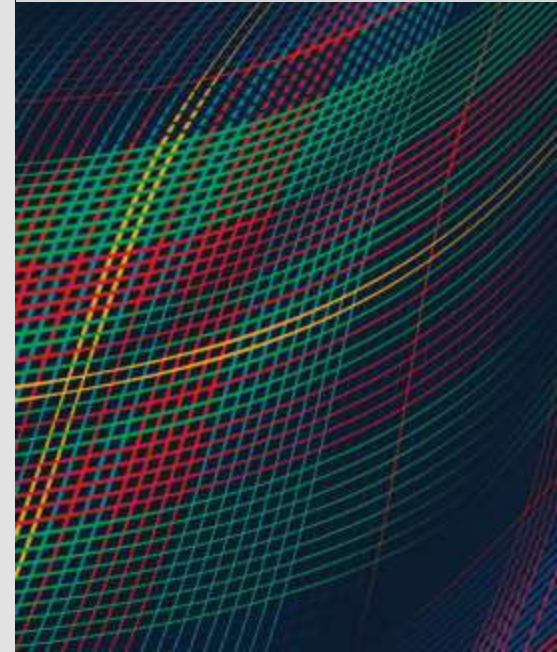


A Disciplined Approach to Making and Analyzing Architectural Decisions

SEPTEMBER 18, 2023

Rick Kazman, Phil Bianco, Sebastián Echeverría, James Ivers, John Klein



Copyright 2023 Carnegie Mellon University.

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.

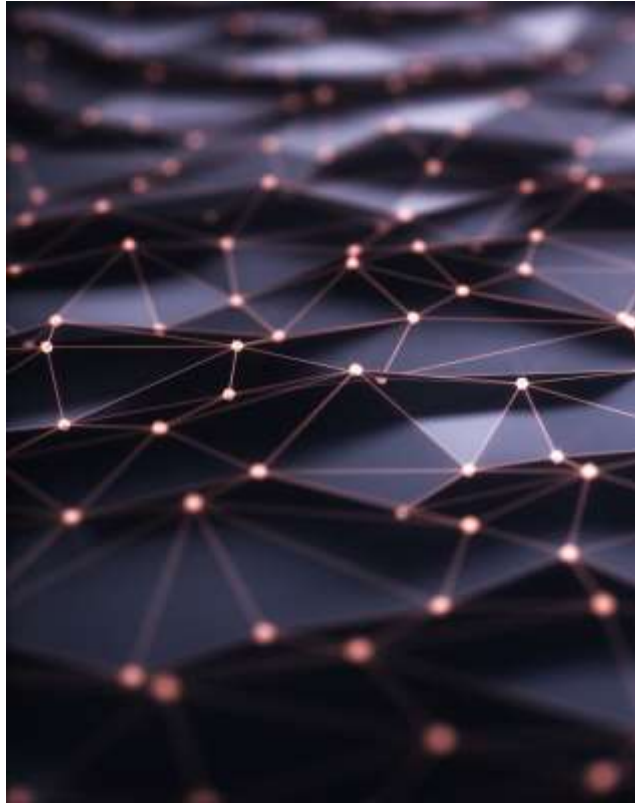
NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, 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.

[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.

DM23-0942

Architecture as an Enabler



Architecture is key to realizing driving quality attribute a (maintainability, robustness, extensibility, integrability, etc.) that directly impact long-term cost and schedule goals.

Architecture helps organizations proactively manage changes in requirements and technology, providing value throughout a system's lifetime.

As such, architectures need to be updated and used for new analyses over time to help keep systems healthy.

Architecture Tradeoffs



When creating and evolving an architecture, tradeoff decisions are inevitable.

- A decision made to satisfy one requirement may undermine another.

However, the impact of decisions across multiple requirements are not always explicitly assessed and are sometimes difficult to assess at all.

As such, architectural decisions may be inappropriate when first made or may be undermined by other decisions as a system evolves, scales, and is maintained.

Tradeoff Examples



Context is vital in architecture.

- implementation time vs. maintainability
- cost and complexity vs. throughput and robustness
- extensibility vs. latency
- maintainability vs. robustness
- portability vs. performance
- implementation time vs. run-time performance
- latency vs. space
- testability vs. latency

The Problem



How can we gain more confidence that architectural decisions are appropriate?

Specifically, how do we

- Elicit architectural requirements?
- Make architectural decisions?
- Analyze the decisions made?

The Solution

We offer a **disciplined** approach to making and analyzing architectural decisions, available as reports on **architecting to predictably achieve critical quality attributes**.



Report Structure

Each report provides

- a set of **definitions, core concepts**, and a **framework for reasoning** about quality attributes and their satisfaction by an architecture and a system
- a **template** for eliciting quality attribute requirements as scenarios
- a set of mechanisms—**patterns and tactics**—that are commonly used to satisfy quality attribute requirements
- a step-by-step method—a **playbook**—that analysts can use to determine whether the architectural decisions contain serious risks relative to the quality attribute requirements

Extensibility Scenario Template

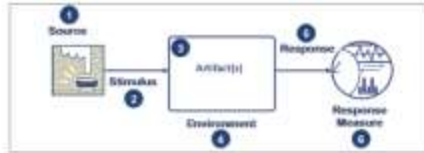


Figure 2: Six Parts of a Scenario

The degree to which an analyst can accurately analyze an architecture is directly correlated to the degree to which the scenarios are accurately specified. Below we provide a general scenario for integrity and then some example integrity scenarios derived from the general scenario. A general scenario is a system-independent scenario that allows stakeholders to communicate more effectively about quality attribute requirements and use assist stakeholders in developing concrete scenarios.

4.1 General Scenario for Integrity

There is no single scenario that specifies all of the possible measurements that could characterize a property like integrity. But we do use some common themes. A general scenario maps those common themes into the parts of a quality attribute scenario, providing a template that we can use to create concrete scenarios for a particular system. The general scenario defines the type of the values for each part of the scenario, and a concrete scenario for the integrity of a system is created by specifying one or more system-specific values of the selected type for each part of the scenario. (We say "values" because, for example, a scenario might have more than one response measure.)

Here is the general scenario for integrity:

| Scenario Part | Possible Type for Each Value |
|---------------|---|
| Source | One or more of the following: <ul style="list-style-type: none"> • managementsystem administrator • component manufacturer • component vendor |
| Stimulus | One of the following: <ul style="list-style-type: none"> • add new component • integrate new version of existing component • integrate existing components together in a new way |
| Artifact | One of the following: <ul style="list-style-type: none"> • entire system • specific set of components • component interface • component configuration |

UNIVERSITY OF PITTSBURGH | SOFTWARE ENGINEERING INSTITUTE | CARNegie MELLON UNIVERSITY
Distribution Statement (S) Approved for public release and unlimited distribution.

| Scenario Part | Possible Type for Each Value |
|------------------|---|
| Response | One or more of the following: <ul style="list-style-type: none"> • addresses are (partially, integrated, based) • new components are successfully (or not) communicating with the core system • addresses in the new configuration do not create any response errors • addresses in the new configuration do not impact the correct behavior of other addresses |
| Response Measure | One or more of the following: <ul style="list-style-type: none"> • if components changed • if scale • if lines of code changed • effort • money • calendar time • effects on quality attribute response measures • if new defects introduced |

4.2 Example Scenarios for Extensibility

Each of the following example scenarios is constructed by selecting one or more of the types of values from each of the six parts of the general scenario and specifying a system-specific value. For each example, we will use an easy-to-understand "logical" system. In practice, you would choose values that are as precise as possible in the context of your system.

4.2.1 Scenario 1: New Track Information

This example scenario describes how an external developer can use the provided extension points to add a new type of moving track information to the system.

| Scenario Part | Value |
|------------------|---|
| Source | External party |
| Stimulus | Add new type of moving track information to the system |
| Artifact | Fusion algorithm extension point |
| Environment | Development shop |
| Response | Increase resolution of fused tracks Code-based modification made using existing extension points |
| Response Measure | No part of the core system needs to be recompiled Development can be done by an external party in less than 8 person-weeks |

CARNEGIE MELLON UNIVERSITY | SOFTWARE ENGINEERING INSTITUTE | CARNegie MELLON UNIVERSITY
Distribution Statement (S) Approved for public release and unlimited distribution.

EXTENSIBILITY SCENARIO TEMPLATE
Architecture requirements can be captured as structured quality attribute scenarios.

Maintainability Tactics

5.1 Tactics

Tactics are the building blocks of design, the raw materials from which patterns are constructed. Each set of tactics is grouped according to the quality attribute goal that it addresses. The goals for the maintainability tactics shown in Figure 2 are to reduce the costs and risks of adding new components, modifying existing components, testing, and integrating sets of components to fulfill evolutionary requirements. As discussed in Section 3.1, the maintainability tactics achieve this by reducing the amount of coupling between components, reducing the distance between components, and making the testing and deployment of components easier, less costly, and more disciplined.

These tactics are known to influence the response (and hence the cost) to the general maintainability requirements (e.g., number of components changed, percent of code changed, effort, calendar time). The tactic descriptions presented below are derived, in part, from the third edition of *Software Architecture in Practice* (Bass 2012).



Figure 2: Maintainability Tactics

We discuss each of the tactics presented in Figure 2 in more detail below. For each tactic that we discuss, we describe the tactic and relate it to the taxonomy defined in Section 3.1 as a way of characterizing the scope and impact of the tactic. The tactic descriptions are inspired by and derived, in part, from the third edition of *Software Architecture in Practice* (Bass 2012). Table 2 summarizes the tactics presented in this section and how each relates to the measures presented in Sections 3.1 and 3.2.

Table 2: Maintainability Tactics and Their Relationship to Measures of Interest

| Tactic | Coupling | Distance | Means of Control of State | Test Efficiency | Generality | Scalability | Efficiency |
|--------------------------|----------|----------|---------------------------|-----------------|------------|-------------|------------|
| Encapsulate | + | + | + | + | | | |
| Use an intermediary | + | | + | + | | | |
| Reduce dependencies | + | | | | | | |
| Abstract common services | + | | | | | | |

SOFTWARE ENGINEERING INSTITUTE | CARNEGIE MELLON UNIVERSITY
[Distribution Statement A] Approved for public release and unlimited distribution.

| Tactic | Coupling | Distance | Means of Control of State | Test Efficiency | Generality | Scalability | Efficiency |
|--------------------------|----------|----------|---------------------------|-----------------|------------|-------------|------------|
| Split module | + | + | + | + | | | |
| Refactor | | | | | | | |
| Reduce semantic cohesion | | | | | | | |
| Data binding | + | | | | | | |
| Specialize interfaces | | | | | | | |
| Localize state storage | | | | | | | |
| Abstract data sources | | | | | | | |
| Serialize | | | | | | | |
| Contextualize services | | | | | | | |
| Segment requirements | | | | | | | |
| Modularize | | | | | | | |
| Package together | | | | | | | |
| Componentize | | | | | | | |

Note: A plus sign indicates that the tactic positively addresses maintainability properties or metrics measures, a minus sign indicates that the tactic has a negative effect, and an asterisk indicates that the tactic might positively or negatively address the measure, depending on its realization. A blank cell means that the property has no discernible effect on the measure.

It is worth discussing some of the semantic entries in Table 2. For example, the encapsulate tactic may have positive or negative effects on the observation of state and control of state. Encapsulation may positively affect these properties if the state variables are appropriately encapsulated, but it may negatively affect these properties if the encapsulation hides the state variables. Similarly, refactoring may have a positive or negative effect on the observation of state, control of state, and test efficiency, depending on whether those were goals of the refactoring.

In addition, in Table 3, we show the anticipated effects of maintainability tactics on the various items of coupling introduced in the work of Karszen and colleagues (Karszen 2013). These items of coupling are syntactic, data semantic, behavioral semantic, temporal distance, and resource distance.

Table 3: Selected Maintainability Tactics and Their Impacts on Various Aspects of Coupling

| Tactic | Syntactic Distance | Data Semantic Distance | Behavioral Semantic Distance | Temporal Distance | Resource Distance |
|--------------------------|--------------------|------------------------|------------------------------|-------------------|-------------------|
| Encapsulate | + | + | + | | |
| Use an intermediary | | | | | |
| Reduce dependencies | | | | | |
| Abstract common services | + | + | + | | |
| Split module | | | | | |

SOFTWARE ENGINEERING INSTITUTE | CARNEGIE MELLON UNIVERSITY
[Distribution Statement A] Approved for public release and unlimited distribution.

MAINTAINABILITY TACTICS
Tactics are building blocks of design decisions for achieving quality attribute requirements.

Pages 22 & 23 of the Maintainability Report

Characteristics of Extensibility



Figure 2: Extensibility Tactics

These tactics are known to influence the response (and hence the cost) to the general scenario for extensibility (e.g., number of components affected, effort, calendar time, new skills introduced). The tactic descriptions presented below are derived from *Software Architecture in Practice* [Bass 2012] and “Forward Design Decisions in Elastic Deployability” [Balkema 2014]. Table 1 summarizes the tactics presented in this section and how each relates to the characteristics and measures presented in Section 3.1.

Table 1: Extensibility Tactics and Their Relationships to Extensibility Characteristics

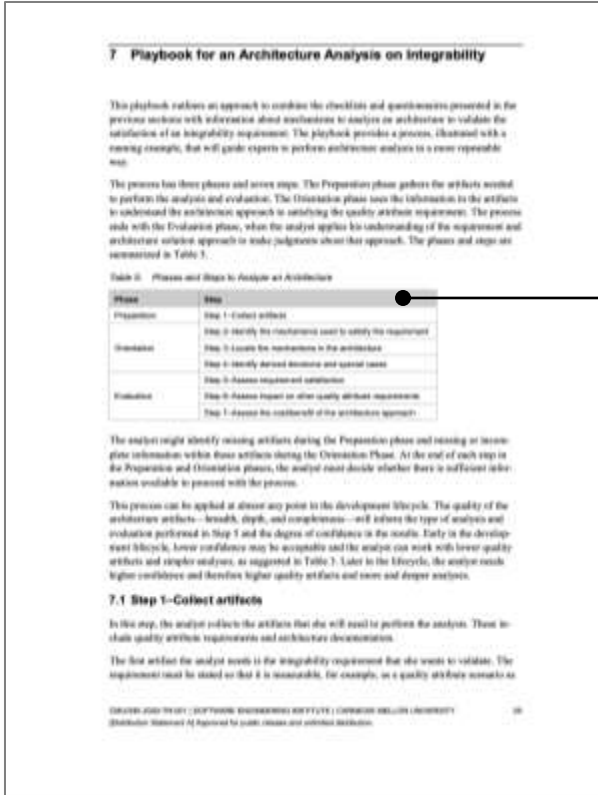
| Tactic | Extensibility Characteristics | | | | | | | | | |
|------------------------------|-------------------------------|-------------------|------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| | Locality Coupling | Highly Coupled | High Coupling | High Coupling Link | High Coupling Link | High Coupling Link | High Coupling Link | High Coupling Link | High Coupling Link | High Coupling Link |
| Organize | + | | | | | | | | | |
| Use an intermediary | | | | | | | | | | |
| Abstract communication paths | | | | | | | | | | |
| Abstract common services | | | | | | | | | | |
| Order loading | | | | | | | | | | |
| Decouple services (links) | | | | | | | | | | |
| Decouple services (domains) | | | | | | | | | | |
| Specialized interfaces | | | | | | | | | | |
| Reconfigurability | | | | | | | | | | |
| Local state storage | | | | | | | | | | |
| Checksums | | | | | | | | | | |
| Manage resources | | | | | | | | | | |
| Segment deployments | | | | | | | | | | |

Note: A plus sign indicates that the tactic positively addresses an extensibility characteristic and/or related measure, a minus sign indicates that the tactic has a negative effect, and an asterisk indicates that the tactic might have a daily or regularly address the measure, depending on its motivation. A blank cell means that the tactic has no apparent effect on the measure.

©2014-2022, PH.D. SOFTWARE ENGINEERING INSTITUTE, CARNEGIE MELLON UNIVERSITY
Distribution Statement B: Approved for public release and unlimited distribution.

QUALITY ATTRIBUTE MATRIX
Quality attributes are complex, and tactics influence distinct characteristics.

Integrability Playbook



PLAYBOOK
Each report includes a playbook that illustrates a step-by-step method for analyzing architectural decisions with an example.

For More information

This QR code will take you to the set of reports:

