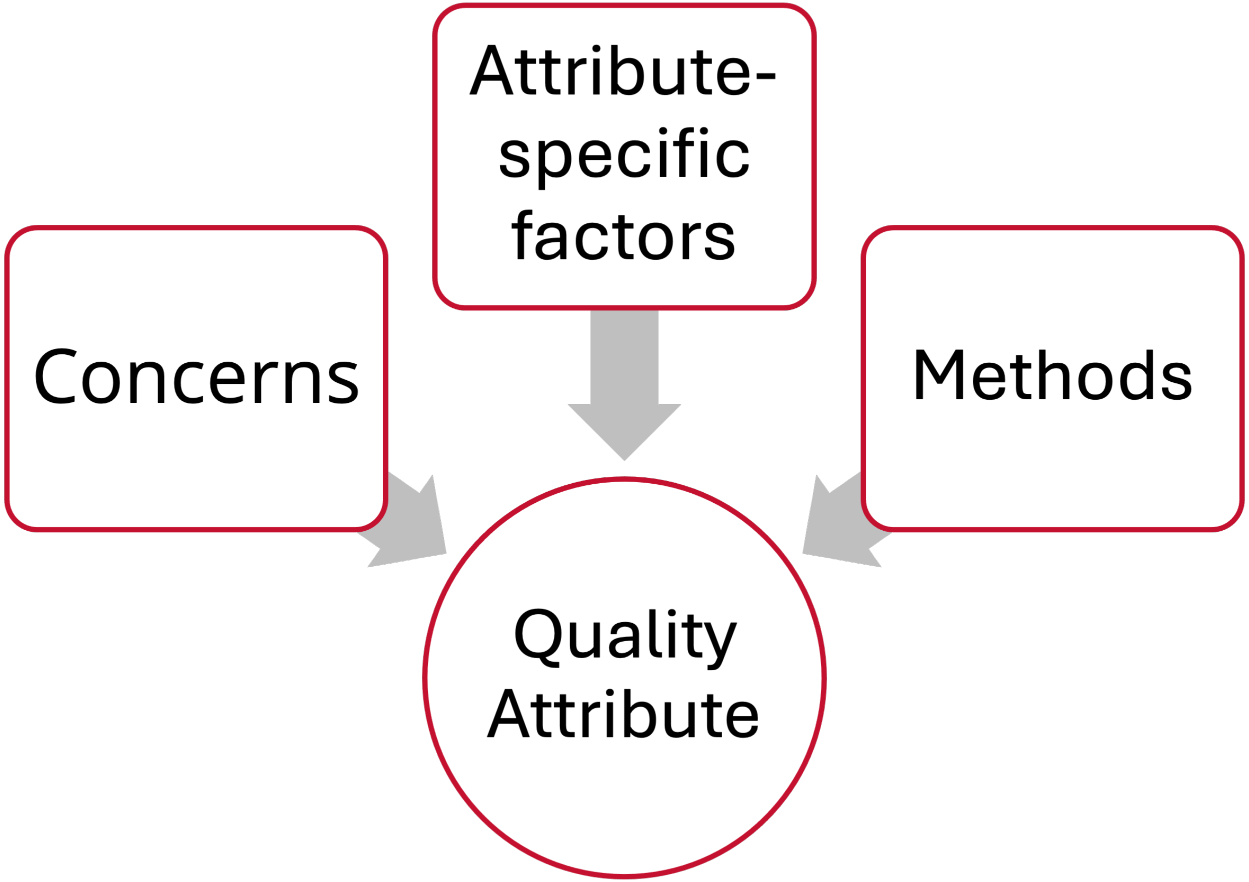
Software quality is "the degree to which software possesses a desired combination of attributes." [1]

1: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=237006

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Building Blocks of Software Quality Attributes

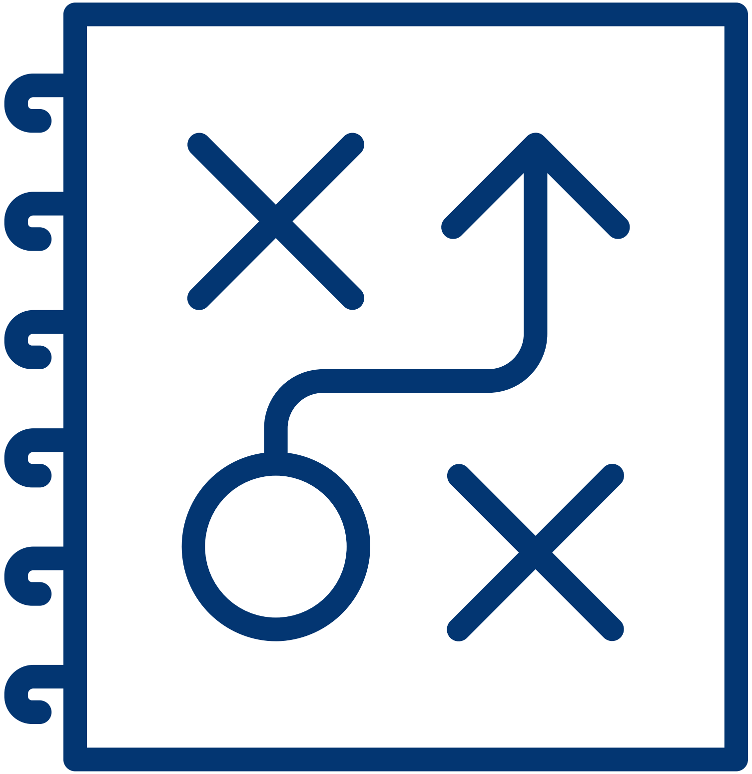


© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

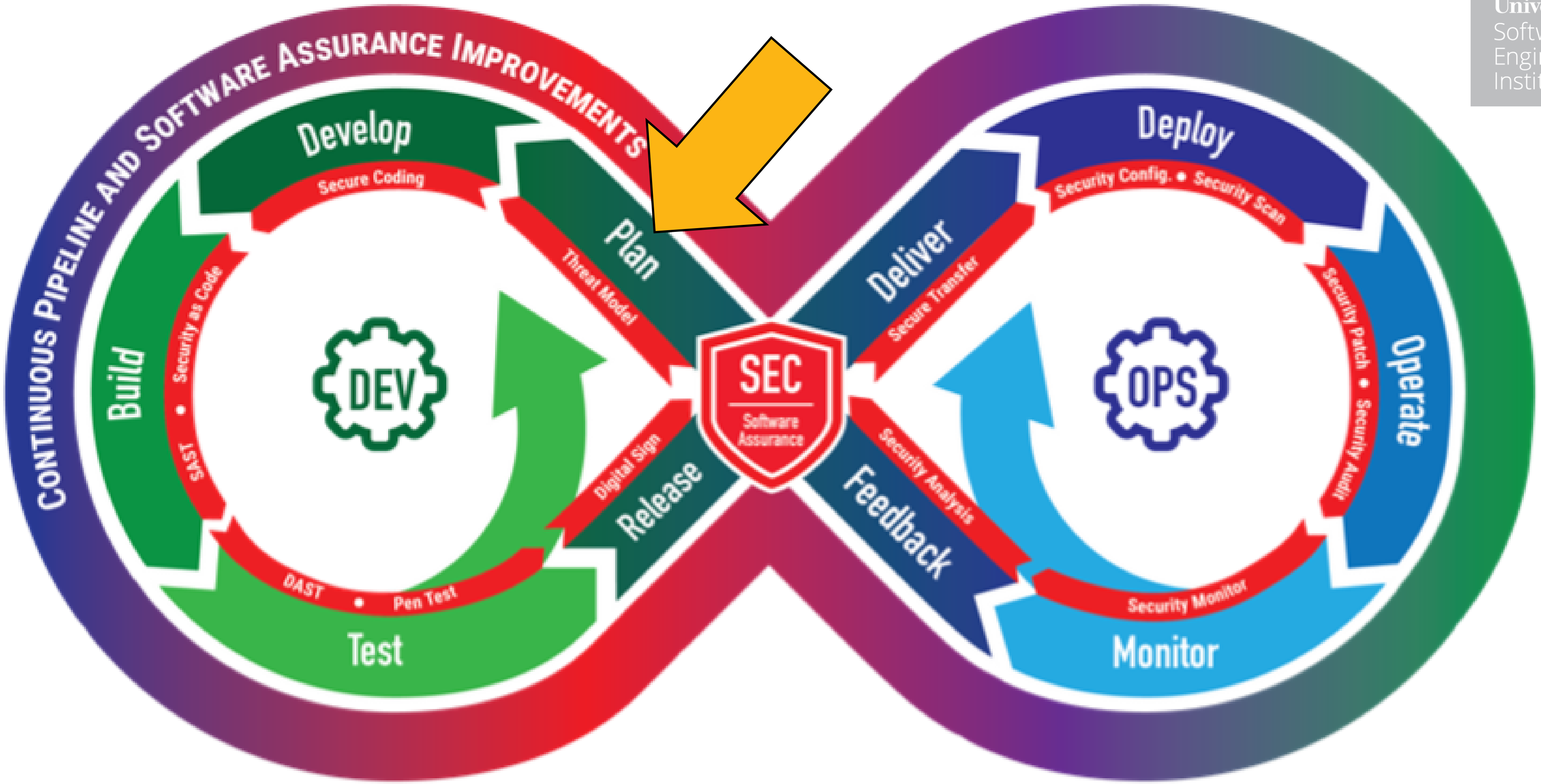
Our Requirements Definition Playbook

- Lean Into Modern Software Engineering Practices
- Analyze Requirement Tradeoffs
- Use Math to Validate and Support Your Decisions



© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

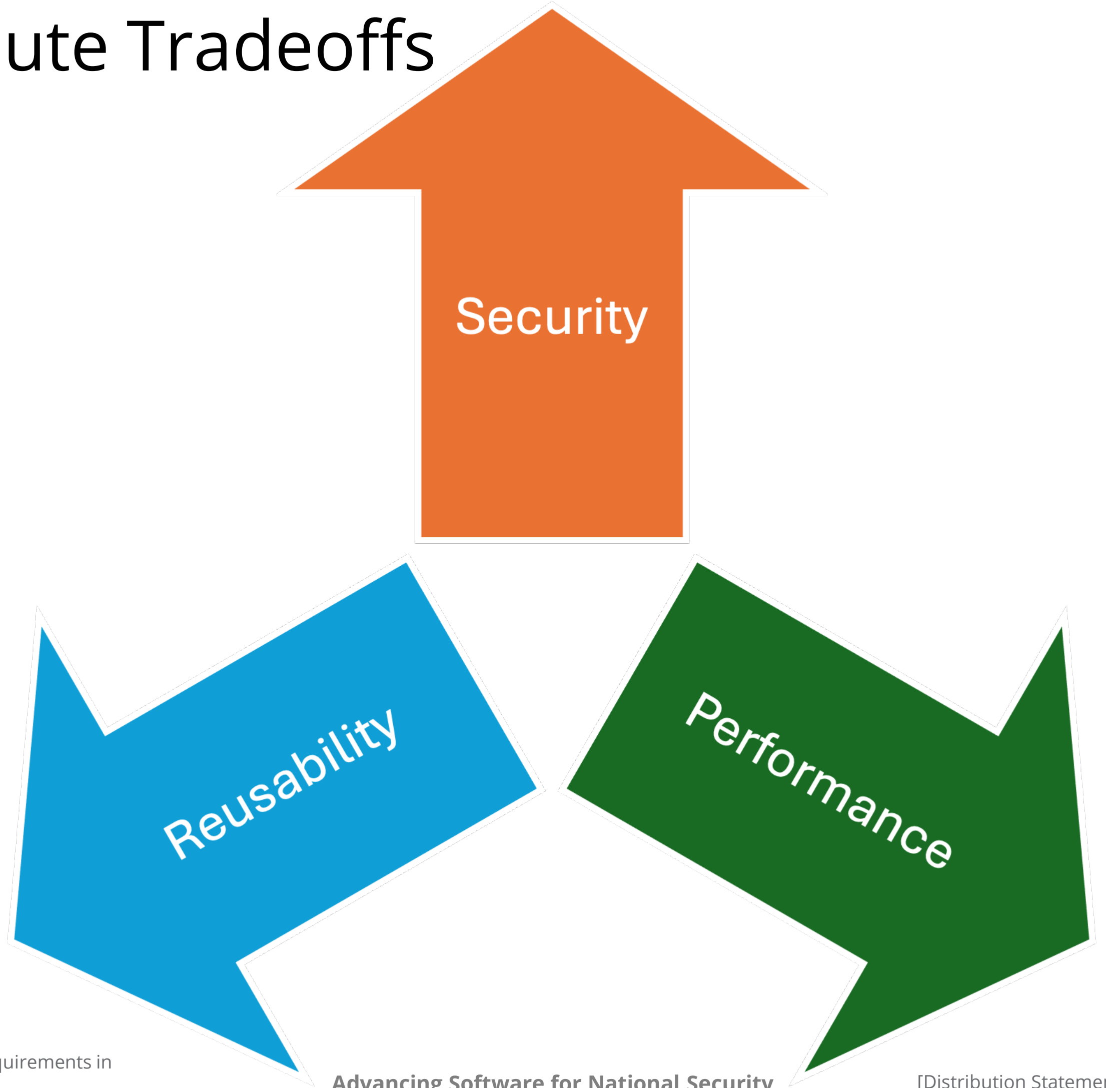


Carnegie Mellon University Software Engineering Institute

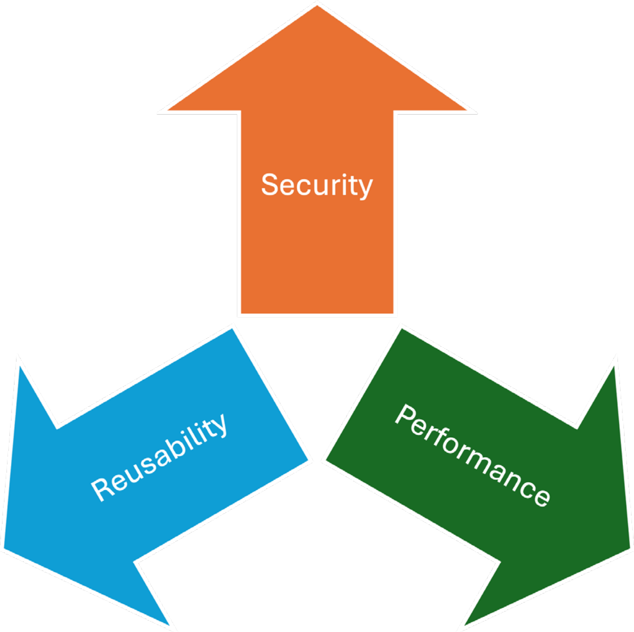
© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Quality Attribute Tradeoffs



Quality Attribute Prioritization Process



Methodology: Architecture Tradeoff Analysis Method (ATAM) [2]

Purpose: "To assess the consequences of architectural decisions in light of quality attribute requirements."

Goals: To elicit, concretize, and prioritize the driving quality attribute requirements.

In Practice: Provides structure for decision making about trade-offs.

2: https://insights.sei.cmu.edu/library/atam-method-for-architecture-evaluation/

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

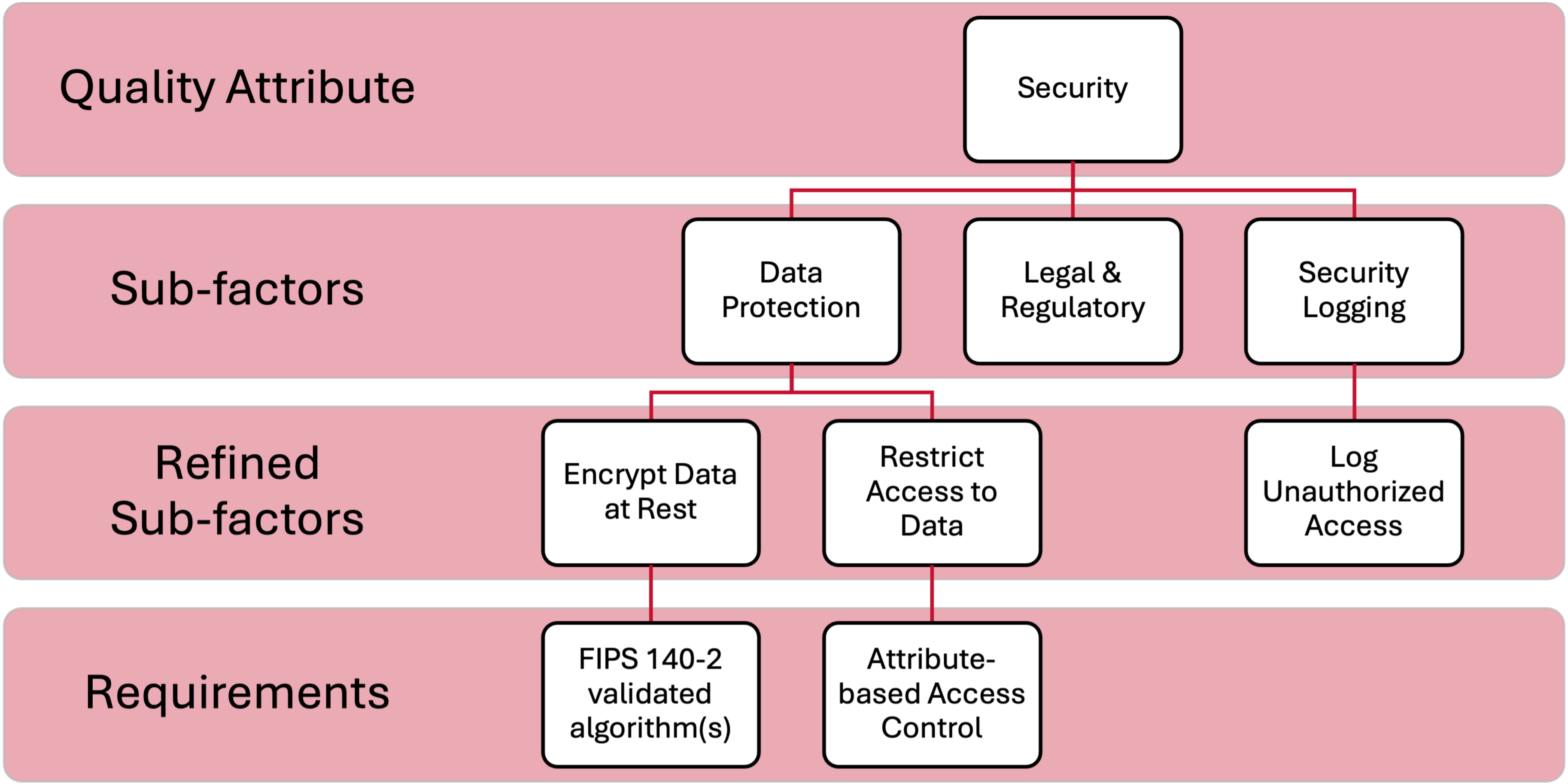
Security Quality Attribute Sub-factors

- Authentication & Authorization
- Disaster Recovery
- Third-party Dependencies
- Security Logging
- Legal & Regulatory
- Data Protection
- Encryption

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Building a Utility Tree



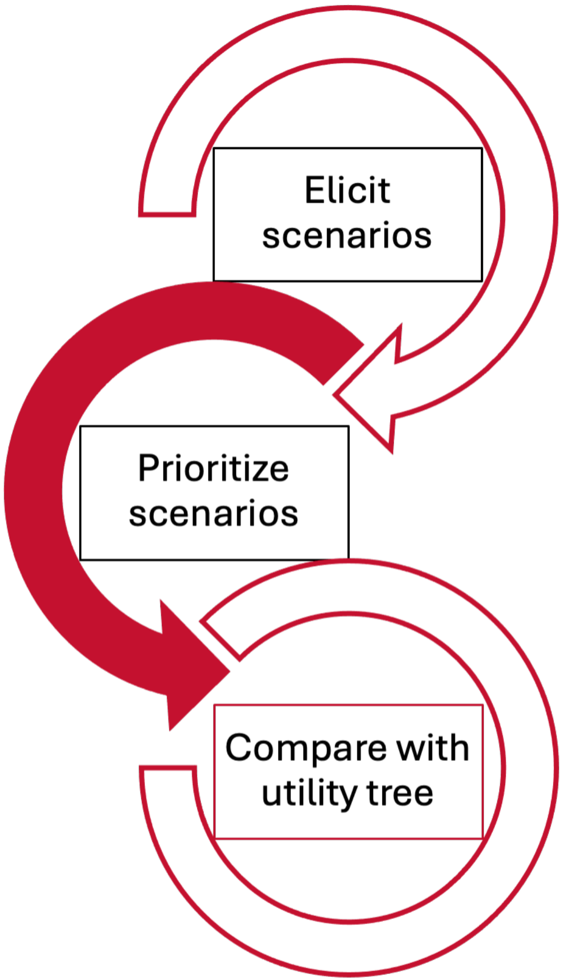
© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Brainstorming Scenarios

Goals:

- Represent stakeholders' interests
- Understand quality attribute requirements



© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Requirements Analysis Outcomes

Carnegie Mellon University Software Engineering Institute

Tradeoffs are known

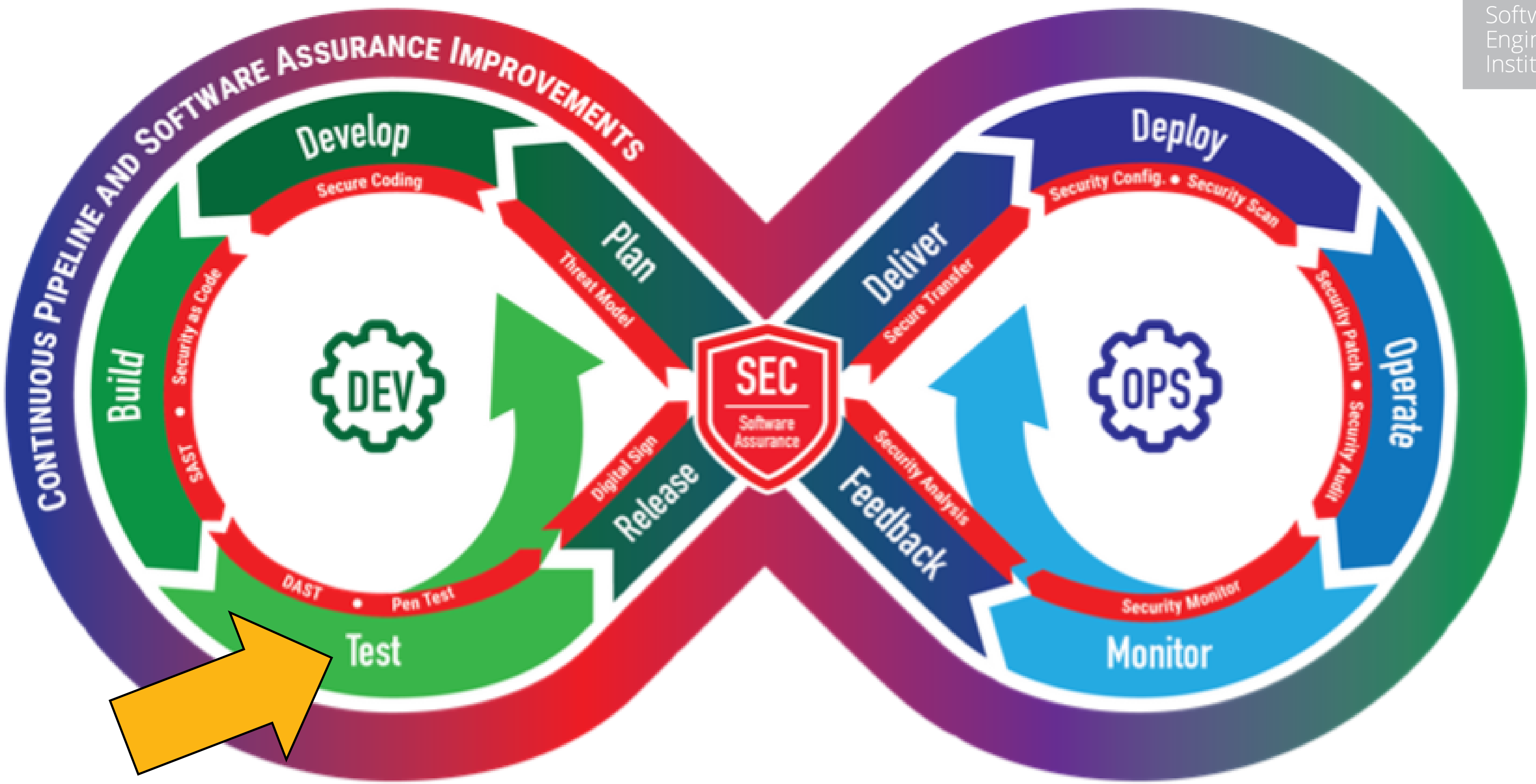
Threats to quality are mitigated

Requirements are:

- Precisely defined
- Measurable
- Prioritized
- Ready to be handed off to architects or developers

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.



© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Traditional Goals of Software Testing

- **Unit testing:** validate small modules of code
- **Regression testing:** validate new doesn’t break existing
- **Performance testing:** app handles anticipated loads?
- **Integration testing:** validate interoperability
- Plus *many* more…

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Modern Improvements to Software Testing

Carnegie Mellon University Software Engineering Institute

Static Application Security Testing

- aka SAST
- White-box testing
- Analyzing code for weaknesses, e.g., potential buffer overflow
- And more…

Dynamic Application Security Testing

- aka DAST
- Black-box testing
- Analyzing exposed interfaces, e.g., penetration testing, fuzzing
- Code coverage
- And more…

Interactive Application Security Testing

- aka IAST
- Inserts sensors into system
- Detects known vulnerabilities by measuring internal performance
- Integrated with development tools at development time

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Still Many Threats to Testing Success

Carnegie Mellon University Software Engineering Institute

Unaddressed Challenges

- Incomplete testing procedures
- Manual testing procedures
- Inadequate testing tools

Negative Outcomes

- Inconsistent test results can negatively impact software quality
- Poor software quality can negatively impact business success

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Designing Your Test Plan

Test what you care about: **Your Requirements!**

When designing your tests:

- Get automated
- Get objective
- Get creative

Example Measurement Tool: Silent Sentinel

- Combines software profiling and dynamic testing
- Observes changes made by the program under test in a sandbox environment
- Provides data about the program under test and the observed system changes to analysts
- Framework accepts additional (external) sensors and analysis

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

How to Address the Problem of Poorly-Defined Requirements in Software System Design

Measurements and Decisions

Carnegie Mellon University Software Engineering Institute

[Distribution Statement A] Approved for public release and unlimited distribution.

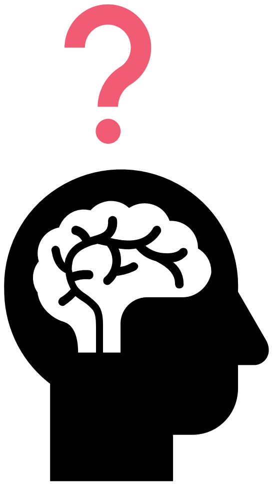
Decision-making with a Weighted Decision Matrix (WDM)

WDM is a well-established technique for supporting decision analysis, using

- criteria (e.g., requirements)
- per-criteria weights (* our metrics)
- total weights, per option

We use a WDM to compare options

- tools
- what to improve next
- what subfactors to focus on improving



Options

- Expert-identified possible match for stakeholder needs
  - functional
  - non-functional
- Involve tradeoffs
  - do better on some quality attribute subfactors
  - worse on others

WDM-assisted comparisons prioritize what matters to stakeholders

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Example Utility Tree with Priorities

Carnegie Mellon University Software Engineering Institute

Seed scenarios for this example:

- Critical Infrastructure Software Development
- Mission Critical Software Development

Quality Attribute	Sub-factor	Priority	Test	Test Quality Threshold Requirement
Security	Adjudicate alerts from SCA, SAST, and secrets-scanning tools	4	Percent adjudicated alerts	All critical and high alerts have been adjudicated
Security	Fix critical and high alerts (repair code or adjudicate false)	3	Have all critical and high alerts been fixed or adjudicated false?	All critical and high alerts have been fixed or adjudicated false
Cost	1-year costs	2	1-year combined costs	
Usability	Code variability	1	Flexibility of code (e.g., compile-time flag counts, variation points)	Average to low code flexibility metrics

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Weighted Decision Matrix

Quality Attribute	Sub-factor	Test	Test Results: Method 1: LLM-assisted alert adjudication +automated code repair	Test Results: Method 2: Fully-manual alert adjudication and code fixes	How to Scale Test Result (R)	Scaled Test Result or "FAIL": Method 1	Scaled Test Result or "FAIL": Method 2	Weight	Weighted Results: Method 1	Weighted Results: Method 2
Security	Adjudicate alerts from SCA, SAST, and secrets-scanning tools	Percent adjudicated alerts	0.90	0.05	R	0.90	0.05	0.40	0.36	0.02
Security	Fix critical and high alerts (adjudicate false and/or repair code)	Have all critical and high alerts been fixed or adjudicated false? (1 true, 0 false)	1	1	R	1.00	1.00	0.30	0.30	0.30
Cost	1-year costs	1-year combined costs (scale 0 low - 1 high)	0.10	0.90	1/R	10.00	1.11	0.20	2.00	0.22
Usability	Code variability	Flexibility of code (e.g., compile-time flag counts, variation points) (scale 0-1)	0.25	0.20	1/R	4.00	5.00	0.10	0.40	0.50
								Sum	$\boxed{3.06}$	1.04

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Weighted Decision Matrix

Carnegie Mellon University Software Engineering Institute

Quality Attribute	Sub-factor	Test	Test Results: Method 1: LLM-assisted alert adjudication +automated code repair	Test Results: Method 2: Fully-manual alert adjudication and code fixes	How to Scale Test Result (R)	Scaled Test Result or "FAIL": Method 1	Scaled Test Result or "FAIL": Method 2	Weight	Weighted Results: Method 1	Weighted Results: Method 2
Security	Adjudicate alerts from SCA, SAST, and secrets-scanning tools	Percent adjudic		.05	R	0.90	0.05	0.40	0.36	0.02
Security	Fix critical and high alerts (adjudicate false and/or repair code)	Have all c alerts bee adjudicate 0 false)			R	1.00	1.00	0.30	0.30	0.30
Cost	1-year costs	1-year co (scale 0 low		0	1/R	10.00	1.11	0.20	2.00	0.22
Usability	Code variability	Flexibility of coc compile-time fla variation points) (		0.20	1/R	4.00	5.00	0.10	0.40	0.50
									3.06	1.04

Sum

Higher is Better!

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Scaling

We scale in 2 ways

1. Higher values mean “better”

- Cost is inverted to `1/R` (since a higher cost is worse)
- Security is not inverted (higher security is better, so it is `R`)

2. Make the different metrics closer in value, by multiplying or dividing. For example:

- Bandwidth (bits/second) may be an enormous number
- Fraction of alerts adjudicated is between 0 and 1

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Weighted Decision Matrix

Quality Attribute	Sub-factor	Test	Test Results: Method 1: LLM-assisted alert adjudication +automated code repair	Test Results: Method 2: Fully-manual alert adjudication and code fixes	How to Scale Test Result (R)	Scaled Test Result or "FAIL": Method 1	Scaled Test Result or "FAIL": Method 2	Weight	Weighted Results: Method 1	Weighted Results: Method 2
Security	Adjudicate alerts from SCA, SAST, and secrets-scanning tools	Percent adjudicated alerts	0.90	0.05	R	0.90	0.05	0.40	0.36	0.02
Security	Fix critical and high alerts (adjudicate false and/or repair code)	Have all critical and high alerts been fixed or adjudicated false? (1 true, 0 false)	1	1	R	1.00	1.00	0.30	0.30	0.30
Cost	1-year costs	1-year combined costs (scale 0 low - 1 high)	0.10	0.90	1/R	10.00	1.11	0.20	2.00	0.22
Usability	Code variability	Flexibility of code (e.g., compile-time flag counts, variation points) (scale 0-1)	0.25	0.20	1/R	4.00	5.00	0.10	0.40	0.50
								Sum	3.06	1.04

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Weighted Decision Matrix

Quality Attribute	Sub-factor	Test	Test Results: Method 1: LLM-assisted alert adjudication +automated code repair	Test Results: Method 2: Fully-manual alert adjudication and code fixes	How to Scale Test Result (R)	Scaled Test Result or "FAIL": Method 1	Scaled Test Result or "FAIL": Method 2	Weight	Weighted Results: Method 1	Weighted Results: Method 2
Security	Adjudicate alerts from SCA, SAST, and secrets-scanning tools	Percent adjudicated alerts	0.90	0.05	R	0.90	0.05	0.40	0.36	0.02
Security	Fix critical and high alerts (adjudicate false and/or repair code)	Have all critical and high alerts been fixed or adjudicated false? (1 true, 0 false)	1	1	R	1.00	1.00	0.30	0.30	0.30
Cost	1-year costs	1-year combined costs (scale 0 low - 1 high)	0.10	0.90	1/R	10.00	1.11	0.20	2.00	0.22
Usability	Code variability	Flexibility of code (e.g., compile-time flag counts, variation points) (scale 0-1)	0.25	0.20	1/R	4.00	5.00	0.10	0.40	0.50
									3.06	**1.04**

Sum

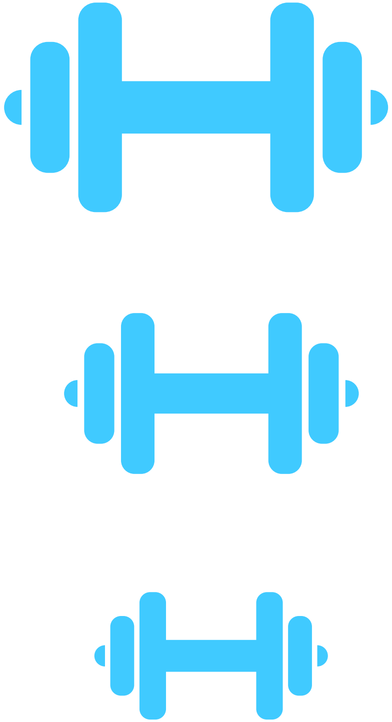
Carnegie Mellon University Software Engineering Institute

Using Priorities to Create Weights: Direct or Formulas

Direct weighting is manual

Formulas weight differently

Select a formula that fits stakeholder prioritization analyses



Consider:

- Are differences in priority importance evenly distributed?
- Do any adjacent priority items exhibit disproportionate differences in importance?
- Should priority tiers (high, medium, low) be differentially weighted?

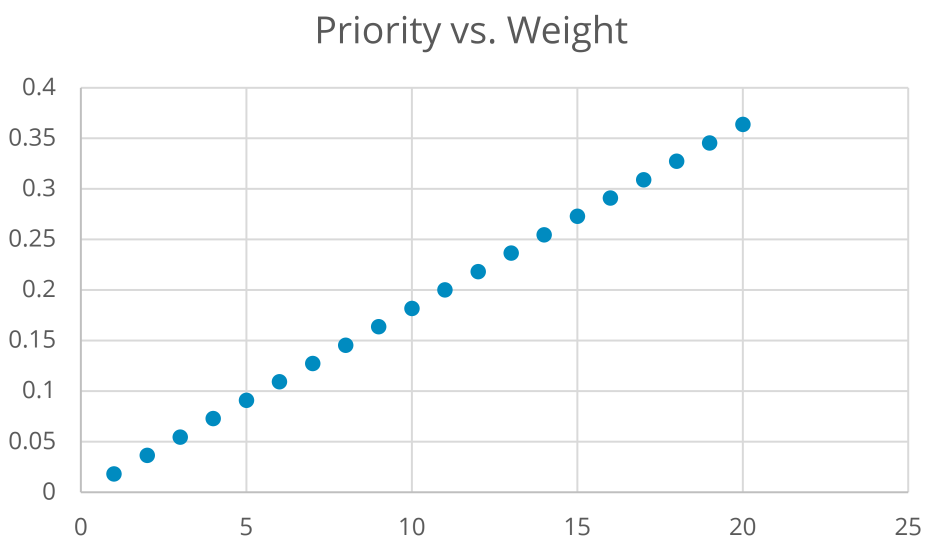
© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Using Priorities to Create Weights with Formula "Rank-to-Linear Weight"

- Weights are larger for higher priorities
- *Differences* between consecutive weights are relatively constant
- Total sum of weights is 1.0
- `N` is the count of items ranked
- `r` is the priority ("rank")

Formula: `2r/N(N+1)`



Note: Standard formula uses smaller rank for higher priorities

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Weighted Decision Matrix

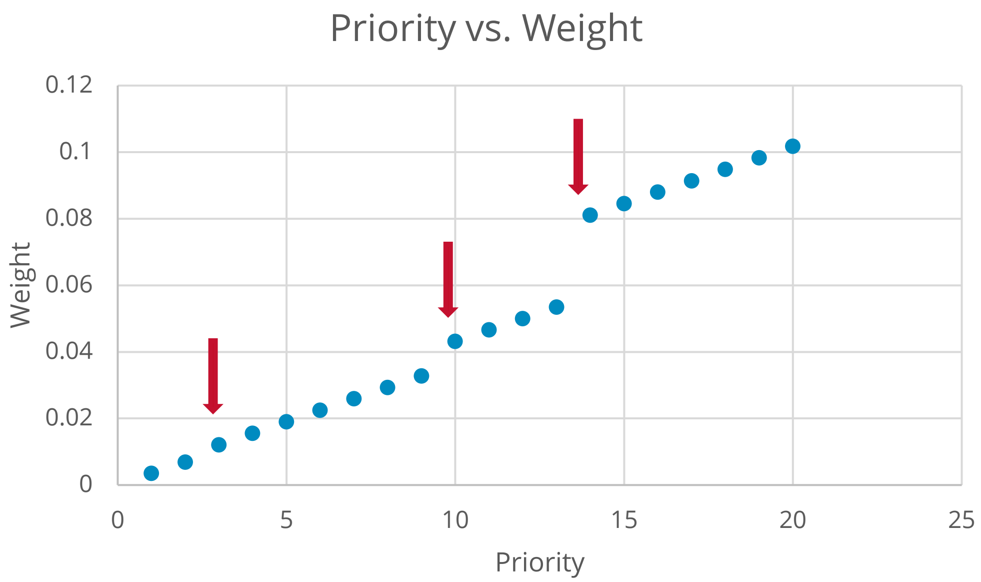
Quality Attribute	Sub-factor	Test	Test Results: Method 1: LLM-assisted alert adjudication +automated code repair	Test Results: Method 2: Fully-manual alert adjudication and code fixes	How to Scale Test Result (R)	Scaled Test Result or "FAIL": Method 1	Scaled Test Result or "FAIL": Method 2	Weight	Weighted Results: Method 1	Weighted Results: Method 2
Security	Adjudicate alerts from SCA, SAST, and secrets-scanning tools	Percent adjudicated alerts	0.90	0.05	R	0.90	0.05	0.40	0.36	0.02
Security	Fix critical and high alerts (adjudicate false and/or repair code)	Have all critical and high alerts been fixed or adjudicated false? (1 true, 0 false)	1	1	R	1.00	1.00	0.30	0.30	0.30
Cost	1-year costs	1-year combined costs (scale 0 low - 1 high)	0.10	0.90	1/R	10.00	1.11	0.20	2.00	0.22
Usability	Code variability	Flexibility of code (e.g., compile-time flag counts, variation points) (scale 0-1)	0.25	0.20	1/R	4.00	5.00	0.10	0.40	0.50
								Sum	3.06	1.04

Using Priorities to Create Weights with Formula “Linear with Local Step Multipliers”

- Weight w_r increases if priority increases
- Weights include sharper jumps at user-specified priorities
- *Differences* between consecutive weights are mostly constant and larger only at specific points
- Total sum of weights is 1.0

Formula: $w_r = \frac{\sum_{k=1}^{r} s_k}{\sum_{k=1}^{N} \sum_{j=1}^{k} s_j}$, where $s_k = 1$ except for the jump factors.

(This example, $s_3 = 1.5$, $s_{10} = 3$, $s_{14} = 8, \mathrm{N} = 17$)



© 2025 Carnegie Mellon University

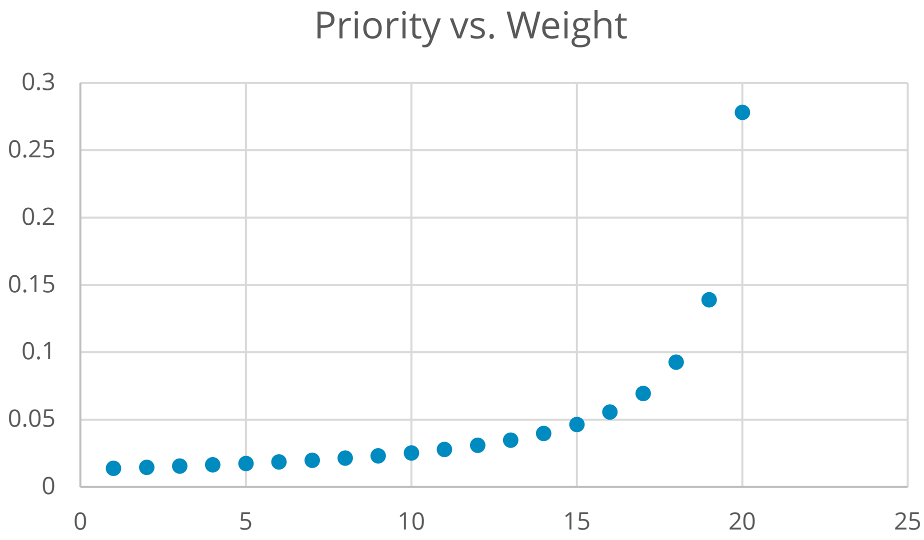
[Distribution Statement A] Approved for public release and unlimited distribution.

Using Priorities to Create Weights with Formula "Rank Reciprocal Method"

- Weight increases if priority increases
- Steeper drop-off for medium and lower-priority weights
- Total sum of weights is 1.0

Formula:

$$w_r = \frac{1/(N+1-r)}{\sum_{k=1}^{N} 1/(N+1-k)}$$



Note: Standard formula uses smaller rank for higher priorities

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

How Does Silent Sentinel Help?

Carnegie Mellon University Software Engineering Institute

Supplements analysis provided by SAST, DAST, and ISAT

- Passively collects and analyzes program and system behavior

Supports flexible and extensible environments

- Configure for special environmental scenarios
- Analyze single program or multistep process
- Add new sensors within framework
- Retains all intermediate data for additional analysis

Integrates into DevSecOps and CI/CD environments

- Built with containers to allow easy orchestration

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

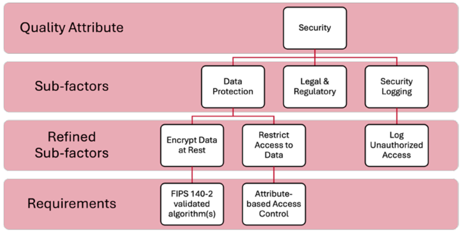
How to address the problem of poorly-defined requirements in software system design?

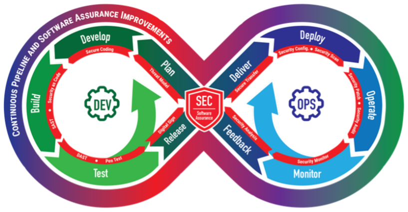
Carnegie Mellon University Software Engineering Institute

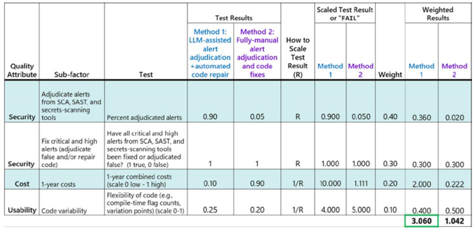
We discussed strategies to help define good requirements:

- Analyze Requirement Tradeoffs
- Lean Into Modern Software Engineering Practices
- Use Math to Validate and Support Your Decisions

…and some tools and techniques you can use with them.







Quality Attribute	Sub-factor	Test	Test Results: Method 1: LLM-assisted alert adjudication +automated code repair	Test Results: Method 2: Fully-manual alert adjudication and code fixes	How to Scale Test Result (R)	Scaled Test Result or "FAIL": Method 1	Scaled Test Result or "FAIL": Method 2	Weight	Weighted Results: Method 1	Weighted Results: Method 2
Security	Adjudicate alerts from SCA, SAST, and secrets-scanning tools	Percent adjudicated alerts	0.90	0.05	R	0.900	0.050	0.40	0.360	0.020
Security	Fix critical and high alerts (adjudicate false and/or repair code)	Have all critical and high alerts from SCA, SAST, and secrets-scanning tools been fixed or adjudicated false? (1 true, 0 false)	1	1	R	1.000	1.000	0.30	0.300	0.300
Cost	1-year costs	1-year combined costs (scale 0 low - 1 high)	0.10	0.90	1/R	10.000	1.111	0.20	2.000	0.222
Usability	Code variability	Flexibility of code (e.g., compile-time flag counts, variation points) (scale 0-1)	0.25	0.20	1/R	4.000	5.000	0.10	0.400	0.500
									3.060	**1.042**

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Engage with us

Resources

- Silent Sentinel on GitHub
- "Silent Sentinel Tool Automates Software Risk Analysis", Feb. 10, 2025.
- "Do You Know What Your Software Is Actually Doing?" fact sheet
- L. Hughes, M. Sherman. "Dynamic Analysis of an Executable's System Impact", InfoSec World 2025
- J. Sible. "Do You Really Know What Your Application is Doing?" ISC2 Security Congress 2025
- ATAM Collection
- Continuous Deployment of Capability

We're interested in improving software quality:

- Creating software that's secure by design
- Improving software engineering processes

Contact: info@sei.cmu.edu

Backup

Carnegie Mellon University Software Engineering Institute

[Distribution Statement A] Approved for public release and unlimited distribution.

Some Gaps Addressed by Silent Sentinel

Silent Sentinel can help answer these types of questions:

- What data should I look at to assess the risk of deploying this program?
- What is the risk of deploying this capability in my environment?
- Does this program behave the way the developers say it does?
- Are there behaviors that were not considered in the specifications?
- Might this program interfere with another program?
- Are there behaviors that should not be present?
- Is the entire DevSecOps process secure from build to operate?

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

How It Works

Silent Sentinel works by following this process:

© 2025 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.