QUality Assessment of System Architectures and their Requirements (QUASAR)

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Donald Firesmith
28 March 2007

Topics

What is QUASAR?

Why Use QUASAR?

QUASAR Philosophy

QUASAR Method

QUASAR Benefits

QUASAR – Today and Tomorrow

What is QUASAR?

What is QUASAR?

QUality Assessment of System Architectures and their Requirements

a Well-Documented and Proven Method based on the use of Quality Cases for Independently Assessing the Quality of:

- Software-intensive *System* / Subsystem *Architectures* and the
- *Architecturally-Significant Requirements* that Drive Them

QUASAR Version 1 (July 2006) emphasized the quality assessment of architectures against architecturally-significant requirements.

QUASAR Version 2 (February 2007) addresses the quality assessment of *both* architectures *and* their architecturally-significant requirements.

Understanding QUASAR's Definition

To understand QUASAR's definition, you must understand:

- Quality
- Quality Cases
- Software-Intensive Systems (as opposed to just Software)
- Architecture
- Architecturally-Significant Requirements

Quality Model:

Defining System Architecture Quality

What is Quality?

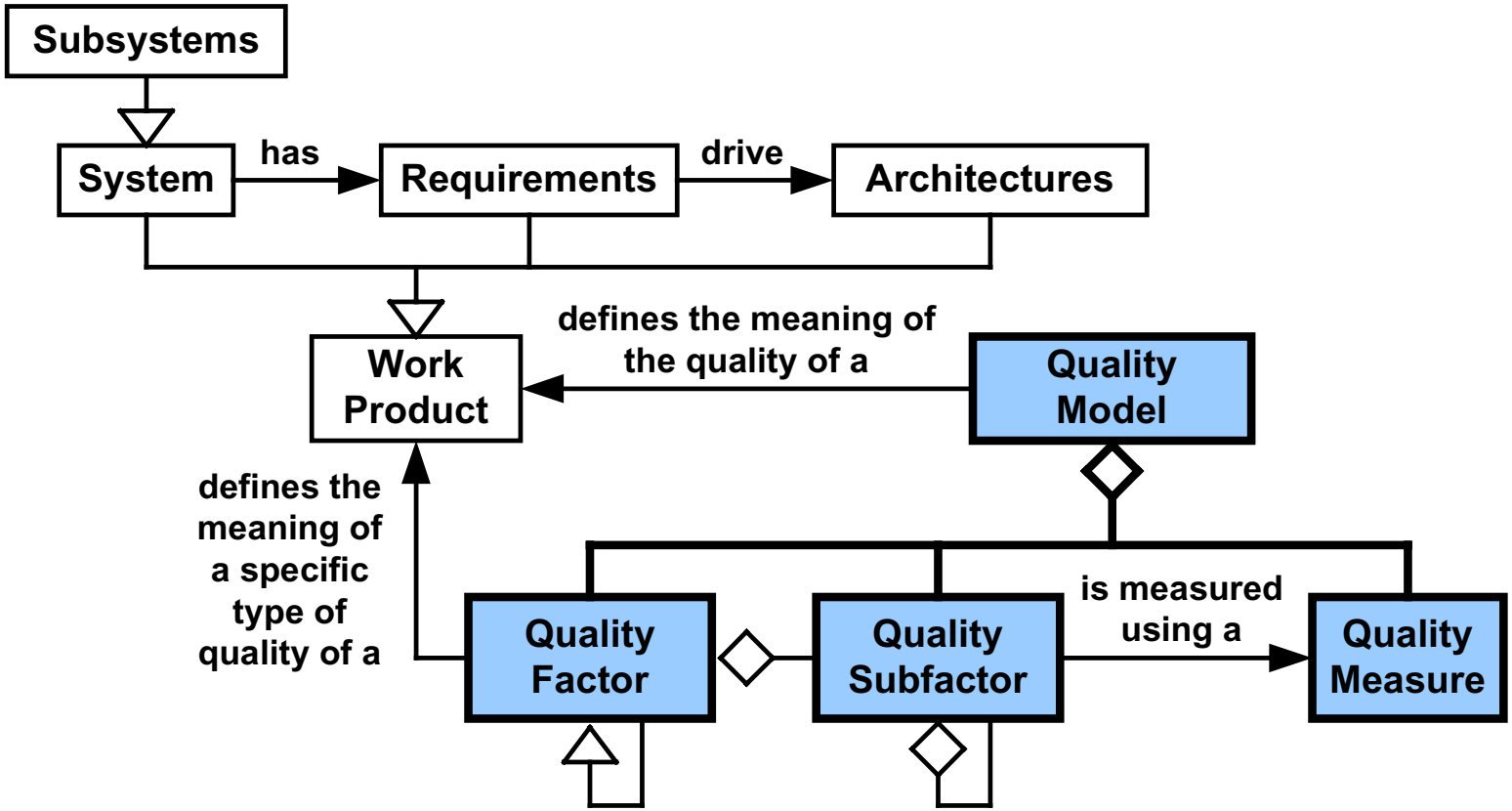
Quality

> the Degree to which a Work Product (e.g., System, Subsystem, Requirements, Architecture) Exhibits a Desired or Required Amount of Useful or Needed Characteristics

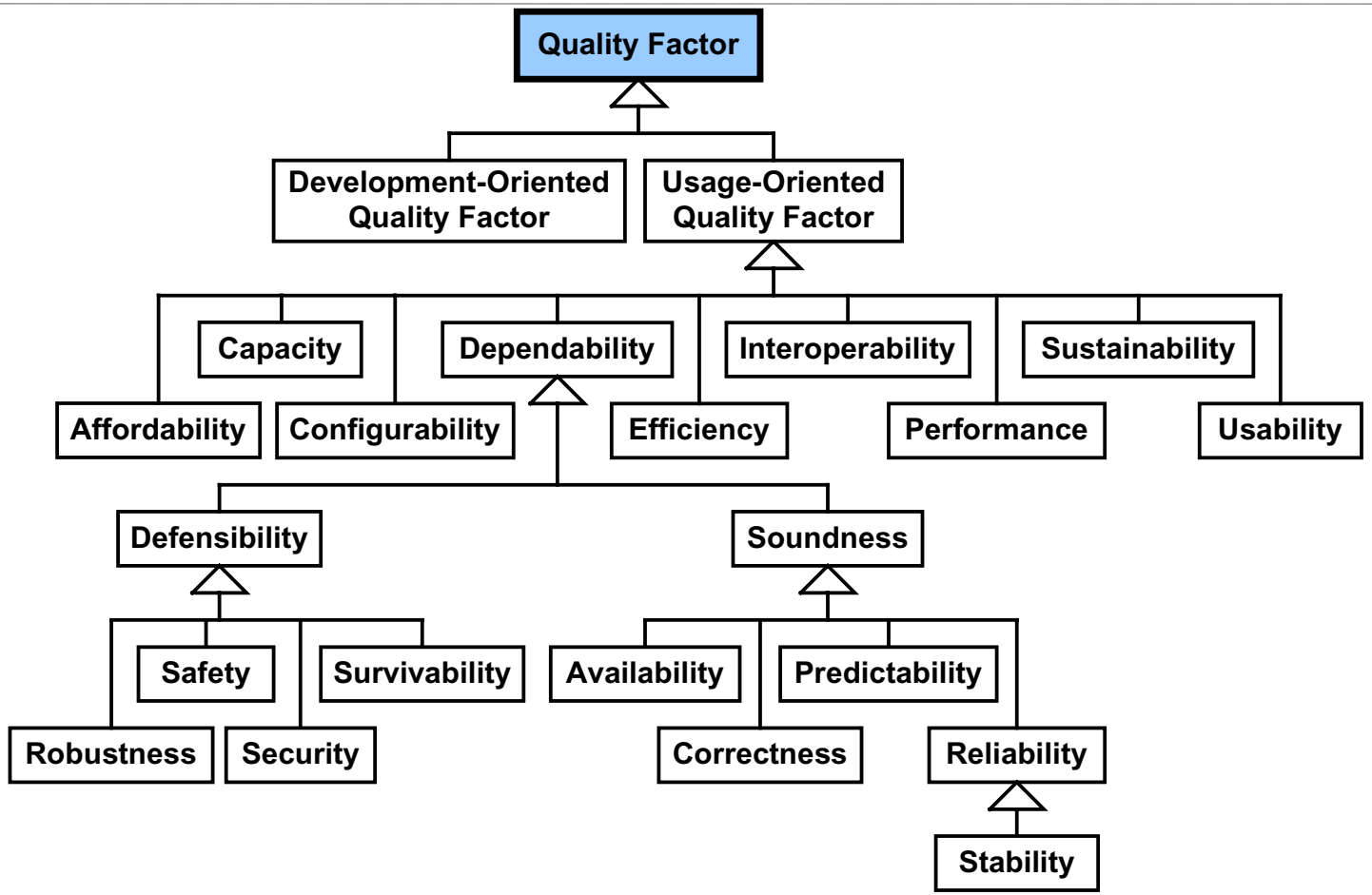
Quality of a Work Product is defined in terms of a Quality Model:

- **Quality Factors**
  (a.k.a., Quality Attributes, Quality Characteristics, 'ilities')
  (e.g., availability, interoperability, performance, reliability, etc.)
- **Quality Subfactors**
  (e.g., jitter, latency, response time, schedulability, throughput)
- **Quality Measures**
  (e.g., milliseconds, transactions per second)

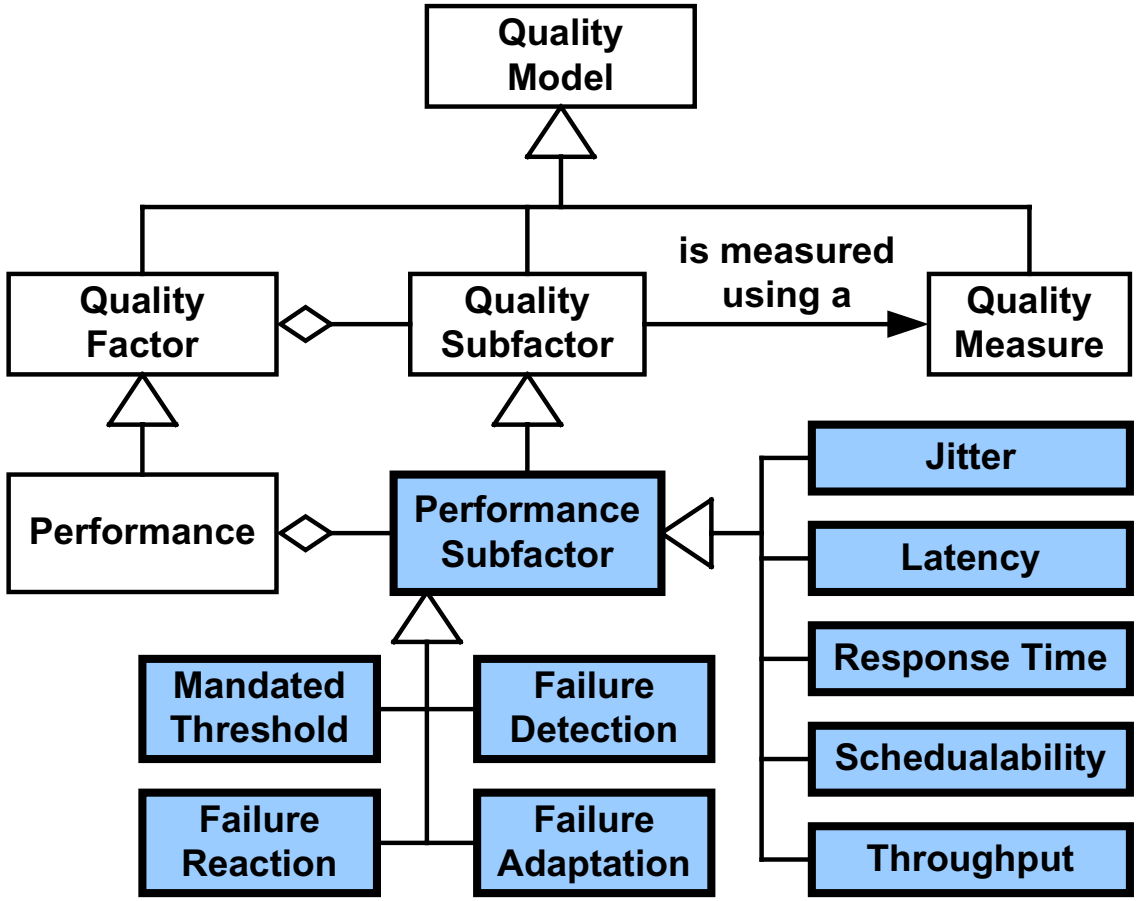
Quality Model



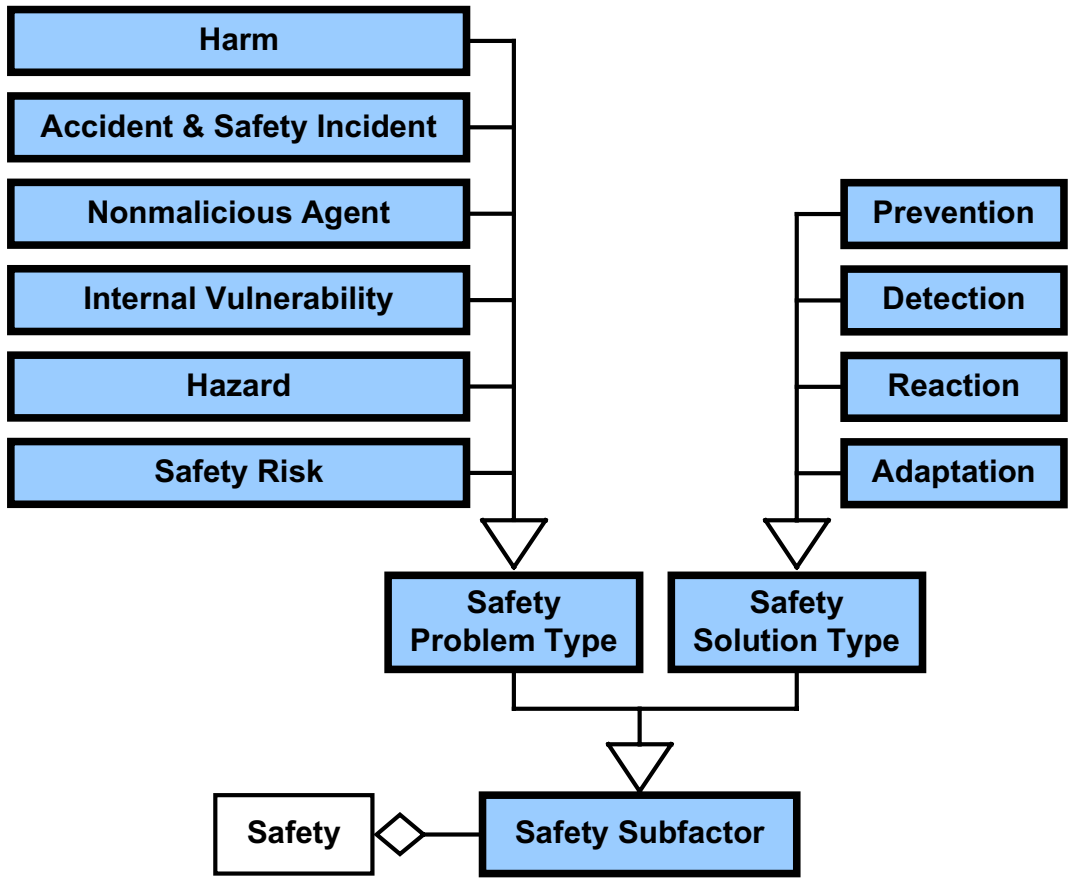
Quality Model – Quality Factors



Quality Model – Performance Quality Subfactors



Quality Model – Safety Quality Subfactors



Quality Case: Foundation of QUASAR

Quality Case - Definition

Quality Case

> a Cohesive Collection of *Claims*, *Arguments*, and *Evidence* that Makes the Developers' Case that their Work Product has **Sufficient Quality**

A Generalization of Safety Cases from the Safety Community:

- Can Address any Quality Factor and/or Quality Subfactor

Useful for:

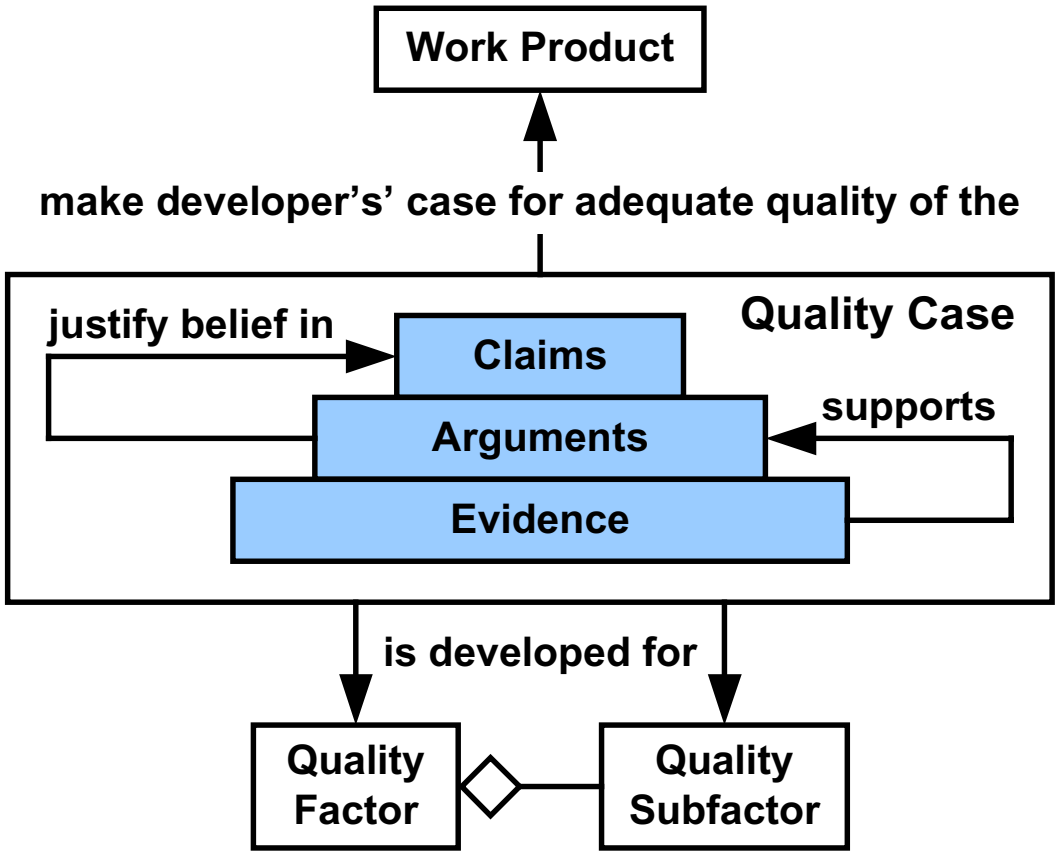
- Assessing Quality
- Certification and Accreditation

Quality Cases – Components$_1$

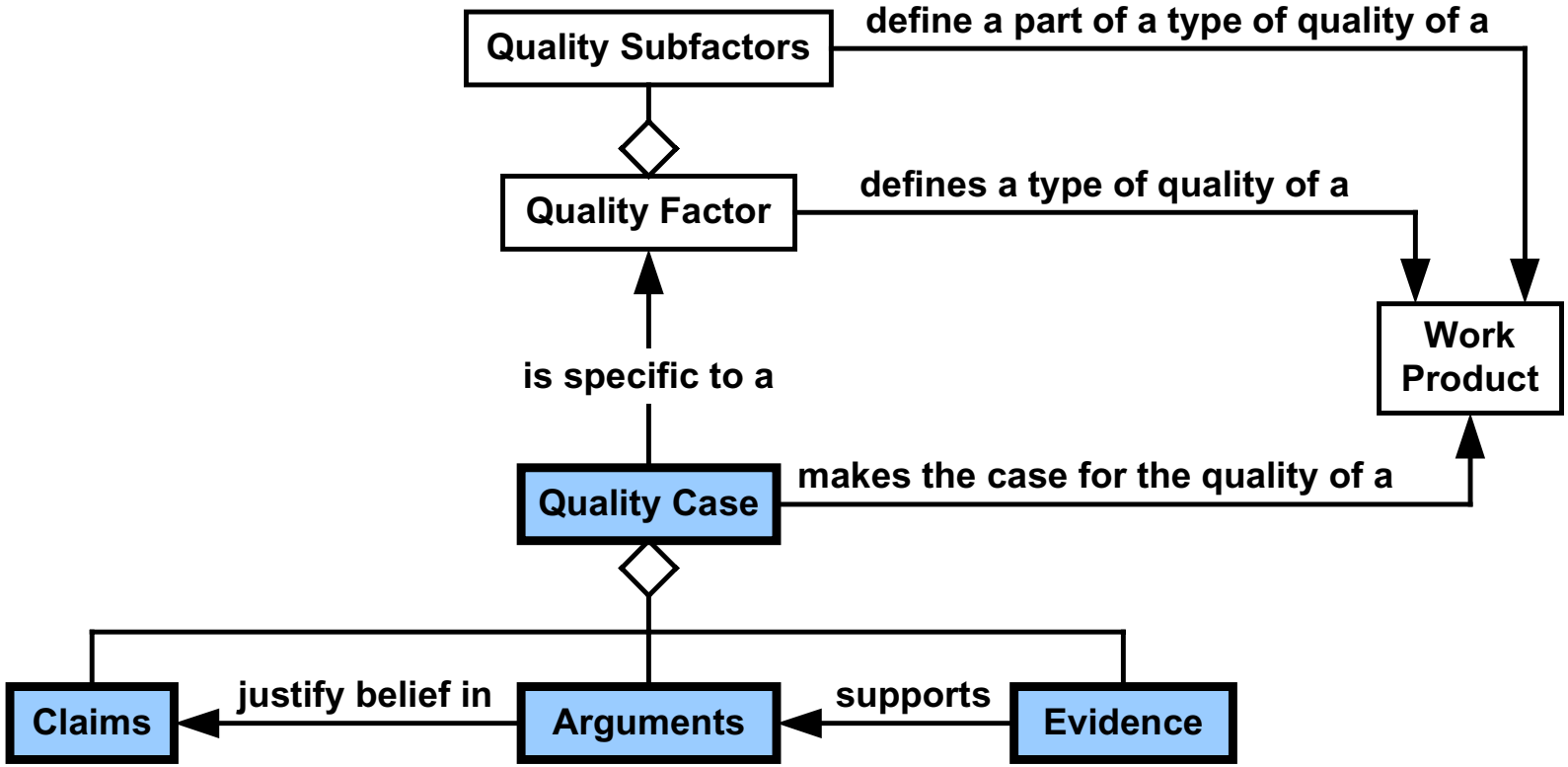
A Quality Case consists of the following types of Components:

- **Claims**
  Developers' Claims that their Work Products have *Sufficient* Quality, whereby quality is defined in terms of the qualify factors and quality subfactors defined in the official project quality model
- **Arguments**
  Clear, Compelling, and Relevant Developer Arguments Justifying the Assessors' Belief in the Developers' Claims
  (e.g., inventions, decisions, analysis and simulation results, trade-offs, associated rationales, and assumptions)
- **Evidence**
  Adequate Credible Evidence Supporting the Developers' Arguments
  (e.g., official project diagrams, models, requirements specifications and architecture documents; requirements repositories; analysis and simulation reports; test results; and demonstrations witnessed by the assessors)

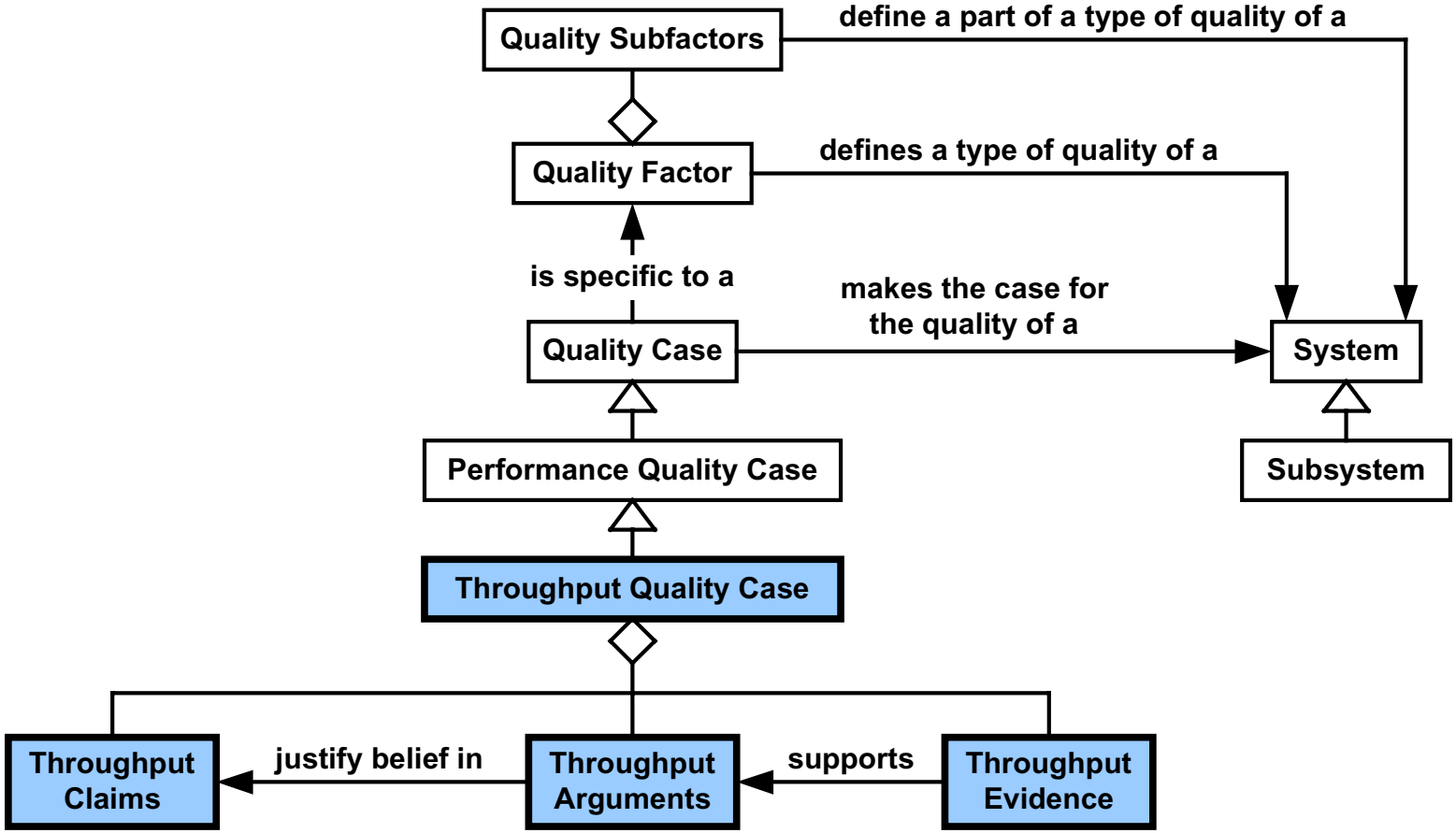
Quality Cases – Components$_2$



Quality Cases – Components$_3$



QUASAR Throughput Quality Case



Specialized QUASAR Quality Cases

QUASAR utilizes the following types of Quality Case:

- Requirements Quality Cases
- Architectural Quality Cases

QUASAR Version 1 only had Architectural Quality Cases.

QUASAR Version 2 has Both Types of Quality Cases.

QUASAR Requirements Quality Cases$_1$

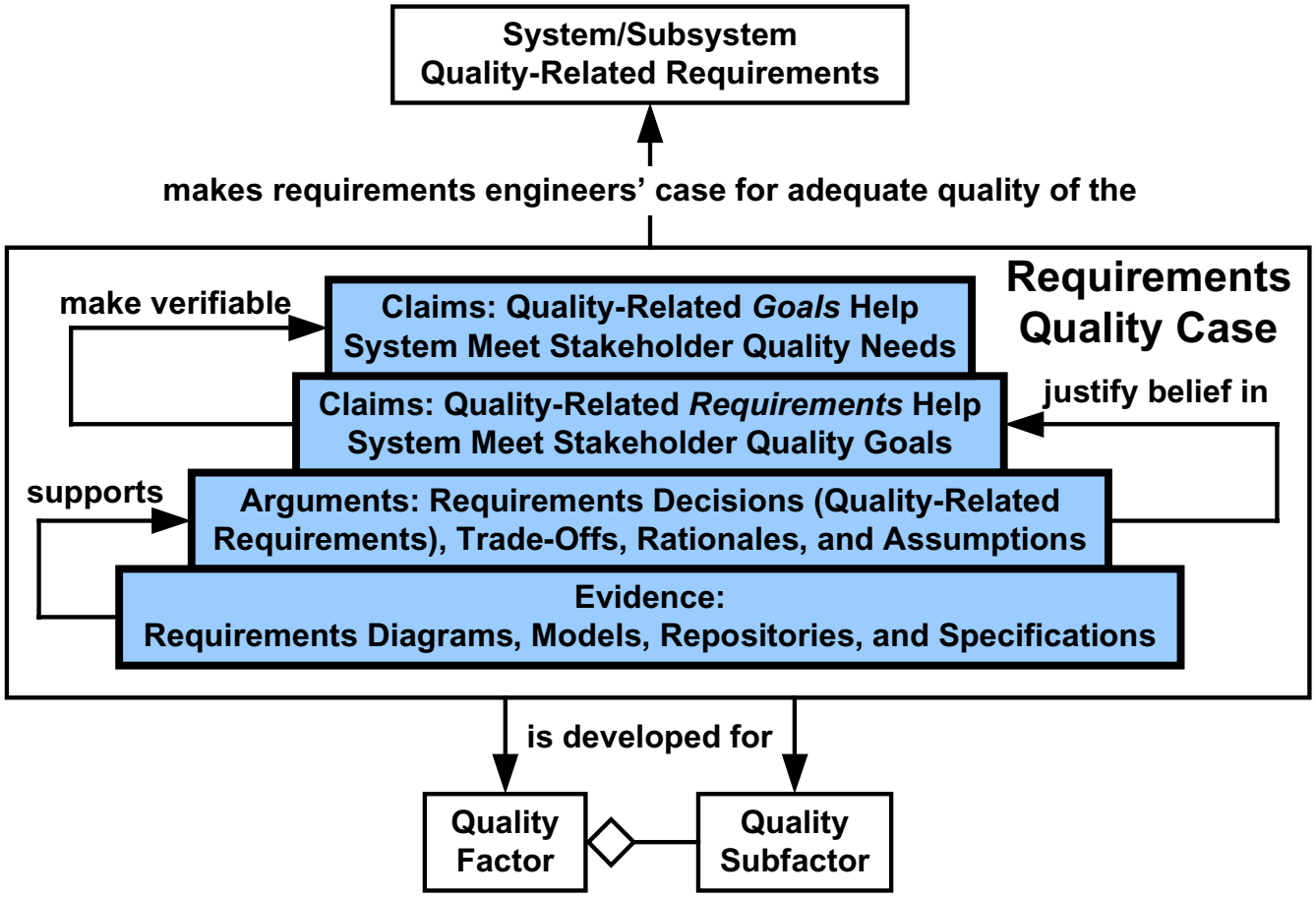
Requirements Quality Case

> a Specialized Quality Case that is Limited to the Quality of Architecturally-Significant, Quality-Related Requirements

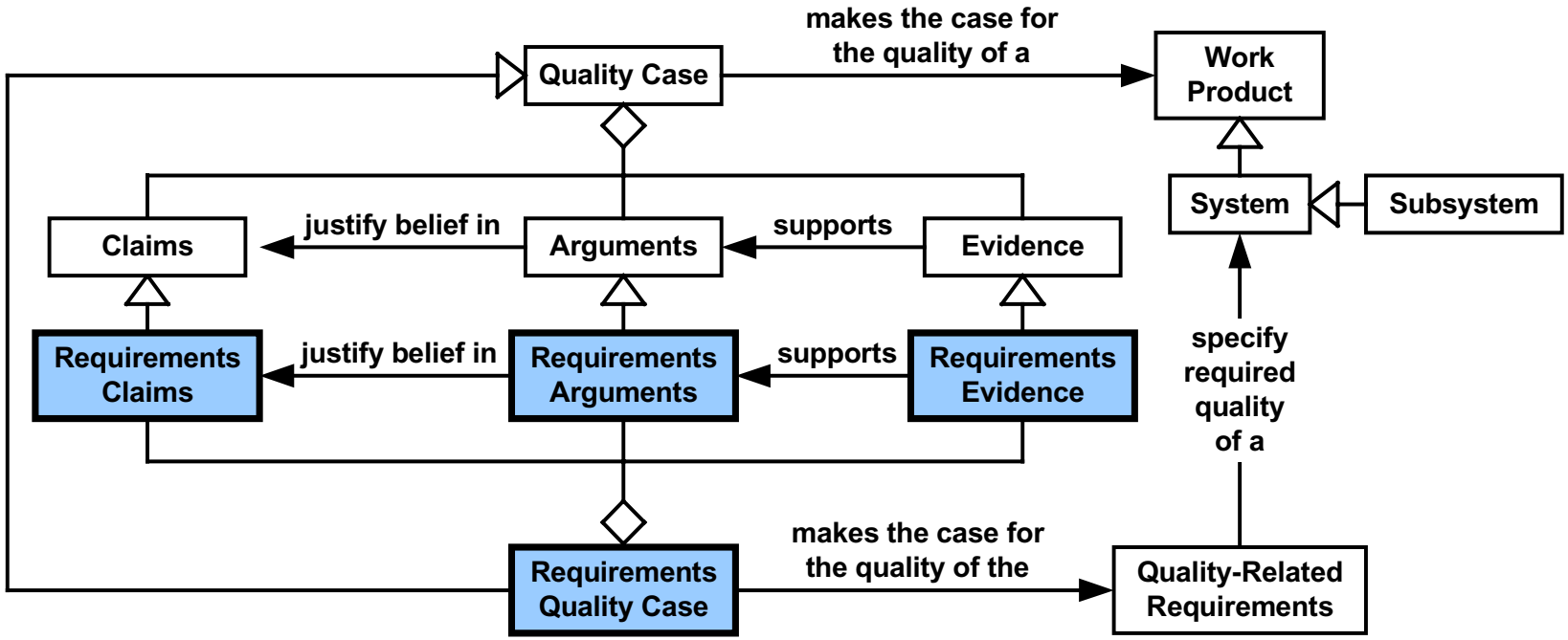
Makes Requirements Team's Case that their Relevant Requirements are:

- Ready to Drive the Engineering of the Architecture:
  - **Sufficient Quality**
    (e.g., are Correct, Complete, Consistent, Mandatory, Unambiguous, Verifiable, Usable, etc.)
  - **Sufficient Quantity**
    (e.g., Sufficient for Current Point in Project Schedule)
- Sufficient on which to base the Subsystem Architecture Assessment

QUASAR Requirements Quality Cases$_2$



QUASAR Requirements Quality Cases$_3$



QUASAR Requirements Quality Cases$_4$

Claims:

- Quality-Related Goals Sufficiently Meet Stakeholder Quality Needs
- Quality-Related Requirements Sufficiently Operationalize Quality Goals

Arguments:

- Existence and Quality of Quality-Related Requirements
- Requirements Engineering Trade-Offs, Rationales, and Assumptions

Evidence:

- Requirements Diagrams, Models, and Prototypes
- Requirements Repositories and Specification Documents
- Requirements Inspection and Checking Results

Example Requirements Quality Case$_1$

Example Requirements Reliability Case

Claims:

- Subsystem X Requirements Engineers claim that their Subsystem Goals Sufficiently Meet Stakeholder Reliability Needs:
  - "All Stakeholder Reliability Needs Allocated to Subsystem X have been Transformed into Subsystem X Reliability Goals."
- Subsystem X Requirements Engineers Claim that their Subsystem Reliability Requirements Sufficiently Help the Subsystem Meet its Reliability Goals:
  - "All Subsystem X Reliability Goals for this block/release have been Operationalized into Requirements."
  - "All Subsystem X Reliability Requirements for this block/release have been Properly Engineered."

Example Requirements Quality Case$_2$

Arguments:

- Subsystem X Reliability Requirements are:
  - Stored in the Project Requirements Repository
  - Published in the Subsystem X Requirements Specification
- Subsystem X Reliability Requirements in the Requirements Repository have been annotated as Reliability Requirements using Requirements Metadata.
- The Subsystem X Architects have verified the Feasibility of the Reliability Requirements given available Hardware and Software Technology.
- Appropriate Availability, Reliability, and Security Requirements Trade-Offs have been made.
- The Subsystem X Reliability Requirements have been Checked against a Checklist of Necessary Quality Characteristics (e.g., Correctness, Completeness, Consistency, Necessity, Unambiguous, Verifiability, and Usability).

Example Requirements Quality Case$_3$

Example Requirements Reliability Case

Evidence:

- **Requirements Traceability Matrix** showing Allocation of Reliability Requirements from Parent Subsystem A to Derived Reliability Requirements in Subsystem X
- Project **Requirements Repository** with Subsystem X Reliability Requirements identified
- **Reliability Section** in Subsystem X **Requirements Specification** Document
- **Reliability Subsection** of Subsystem X **Requirements Inspection Report**

Requirements Quality Case Challenges

Most Requirements Engineers are not trained in the Proper Engineering of Non-Functional Requirements (e.g., Quality Requirements).

Vague Unverifiable Goals are often Mistaken as Requirements.

Stakeholders often do *Not* know where to set Quality Measure Thresholds.

Requirements Repository is Huge and Complex.

Only Small Subset of the Requirements is Relevant to any specific Quality Factor or Quality Subfactor for any specific Subsystem.

Tracing Quality Requirements is more Difficult than Tracing Functional Requirements.

QUASAR Architectural Quality Cases

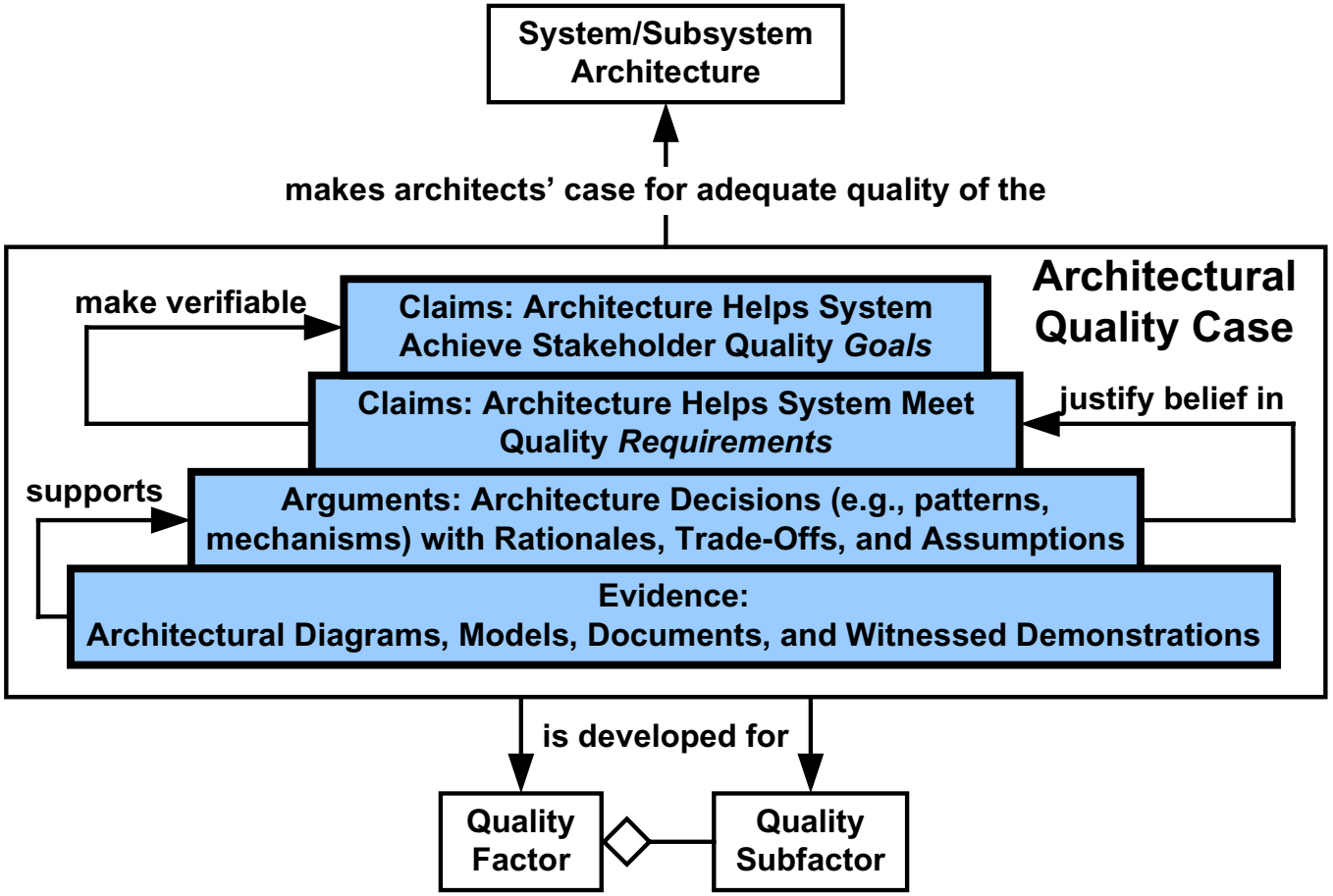
Architecture Quality Case

a Specialized Quality Case that is Limited to the Quality of the System/Subsystem Architectures

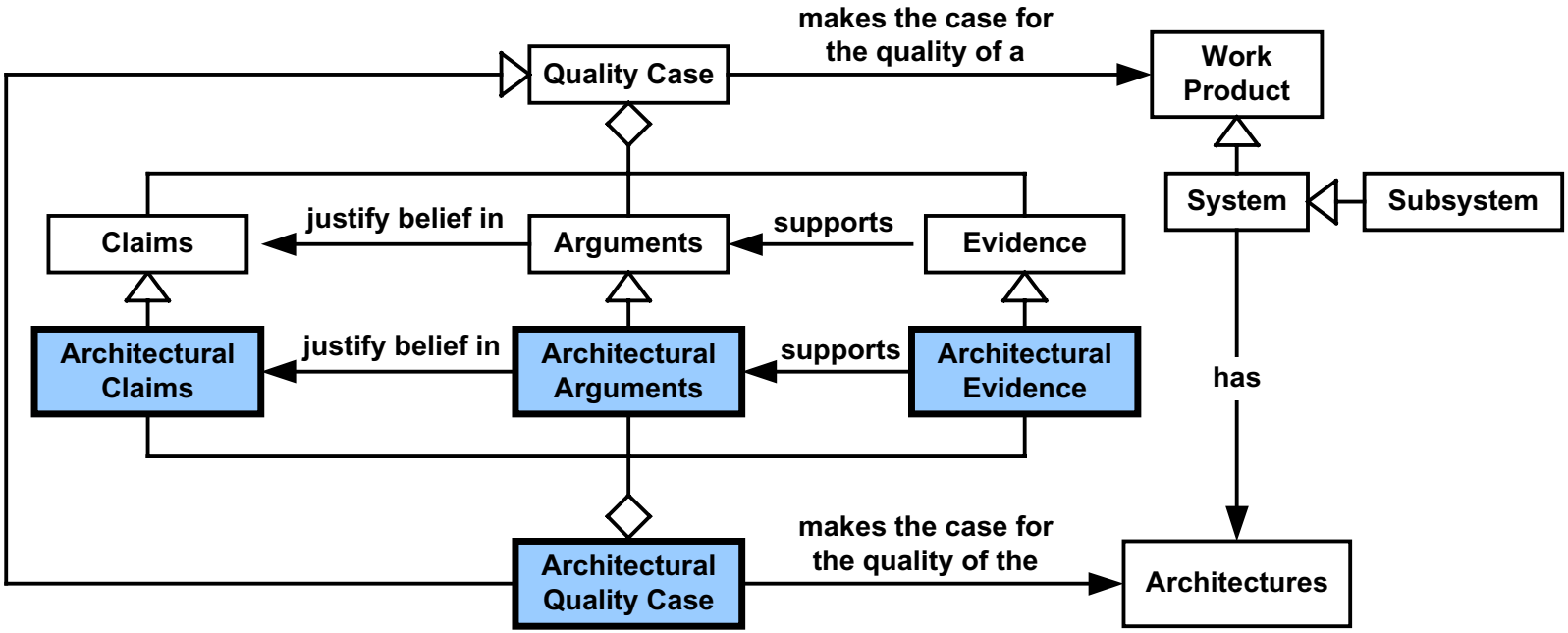
Make Architectures Team's Case that their Architecture(s) are:

- Ready to Drive the Engineering of the Design, Implementation, Integration, and Testing:
  - **Sufficient Quality**
    (e.g., Adequately Support the System's or Subsystem's Ability to meet its Quality-Related Requirements)
  - **Sufficient Quantity**
    (e.g., Sufficient for Current Point in Project Schedule)

QUASAR Architectural Quality Cases$_2$



QUASAR Architectural Quality Case



QUASAR Architectural Quality Cases$_4$

Claims:

- Architectures Sufficiently Supports System/Subsystem's Ability to Meet All Quality Goals and Quality Requirements

Arguments:

- Architectural Decisions (e.g., Architectural Mechanisms, Patterns, and Styles as well as Choice of OTS Components)
- Architectural Engineering Trade-Offs, Rationales, and Assumptions

Evidence:

- Architectural Diagrams, Models (Static and Dynamic), and Prototypes
- Architecture Documents and Architectural Whitepapers
- Properly Documented Architectural Simulation and Test Results
- Properly Witnessed Demonstrations

Example Architectural Quality Case$_1$

Example Protocol Interoperability Case

Claims:

- Subsystem X Architects Claim that their Subsystem Architecture Sufficiently Supports its following derived Protocol Interoperability *Goals*:
  - "Subsystem X will correctly use the interface protocols of all relevant external systems."
  - "Subsystem X will use open interface standards (i.e., industry standard protocols) when communicating with all external systems."

Example Architectural Quality Case$_2$

Example Protocol Interoperability Case

Claims:

- Subsystem X Architects Claim that their Subsystem Architecture Sufficiently Supports its following derived Protocol Interoperability *Requirements*:
  - "The subsystem shall use open interface standards (i.e., industry standard protocols) when communicating with external systems across all key interfaces identified in document X."
  - "The subsystem shall use the Ethernet over RS-232 for communication across interface X with external system Y."
  - "The subsystem shall use HTTPS for communicating securely when performing function X across interface Y with external system Z."

Example Architectural Quality Case$_3$

Arguments:

- **Layered Architecture**
  The subsystem uses a layered architecture.
  Rationale: Interface layer supports interoperability.
- **Modular Architecture**
  The subsystem architecture is highly modular.
  Rationale: Architecture includes modules (proxies) for interoperability.
- **Wrappers and Proxies**
  The subsystem architecture includes proxies that wrap the interfaces to external subsystems.
  Rationale: Proxies localize and wrap external interfaces.
- **Service Oriented Architecture (SOA)**
  The subsystem service oriented architecture uses XML, SOAP, and UDDI to publish and provide web services over the Internet to external client systems.
  Rationale: Standard languages and protocols support interoperability between heterogeneous systems.

Example Architectural Quality Case$_3$

Evidence:

- **Context Diagram**
  (shows external interfaces requiring protocols)
- **Architectural Class Diagram**
  (shows modularity and location of proxies and web services)
- **Allocation Diagram**
  (shows allocation of software modules to hardware - modularity)
- **Layer Diagram**
  (shows architectural layers)
- **Activity/Collaboration Diagrams**
  (show proxies, wrappers, and the source and use of services)
- **Interoperability Whitepaper**
- **Vendor-Supplied Technical Documentation**
  (show COTS product support for SOA and associated protocols)

QUASAR Quality Case Responsibilities

Requirements Engineers and Architects' Responsibilities:

- Prepare Quality Cases
- Provide Preparation Materials (including Presentation Materials and Quality Cases) to Assessors Prior to Assessment Meeting
- Present Quality Cases (Make their Case to the Assessors)
- Answer Assessors' Questions

Assessor Responsibilities:

- Prepare for Assessments
- *Actively* Probe Quality Cases
- Develop Consensus regarding Assessment Results
- Determine and Report Assessment Results:
  - Present Outbriefs
  - Publish Reports

Architectural Quality Case Challenges

Huge and Complex System Architectures

Only Small Subset of the Architecture (a.k.a., focus area) is Relevant to any one Quality Factor or Quality Subfactor.

Quality Cases still Contain a Large Amount of Information.

Claims, Arguments, and a large amount of Evidence are typically Text.

Easy to get Lost in Real-World Quality Cases

Hard to know that:

- Arguments *Sufficiently* Justify Belief in Claims
- Evidence *Sufficiently* Supports Arguments

Need practical way to Communicate, Summarize, and Act as Index to the Quality Case Essentials

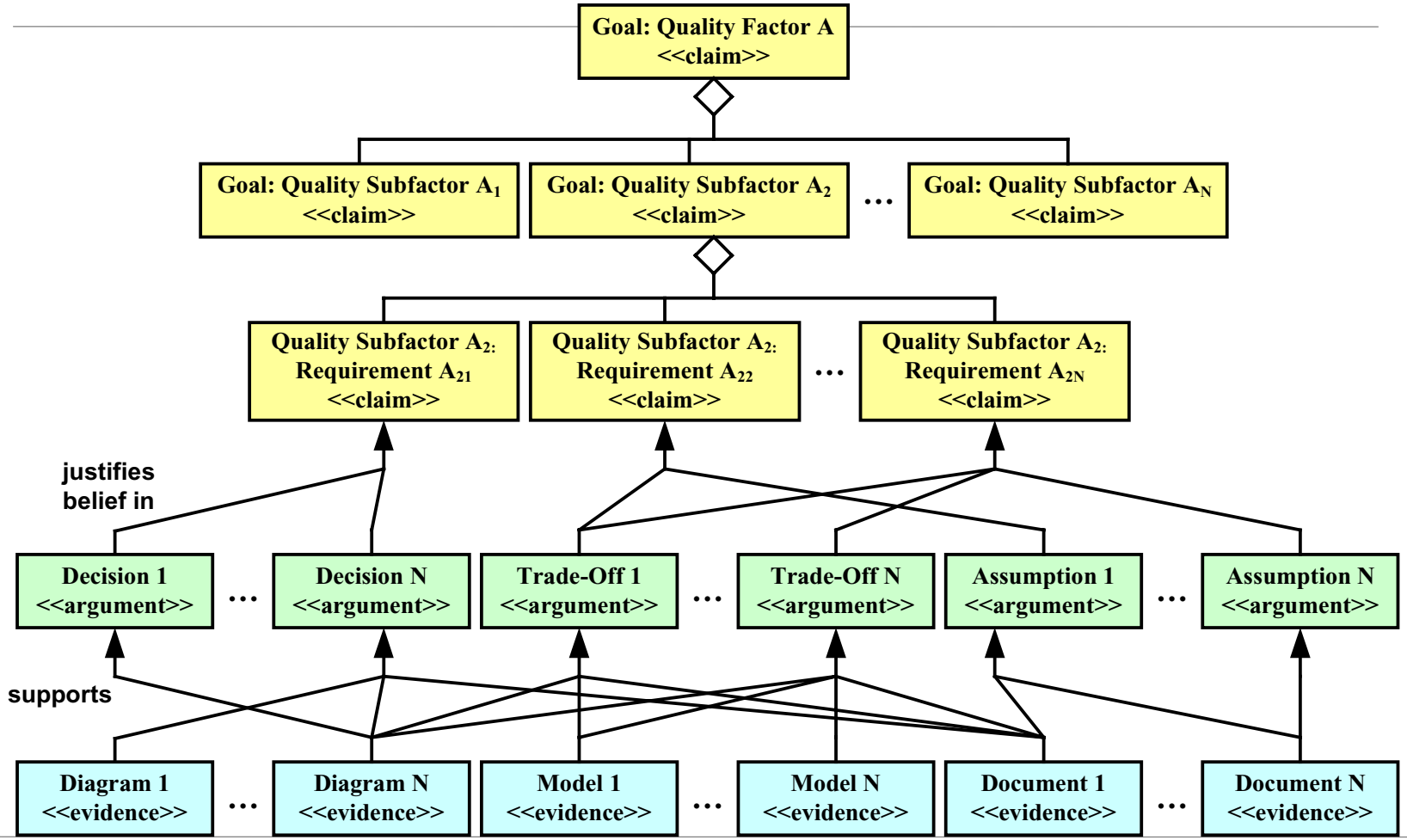
Quality Case Diagram

A *Layered* UML Class Diagram that Labels and Summarizes the parts of a *Single* Quality Case:

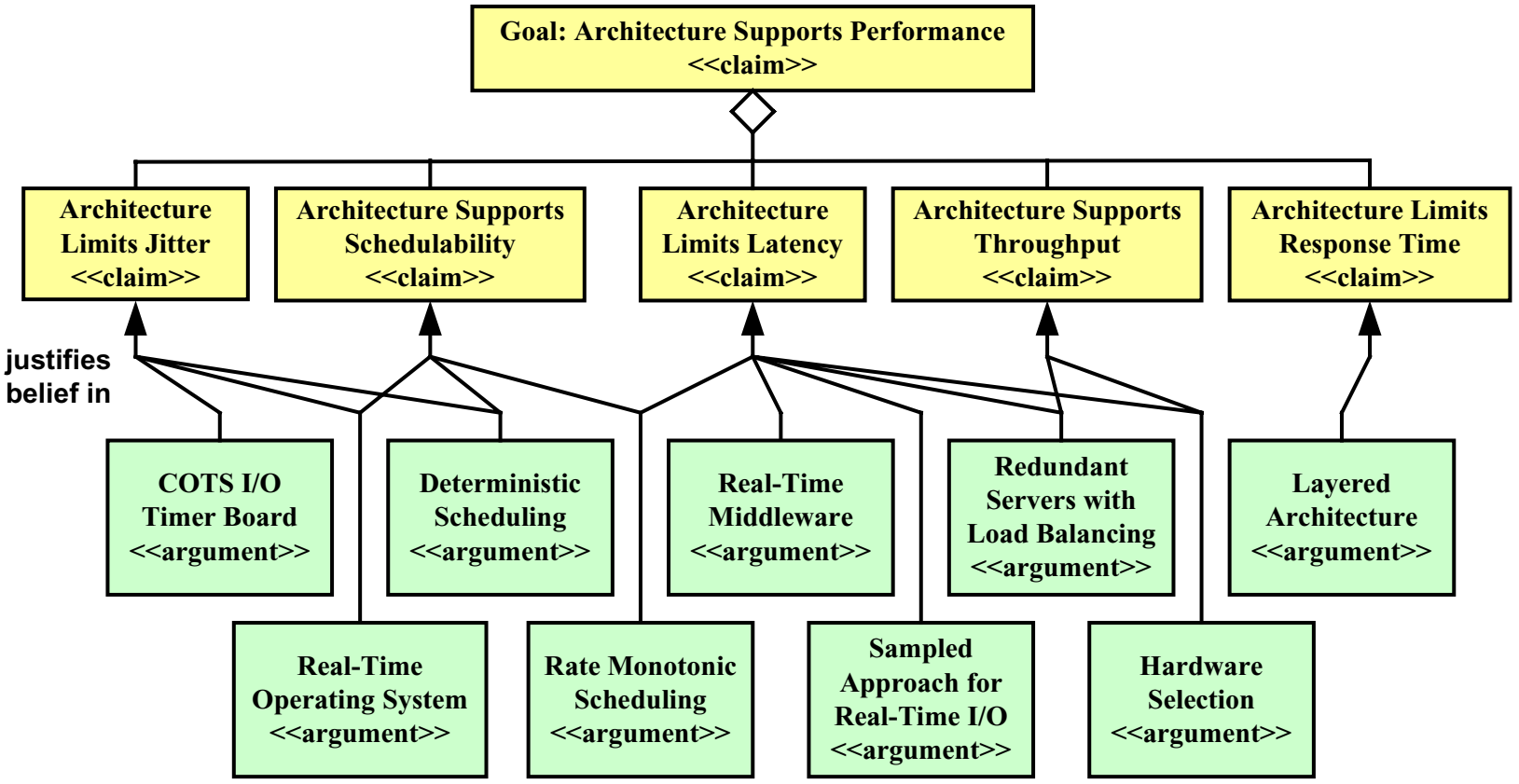
- Classes:
  - Claims
  - Arguments
  - Evidence
- Relationships Among Them:
  - Aggregation Relationships Between Claims
  - "Justifies Belief In" Associations from Arguments to Claims
  - "Supports" Associations from Evidence to Arguments

Acts as an Index and Guide to the Quality Case

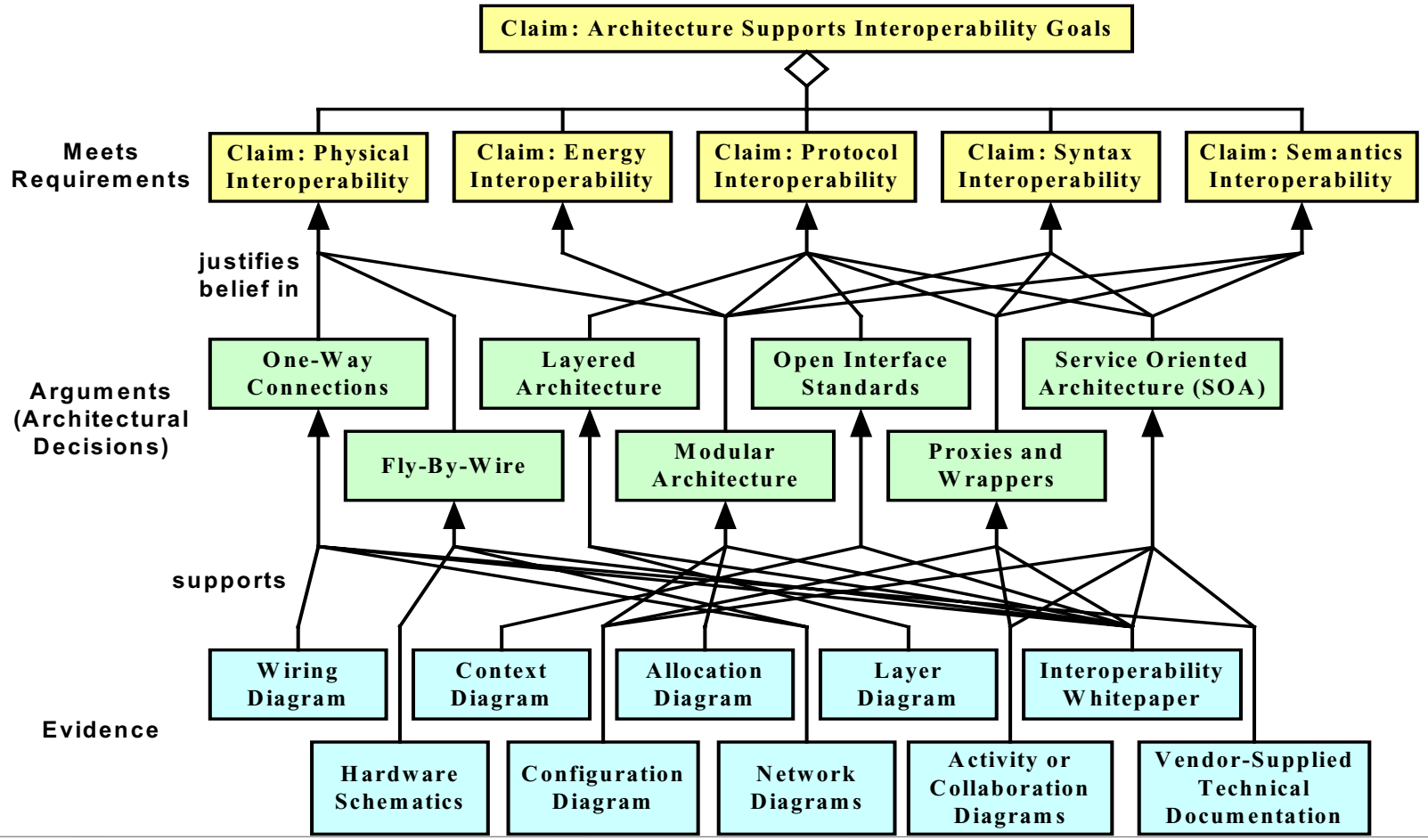
Quality Case Diagram Notation



Example Partial Architectural Performance Case Diagram



Architectural Interoperability Case Diagram



System:

QUASAR assesses System and Subsystem Requirements and Architectures

What is a System?

System

> a Major, Cohesive, Executable, and Integrated Set of *Architectural Elements* that Collaborate to Provide the Capability to Perform one or more related *Missions*

Systems are Decomposed into Architectural Elements:

- Subsystems
- Data Components
- Hardware Components
- Software Components
- People Roles (e.g., Operators, Administrators)
- Procedural Components
- Facilities, Equipment, Materials

Systems Imply

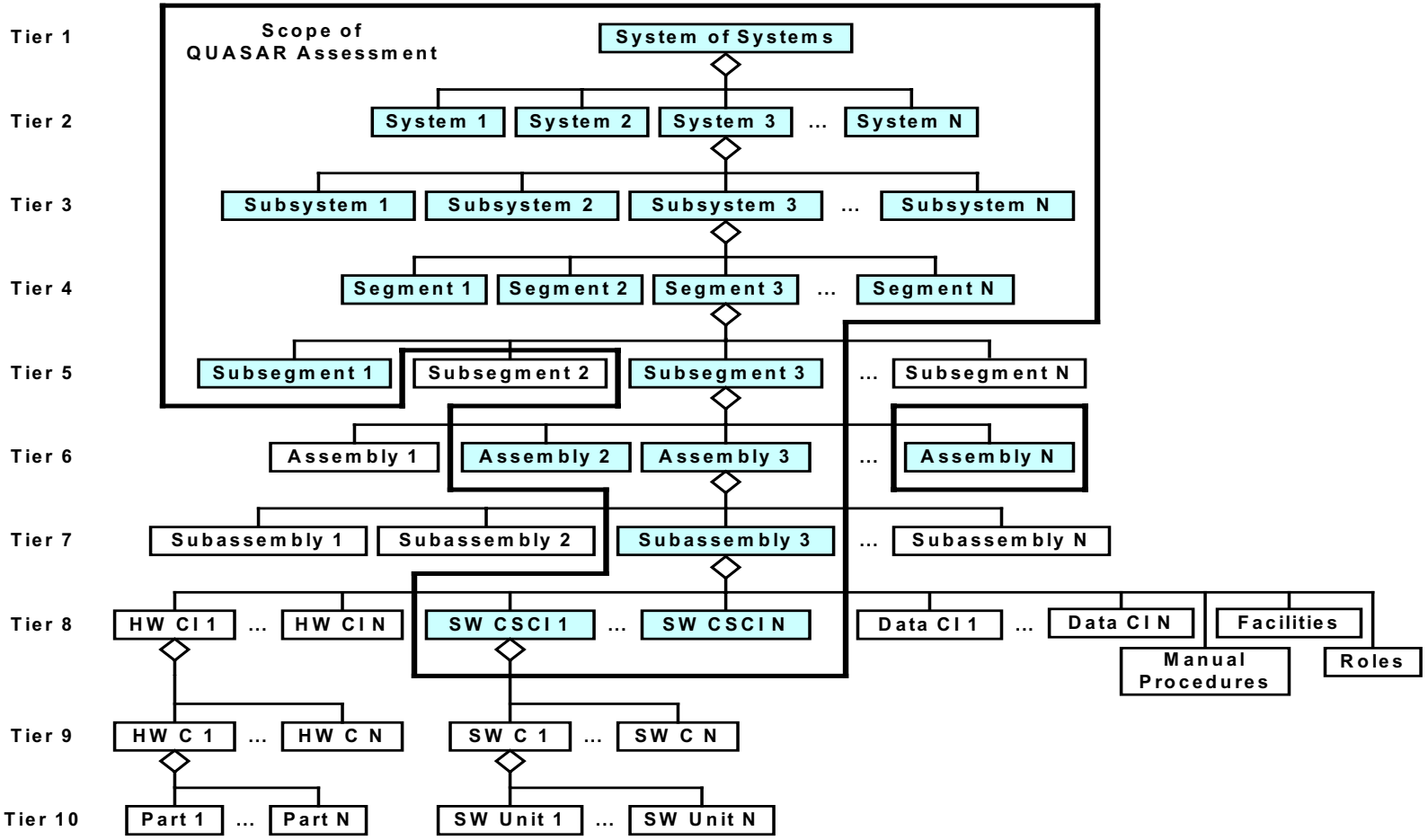
Multiple Static and Dynamic Logical and Physical “Structures” that exist at Multiple ‘Levels’ in the System:

- Static Functional Decomposition Logical Structure
- Static Subsystem Decomposition Physical Structure
- Hardware, Software, and Data Structures
- Allocation Structure (Software and Data to Hardware)
- Network Structure
- Concurrency (Process) Structure

Multiple Specialty Engineering Focus Areas (e.g., Performance, Reliability, Safety, and Security)

Requirements are Derived and Allocated to Lower-Level Architectural Elements

System and QUASAR Scope
(Static Subsystem Decomposition Physical View Only!)



System Architectures:

Strategic Pervasive Decisions, Inventions, and their Rationales

What is a System Architecture?$_1$

System Architecture

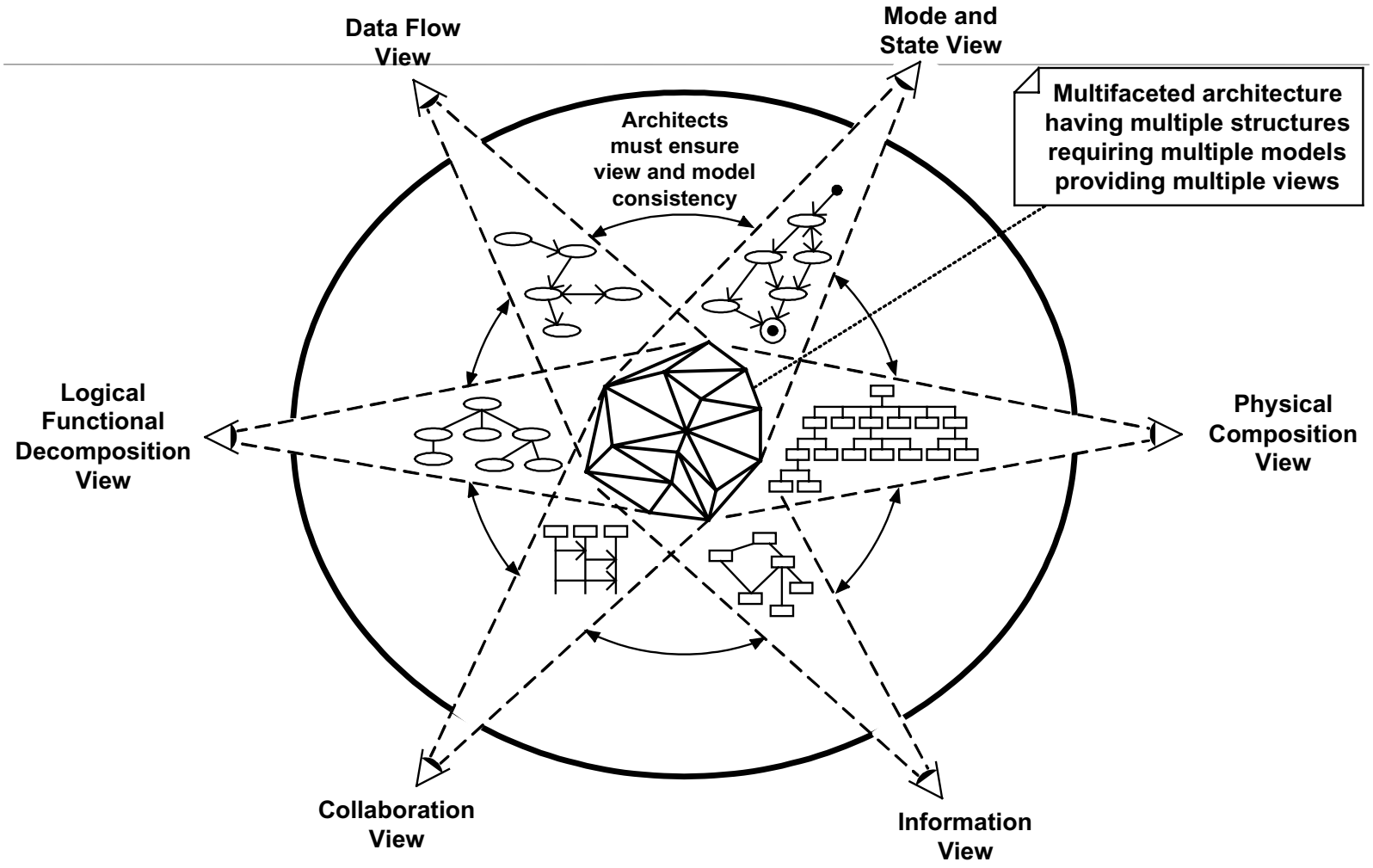
> the *Most Important, Pervasive, Top-Level, Strategic Inventions, Decisions, Engineering Trade-Offs,* associated *Rationales,* and *Assumptions* about *How* a System and its Subsystems will meet their Derived and Allocated Requirements

What is a System Architecture?$_2$

System Architecture Includes:

- **The System's Numerous Static and Dynamic, Logical and Physical Structures**
  (i.e., Essential Architectural Elements, their Relationships, their Associated Blackbox Characteristics and Behavior, and how they Collaborate to Support the System's Mission and Requirements)
- **Architectural Inventions and Decisions**
  (e.g., Styles, Patterns, and Mechanisms used to ensure that the System Achieves its Architecturally-Significant Product and Process Requirements (esp. Quality Requirements or 'ilities')
- **Strategic and Pervasive Design-Level Decisions**
  (e.g., using a *Design* Paradigm such as Object-Orientation or Mandated Widespread use of common Design Patterns)
- **Strategic and Pervasive Implementation-Level Decisions**
  (e.g., using a Safe Subset of C++)

Some Example Views of Models of Structures



Architecture vs. Design

Architecture	Design
Pervasive (Multiple Components)	*Local* (Single Components)
Strategic Decisions and Inventions	*Tactical* Decisions and Inventions
Higher-Levels of System	*Lower-Levels* of System
Huge Impact on Quality, Cost, & Schedule	*Small Impact* on Quality, Cost, & Schedule
Drives Design and Integration Testing	*Drives* Implementation and Unit Testing
Driven by Requirements and Higher-Level Architecture	*Driven by* Requirements, Architecture, and Higher-Level Design
Mirrors Top-Level Development Team Organization (Conway's Law)	*No* Impact on Top-Level Development Team Organization

Architectural Documentation Current-State

System Architecture Documents:

- Mostly natural language Text with Visio-like Diagrams
- Logical (functional) and Physical Architecture

DOD Architecture Framework (DODAF):

- All-Views, Operational Views, Systems Views, and Technical Standards Views for allocating Responsibilities to Systems and Supporting System Interoperability

Models (both static and dynamic; logical and physical):

- Tailored UML becoming *de facto* Industry Standard
- SysML starting to become Popular

Visio Diagrams as Wall Posters

Whitepapers, Reports, and other Specialty-Engineering Documents:

- Performance, Fault Tolerance, Safety, Security

What an Architecture is NOT

A System Architecture is *Not* an Architectural:

- Plan
- Method
  (architecting procedures and architecture documentation standards)
- Team Organization Chart
  (in spite of Conway's Law)
- Development Schedule

QUASAR assesses Actual Architectures:

- As they Currently Exist (i.e., a Snapshot in Time)
- Not Good Intentions

Systems Architect:

Responsible for the Architecture

What is a Systems Architect?$_1$

A *Role* played by a Systems Engineer, who is Responsible for:

- *Developing* one or more System or Subsystem Architectures
- *Ensuring* the *Quality* of the System or Subsystem Architectures
- Ensuring the *Integrity* of the System or Subsystem Architectures during Design, Implementation, Manufacture, and Deployment (e.g., Installation and Configuration)
- *Communicating* the System or Subsystem Architectures to their Stakeholders
- *Maintaining* the System or Subsystem Architectures

What is a Systems Architect? 2

A *Role* that*:*

- Requires Significant:
  - Training
  - Experience (Apprenticeship)
  - Mindset:
    - Big Picture
    - Generalist
  - Communications Ability
- Should Probably be a *Job Title*
- But may *Not* be a Job Title

Architecturally-Significant Requirements:

Requirements Driving the Architecture

Architecturally-Significant Requirement

Architecturally-Significant Requirements

> any Requirement that has a Significant Impact on a System / Subsystem Architecture

Architecturally-Significant Requirements typically include:

- Quality Requirements, which specify a Minimum Amount of some Quality Factor or Quality Subfactor
- Architectural Constraints
- Primary Mission Functional Requirements

Quality Requirements are often the:

- Most Important
- Least Well Engineered

Quality Requirements

Format

Under condition(s) X, the system/subsystem shall exhibit quality criterion Y meeting or exceeding threshold Z.

Bad Example(s)

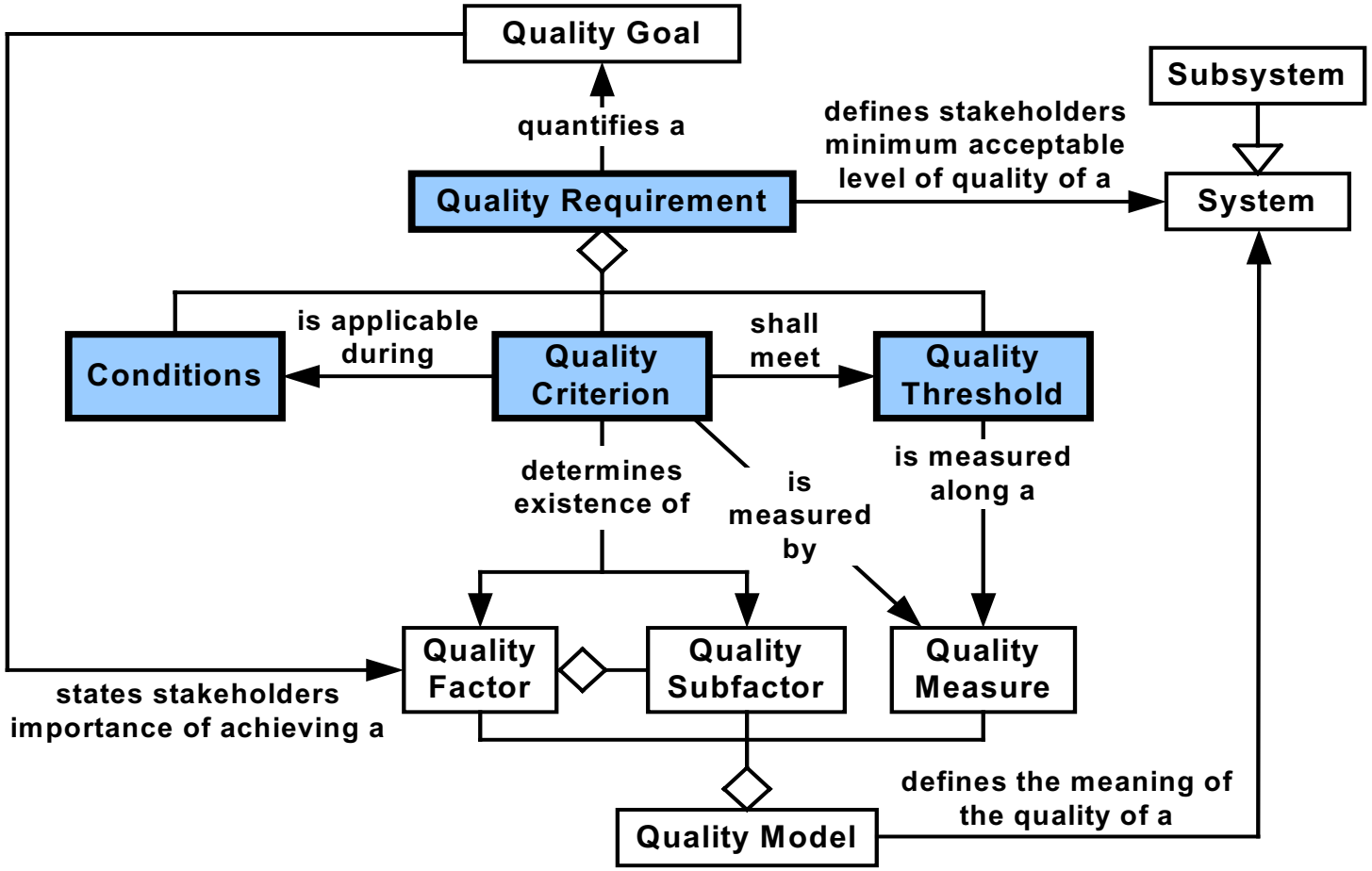
The system shall be highly reliable, robust, safe, secure, stable, etc.

Good Example (Stability)

Under normal operating conditions*, the system shall ensure that the mean time between the failure of mission critical functionality* is at least 5,000 hours of continuous operation.

* Must be Properly defined in the Project Glossary

Quality Requirements – Quality Cases



Why Use QUASAR?:

Quality, Requirements, and Architecture

Why Use QUASAR?$_1$

Requirements and Architecture are the first two Opportunities to make Major Engineering Mistakes.

Architecture and associated Architecturally-Significant Requirements Affect:

- Project Organization and Staffing (Conway's Law)
- Design, Implementation, Integration, Testing, and Deployment Decisions

QUASAR emphasizes using a common project-specific Quality Model:

- Quality Factors and Quality Subfactors
  - ? Quality Requirements
  - ? Quality of Architecture
  - ? Quality of System

QUASAR Ensures Specification of *Architecturally-Significant* Requirements.

Why Use QUASAR?$_2$

Architecturally-Significant, Quality-Related Requirements and their associated Architectural Decisions *Drive* the System / Subsystem:

- Ultimate Quality
- Development Schedule
- Development Costs
- Sustainment Costs
- Maintainability and Upgradeability
- Acceptance and Usage by Stakeholders

Why Use QUASAR?$_3$

System Quality is Union of Relevant Quality Factors:

- **Availability**
- **Functionality**
- **Interoperability**
- **Modifiability**
- **Performance**
- **Reliability**
- **Robustness (Error, Failure, and Fault Tolerance)**
- **Safety**
- **Security**
- **Scalability**
- **Stability**
- **Testability**
- **etc.**

Why Use QUASAR?$_4$

Determine *Actual System/Subsystem Requirements and Architecture*:

- Quality
- Maturity and Completeness
- Integrity and Consistency
- Usability

Identify System/Subsystem Requirements and Architectural *Defects* Early:

- Fix Defects Early
- Decrease Development and Maintenance Costs
- Decrease Schedule

Why Use QUASAR?$_5$

Identify (and thereby help Manage) Risks:

- Requirements Risks
- Architecture Risks
- System Risks

Provide Acquirer/Management:

- *Visibility* into
- *Oversight* over

the System/Subsystem Requirements and Architecture

Determine *Compliance* :

- Requirements and Architecture with Contract (Acquirer) Requirements
- Architecture with System/Subsystem (Developer) Requirements

Why Use QUASAR?$_6$

Develop *Consensus*:

- Among Developers (e.g., Requirements and Architecture Teams)
- Between Acquirer and Developer Organization

QUASAR Helps:

- *Requirements Engineers* Succeed
- *Architects* Succeed
- *Program* Succeed

QUASAR Philosophy:

Reasons to use QUASAR

Assessment Philosophy$_1$

Informal *Peer Reviews* are Inadequate:

- Too Informal
- Lack of *Independent* Expert Input
- Requirements and Architecture are too Important

Quality Requirements:

- Most important Architecturally-Significant Requirements
- Largely Drive the System Architecture
- Criteria against which the System Architecture is Assessed

Assessment Philosophy$_2$

Requirements Engineers (REs) should *Make Case* to Assessors:

- REs *should* know Stakeholder Needs and Goals
- REs *should* know What they Did and Why
  (Architecturally-Significant Requirements, Rationales, & Assumptions)
- REs *should* Know Where they Documented their Decisions

Architects should *Make Case* to Assessors:

- Architects *should* know Architecturally-Significant Requirements
- Architects *should* know What they Did and Why
  (Inventions, Decisions, Rationales, Trade-Offs, and Assumptions)
- Architects *should* know Where Documented their Decisions

Assessment Philosophy$_3$

Assessors should *Actively* Probe Quality Cases:

- **Claims Correct and Complete?**
  Do the Claims include *all* relevant Quality Factors, Quality Subfactors, Quality Goals, and Quality Requirements?
- **Arguments Correct, Complete, Clear, and Compelling?**
  Do the Arguments include *all* relevant Quality Factors, Quality Subfactors, Quality Goals, Quality Requirements, Inventions, Decisions, Assumptions, and Rationales?
- **Arguments Sufficient?**
  Are the Arguments Sufficient to Justify the Claims?
- **Evidence Sufficient?**
  Is the Evidence Sufficient to Support the Arguments?
- **Current Point in the Schedule?**
  Are the Claims, Arguments, and Evidence appropriate for the Current Point in the Schedule?

QUASAR Method: Phases and Tasks

QUASAR Method - Phases

Four **Phases**:

1. System Assessment Initiation (SAI)

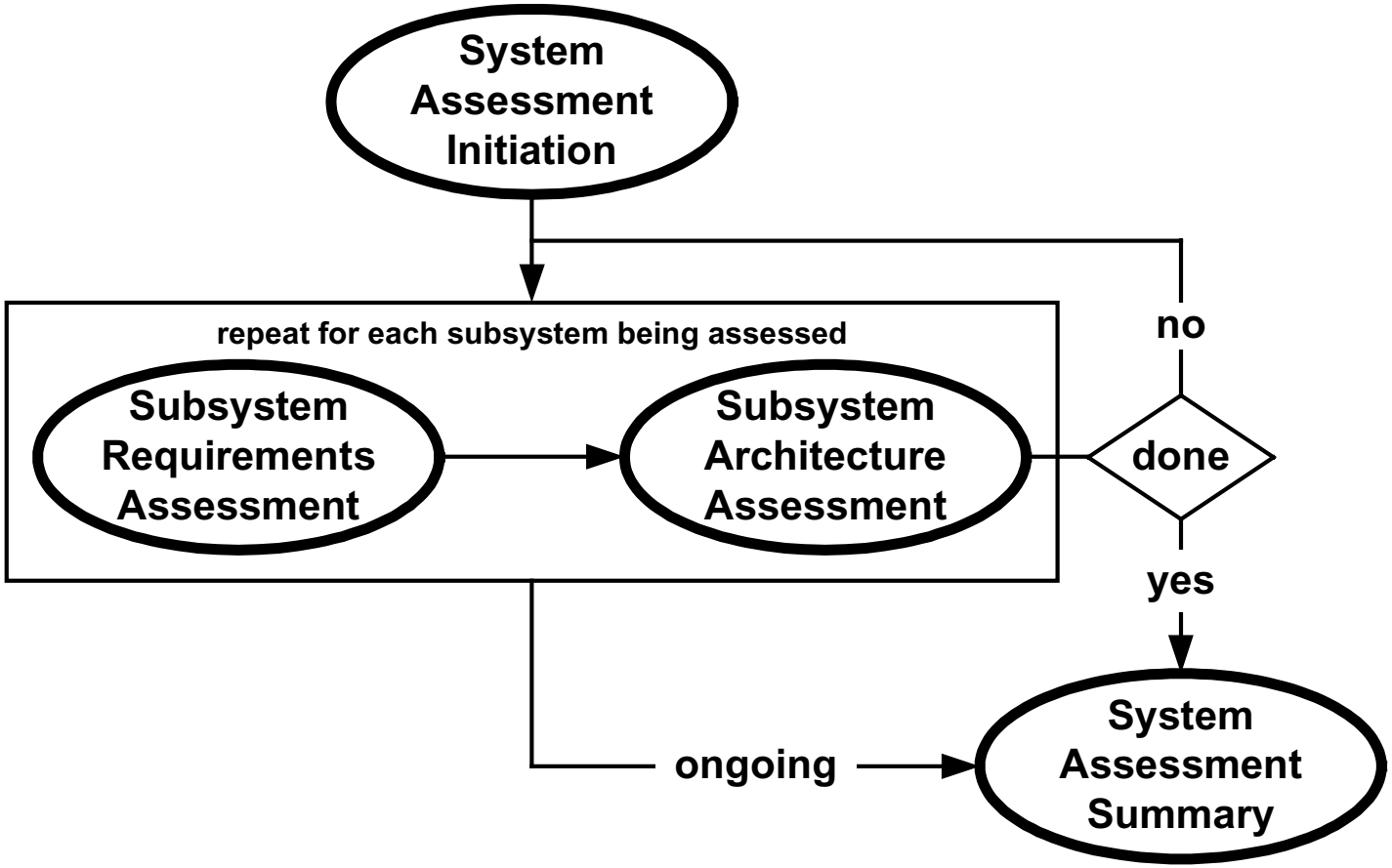
   For each Subsystem to be assessed:

   2. Subsystem Requirements Assessment (SRA)
   3. Subsystem Architecture Assessment (SAA)

4. System Assessment Summary (SAS)

Phase 2 and 3 may also apply to system as a whole.

QUASAR Phases

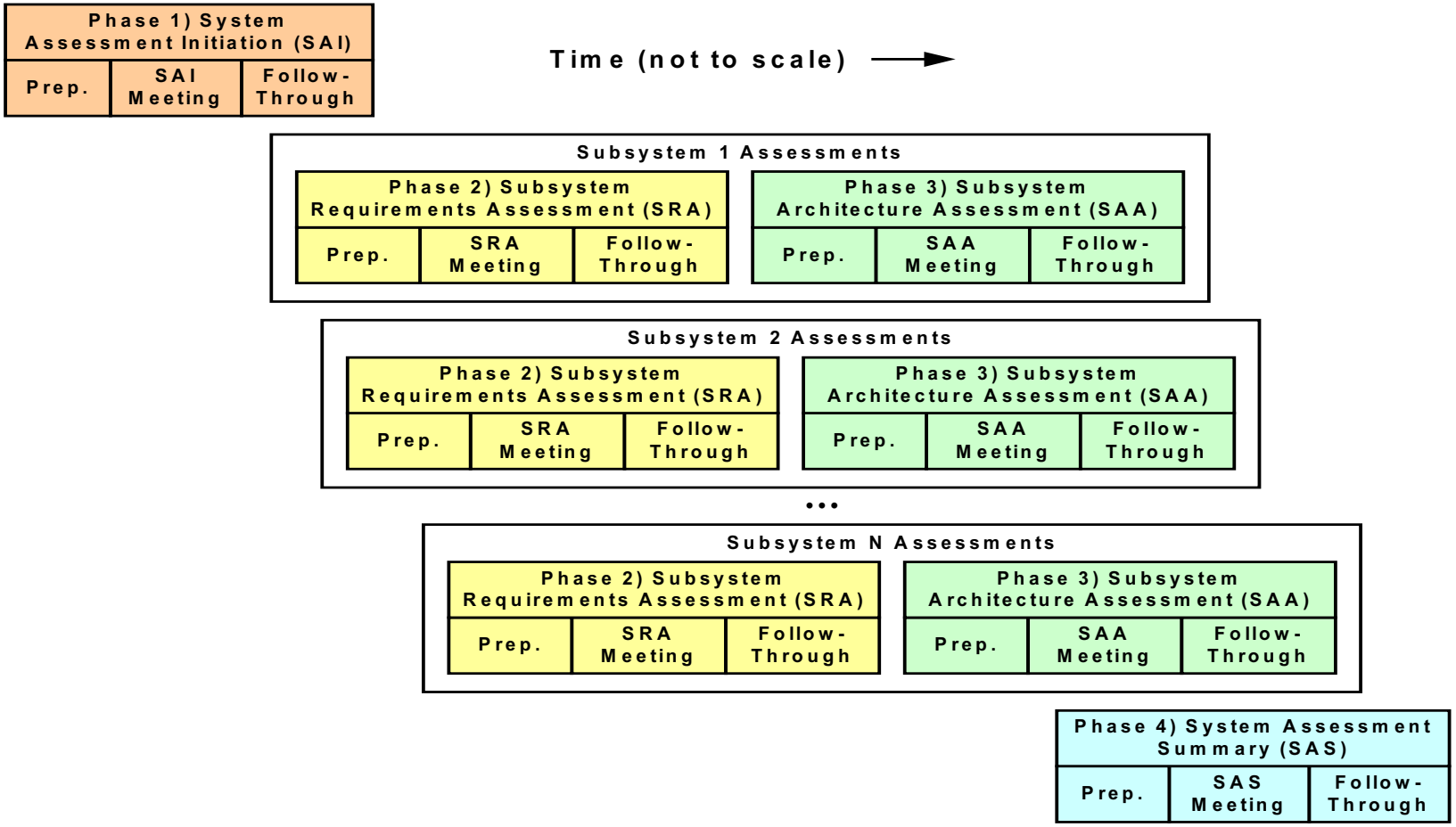


QUASAR Methods - Tasks

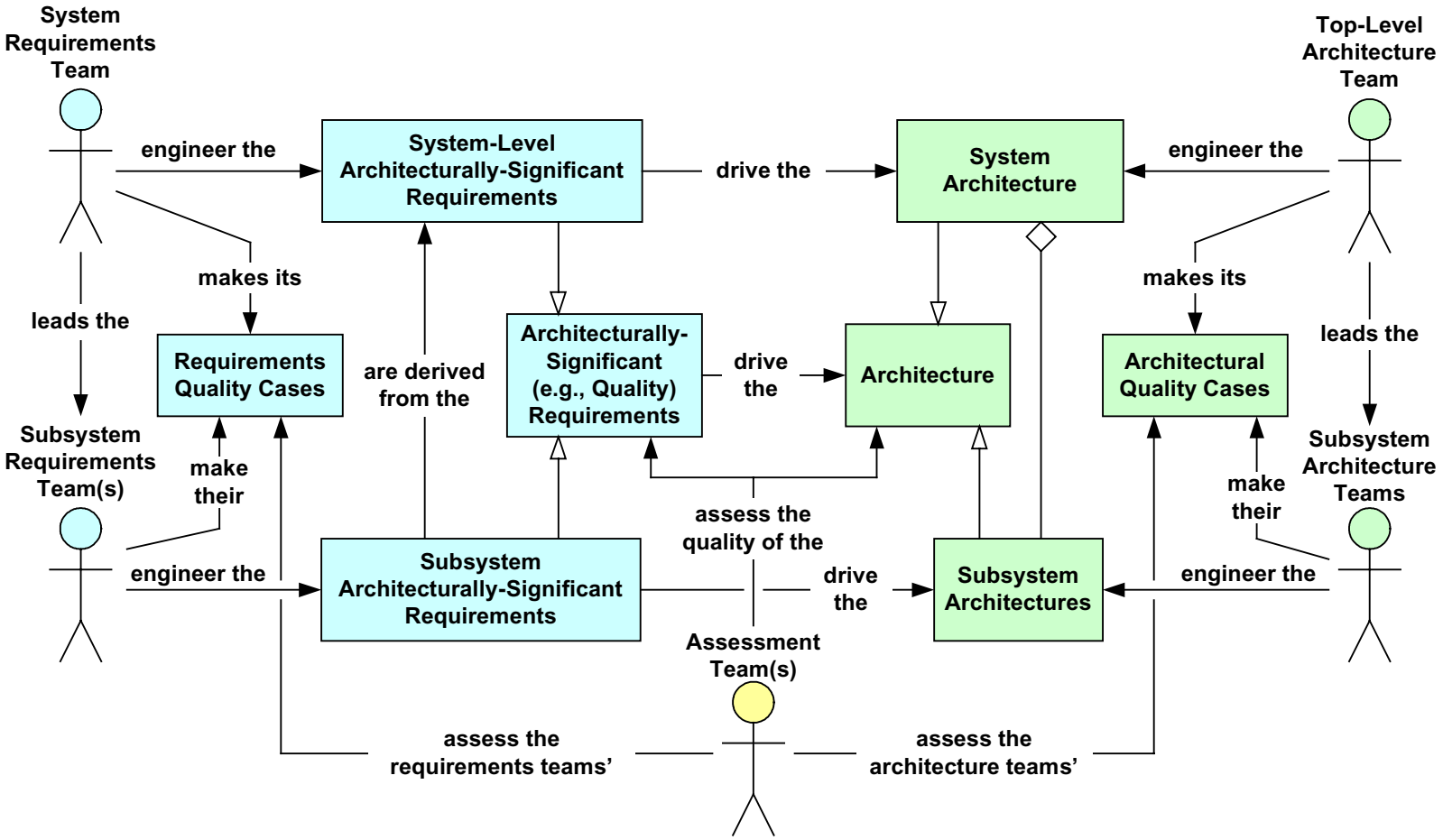
Each Phase consists of same 3 ***Tasks***:

- Preparation
- Meeting
- Follow-Through

QUASAR Phases and Tasks

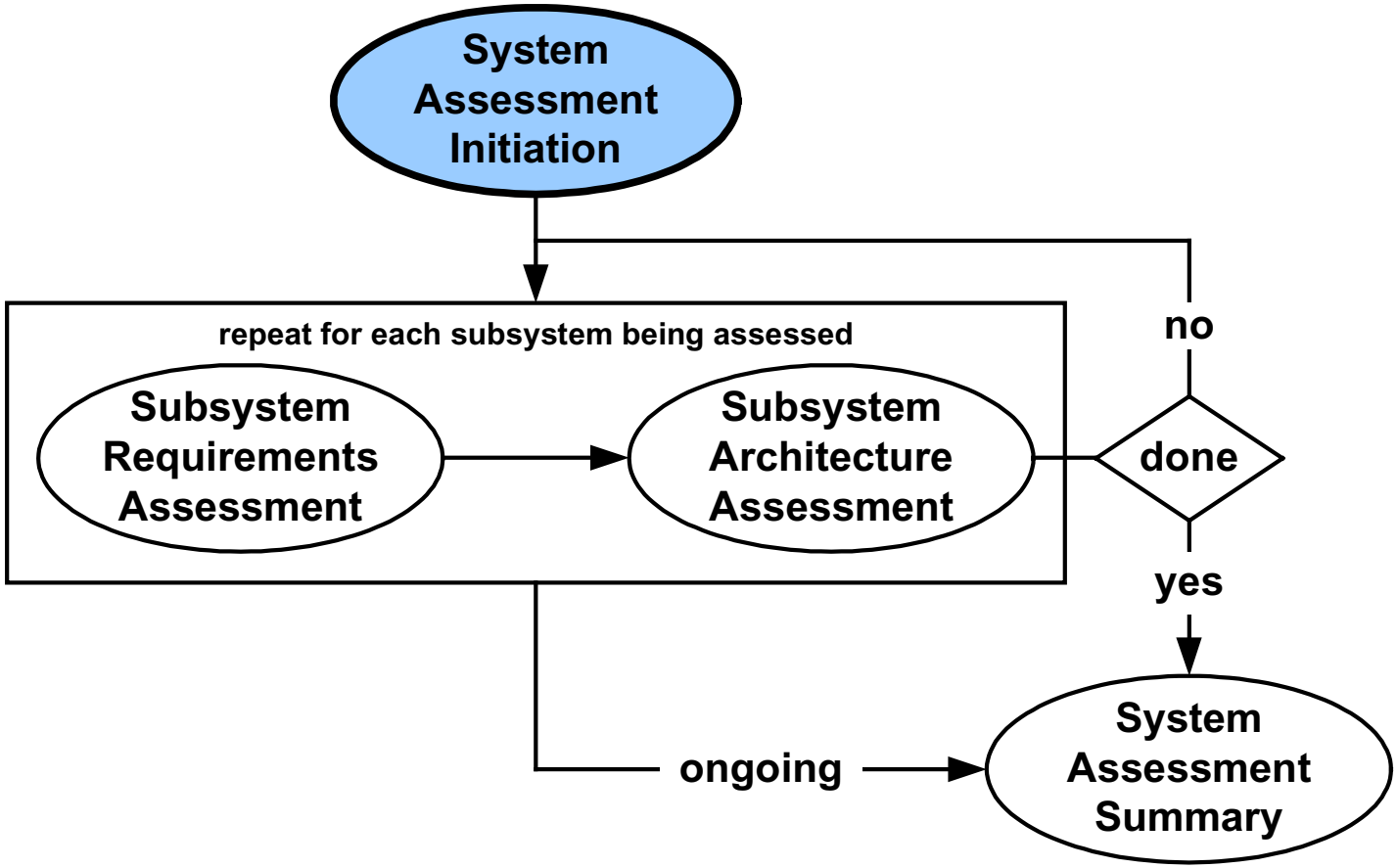


Quasar Teams and their Work Products



Phase 1:
System Assessment Initiation (SAI)

System Assessment Initiation (SAI)



Phase 1) SAI – Topics

System Assessment Initiation (SAI) Phase:

- Objectives
- Principles
- Challenges
- Tasks:
  - Preparation
  - Meeting
  - Follow-Through
- Primary Work Products
- Team Memberships
- Lessons Learned

Phase 1) SAI – Objectives

Prepare Teams for Requirements and Architecture Assessments

Develop Consensus:

- Scope of Assessments
- Schedule Assessments
- Tailor the Assessment Method and associated Training Materials

Produce and Publish Meeting Outbrief and Minutes

Manage Action Items

Capture Lessons Learned

Tailor/Update QUASAR Method and Training Materials

Phase 1) SAI – Principles

It is Important to:

- **Develop Consensus** among Teams
- **Scope the Assessment** to meet Project-Specific Needs and Resources
- **Tailor the Assessment Method** to meet specific Needs of the Overall Assessment

Subsystem Assessments must be scheduled to **Ensure Availability** of the:

- Requirements and Architecture
- Required Resources (e.g., people and funding)

Phase 1) SAI – Challenges$_1$

Acquirer and Development Organizations may Disagree as to the:

- Need to Independently Perform Quality Assessments
- Relative Importance of Quality Factors, Quality Subfactors, and Related Goals and Requirements

It can be Difficult to reach Consensus on the Scope of the Assessments in terms of the:

- Number and Identity of Subsystems to Assess
- Number and Identity of Quality Factors and Quality Subfactors
- Tailoring of the QUASAR Method

Phase 1) SAI – Challenges$_2$

Quality Assessments of System and Subsystem Architectures and their Architecturally-Significant Requirements may not have been included in the Project:

- Request for Proposal (FRP)
- Contract
- Budget and Schedule

It is often very Difficult to obtain Commitment of Resources:

- Availability of Requirements Engineers and Systems Architects
- Availability of Assessors with Adequate Experience and Expertise
- Consensus on Schedule
- Budget Funding to Pay for the Assessment

Phase 1) SAI – Preparation Task

- Management Team staffs Assessment Team
- Process and Training Teams train Assessment Team
- Assessment Team identifies:
  - System Requirements Team
  - System Architecture Team
- Process and Training Teams train System Requirements and Architecture Teams
- Assessment, Requirements, and Architecture Teams collaborate to Organize SAI Meeting (i.e., Attendees, Time, Location, Agenda)

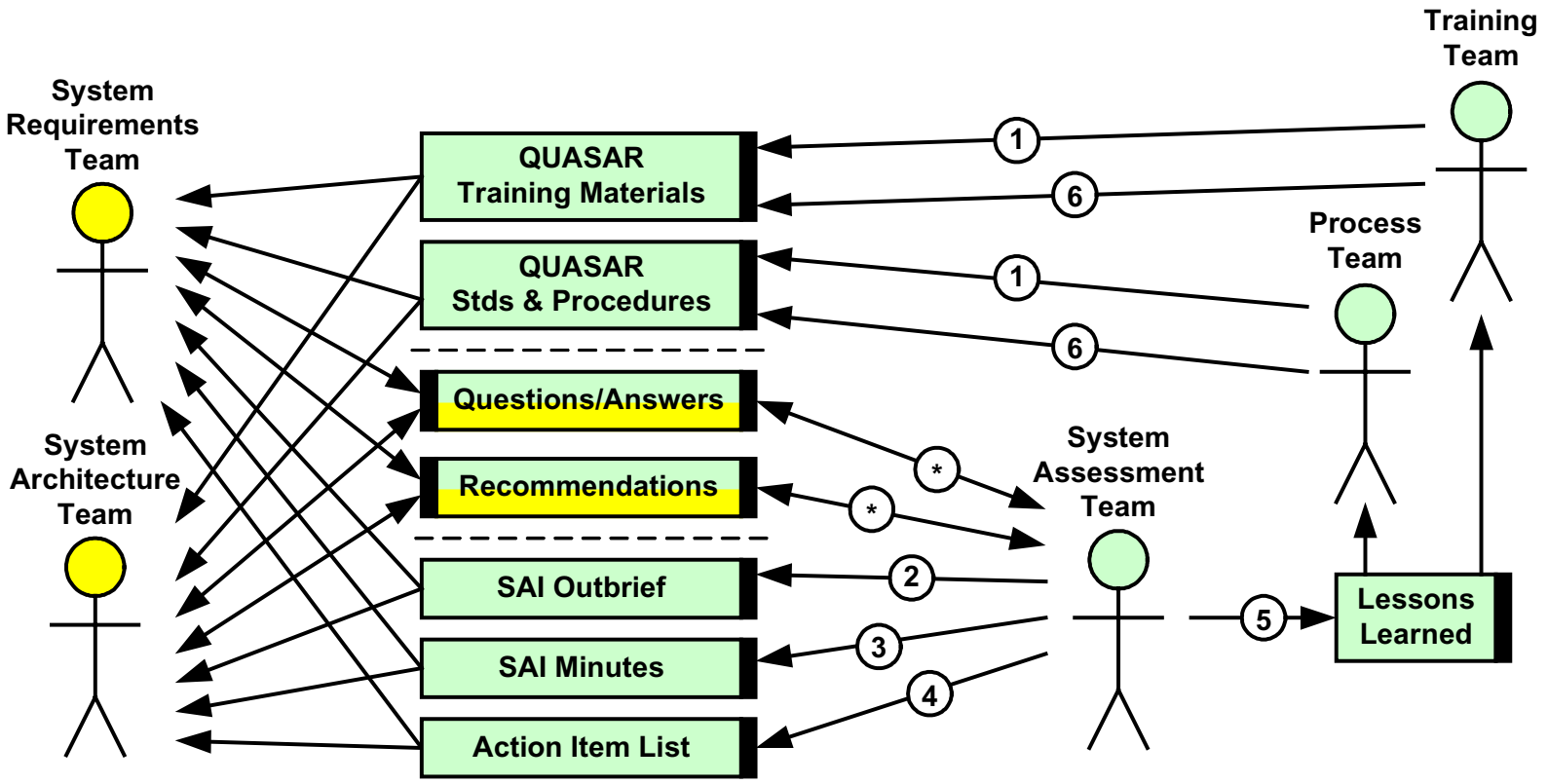
Phase 1) SAI – Meeting Task

- Assessment, System Requirements, and System Architecture Teams Collaborate to determine Assessment Scope:
  - Quality Factors and Quality Subfactors underlying Assessment
  - Architecturally-Significant Product and Process Requirements
  - Subsystems/Architectural Elements/Focus Areas to Assess (Number and Identity)
  - Assessment Resources (e.g., Time, Budget, and Staffing)
- Teams Collaborate to develop Initial Assessment Schedule
- Teams Collaborate to tailor QUASAR Method
- Assessment Team captures Action Items

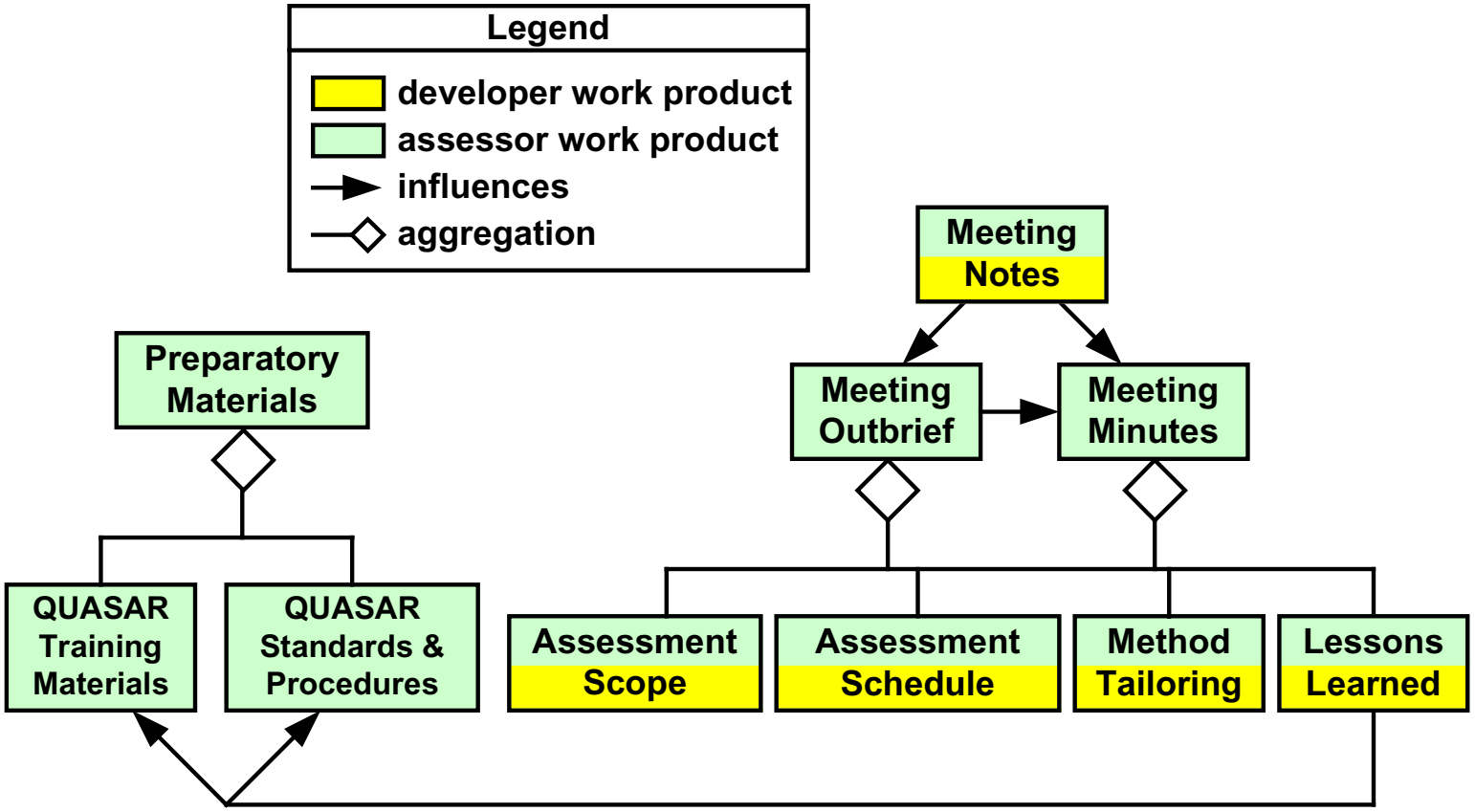
Phase 1) SAI – Follow-Through Task

- Assessment Team develops and presents Meeting Outbrief
- Assessment Team develops, reviews, and distributes Meeting Minutes
- Assessment/Process/Training Teams tailor, internally review, and distribute:
  - QUASAR Procedure, Standards, and Templates
  - QUASAR Training Materials
- Assessment Team distributes Assessment Schedule
- Teams obtain Needed Resources
- Assessment Team captures Lessons Learned
- Assessment Team Manages Action Items

Phase 1) SAI – Work Product Flow



Phase 1) SAI – Work Products



Phase 1) SAI – Team Memberships$_1$

Assessment Team (Assessors):

- Assessment Team Leader
- Meeting Facilitator
- Subsystem Liaisons to:
  - Requirements Team
  - Architecture Team
- Subject Matter Experts (SMEs)
  - Application Domain
  - Specialty Engineering
- Acquisition/Customer *Observers*
- Scribe

Phase 1) SAI – Team Memberships$_2$

System Requirements Team (Requirements Engineers):

- Chief System Requirements Engineer
- System Requirements Engineers
- Subsystem Requirements Engineers

System Architecture Team (Architects):

- Chief System Architect
- System Architects
- Subsystem Architects

Phase 1) SAI – Lessons Learned$_1$

Ensure Appropriate Team Memberships (e.g., Authority)

Ensure Adequate Resources (e.g., Staffing, Budget, and Schedule)

Obtain Consensus on:

- Assessment Objectives and Scope
- Definitions (e.g., of Quality Factors, Subfactors, and Cases)

Provide Early Training:

- Method Training
  (QUASAR, Requirements Engineering, and Architecting)
- System/Subsystem Training
  (Requirements and Architecture)

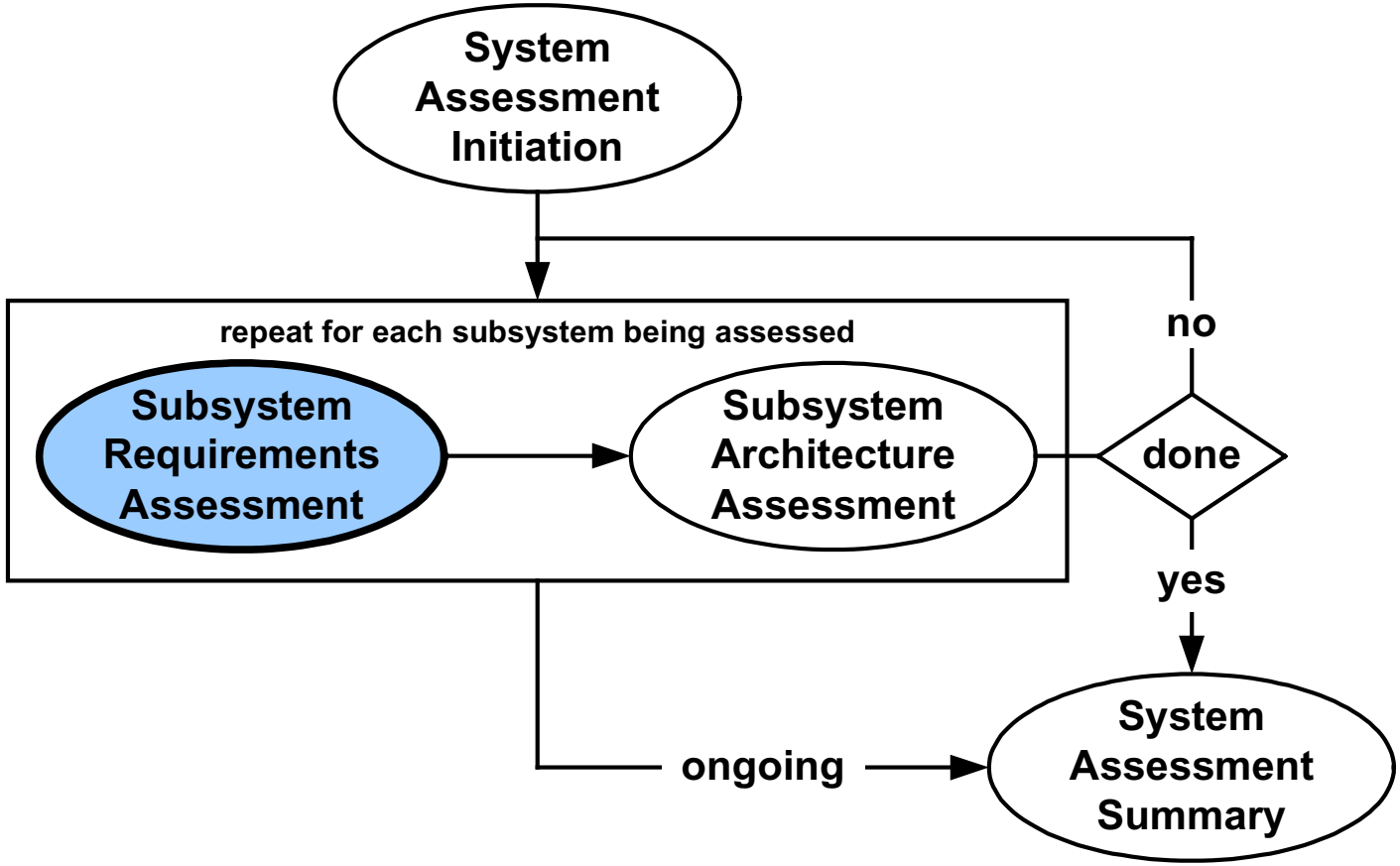
Phase 1) SAI – Lessons Learned$_2$

QUASAR assessments should be Organized according to a Quality Model that defines Quality Factors (a.k.a., attributes, “ilities’) and their Quality Subfactors such as:

- Availability
- Interoperability
- Performance
  - Jitter, Response Time, Schedulability, and Throughput
- Portability
- Reliability
- Safety
- Security
- Usability

Phase 2: Subsystem Requirements Assessment (SRA)

Subsystem Requirements Assessment (SRA)



Phase 2) SRA – Topics

System Requirements Assessment (SRA) Phase:

- Objectives
- Principles
- Challenges
- Tasks:
  - Preparation
  - Meeting
  - Follow-Through
- Primary Work Products
- Team Memberships
- Lessons Learned

Phase 2) SRA – Objectives

Assess Quality and Maturity of the Architecturally-Significant, Quality-Related, Subsystem Requirements including adequacy to:

- Drive the Subsystem Architecture
- Form Foundation for Subsystem Architecture Assessment

Ensure Subsystem Architecture Team will be Prepared to Support the coming Subsystem Architecture Quality Assessment

Produce and Publish Meeting Outbrief and Report

Manage Action Items

Capture Lessons Learned

Update QUASAR Method and associated Training Materials

Phase 2) SRA – Principles$_1$

Not all Requirements are Architecturally-Significant.

Quality-Related Requirements:

- Are typically Major Drivers of the System Architecture.
- Should be Unambiguous, Feasible, Complete, Consistent, Mandatory, Verifiable, Validatable, etc.
- Should *Not* Unnecessarily Constrain the Architecture.

Quality Requirements should Specify a Minimum Required Amount of some Quality Factor or Quality Subfactor.

Phase 2) SRA – Principles$_2$

Quality Requirements should be Organized according to a Quality Model that defines Quality Factors (a.k.a., attributes, “ilities’) and their Quality Subfactors such as:

- Availability
- Interoperability
- Performance
  - Jitter, Response Time, Schedulability, and Throughput
- Portability
- Reliability
- Safety
- Security
- Usability

Phase 2) SRA – Principles$_3$

Different Quality Factors are Important for Different Subsystems:

- Performance is Paramount for Real-Time Subsystems.
- Security is more Important for other Subsystems.

Engineering Quality Requirements requires Significant Effort and Resources (it *cannot* be accomplished during a short meeting).

Engineering Architecturally-Significant Requirements is the Responsibility of the *Requirements* Team –
Not the Architecture Team and *Not* the Assessment Team.

- Architects and Assessors are Not Qualified to Engineer Quality Requirements.
- Many Stakeholders have Different and Inconsistent Quality Needs.
- Architecture Assessment Time is Too Late to be Engineering Quality Requirements.

Phase 2) SRA – Challenges$_1$

Many Requirements Engineers are not taught how to Engineer Non-functional Requirements including Quality Requirements.

Although popular, Use Case Modeling is not very Effective for Engineering *Quality* Requirements.

Quality Requirements often require the Input from Specialty Engineering Teams (e.g., Reliability, Safety, and Security), who are not often adequately involved during Requirements Engineering.

Quality Goals are often Mistakenly Specified as Quality Requirements.

The resulting Architecturally-Significant Requirements are typically:

- Incomplete
  (missing important Relevant Quality Factors and Subfactors)
- Of Poor Quality (lack important characteristics)

Phase 2) SRA – Challenges$_2$

The typical Quality of Derived and Allocated Architecturally-Significant Requirements is Poor:

- Requirements are often *Ambiguous.*
  - "The system shall be safe and secure."
- Requirements Rarely Specify *Thresholds* on relevant Quality Measurement Scales.
  - "The system shall have adequate availability."
- Requirements are often mutually *Inconsistent*.
  - Security vs. usability, performance vs. reliability.
- Many Requirements are *Infeasible* (or at least Impractical) if taken literally.
  - "The system shall be available 24/7 every day of the year."
  - "The system shall have 99.9999 reliability."

Phase 2) SRA – Challenges$_3$

Requirements are often Unstable.

Specialty Engineering Requirements (e.g., reliability, safety, security) are Often Documented Separately from the Functional Requirements.

The Architecturally-Significant Requirements are often *Improperly* Prioritized for Implementation.

The Requirements Engineers often do *Not* Understand how to Prepare for a Subsystem Assessment.

- Too busy
- Not trained
- No standards exist
- Bias against assessments/audits

Phase 2) SRA – Challenges$_4$

Managers believe Schedule Pressures do *Not* allow Time for Requirements Assessments.

Subsystem Requirements Engineers may Not Understand how to give the Assessment Team what they need to assess the Requirements:

- Claims
- Arguments including Decisions, Trade-Offs, Rationales, and Assumptions
- What is the proper Evidence?
  - Official program documentation
  - Not plans and procedures
  - Not hastily produced PowerPoint slides

Phase 2) SRA – Preparation Task

- Process/Training Team trains the Subsystem Requirements and Architecture Teams *significantly prior* to the SRA Meeting.
- Subsystem Requirements and Subsystem Architecture Teams provide Preparatory Materials to the Subsystem Assessment Team *significantly prior* to the SRA Meeting:
  - Summary Presentation Materials
  - Requirements Quality Cases (including electronic access to evidentiary materials)
  - Example of Planned Architectural Quality Case
- Subsystem Assessment Team reviews these Preparatory Materials *prior* to the SRA Meeting.

Phase 2) SRA – Meeting Task

- Subsystem Requirements Team presents:
  - System Overview
  - Requirements Overview
  - *Requirements* Quality Cases
- Subsystem Assessment Team assesses Quality and Maturity of Requirements:
  - Completeness of Quality Cases
  - Quality of Quality Cases
- Subsystem Architecture Team presents Example Architectural Quality Case
- Subsystem Assessment Team recommends Improvements
- Subsystem Assessment Team manages Action Items

Phase 2) SRA – Follow-Through Task

Subsystem Assessment Team:

- Develops Consensus Regarding Subsystem Requirements Quality
- Produces, Reviews, and Presents Meeting *Outbrief*
- Produces, Reviews, and Publishes Requirements Assessment *Report*
- Captures Lessons Learned
- Manages Action Items

Subsystem Requirements Team:

Addresses Risks Raised in Requirements Assessment Report

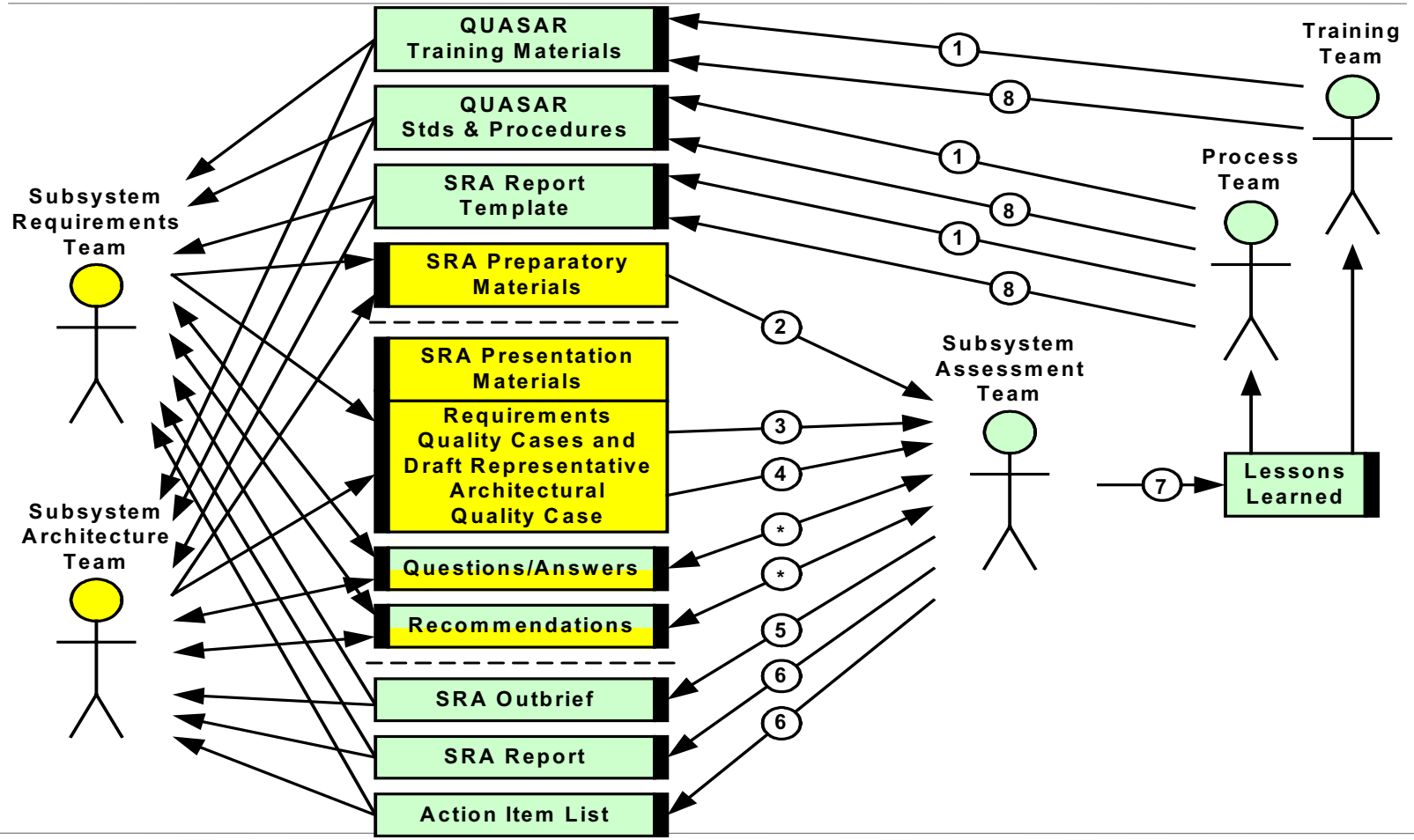
Process Team:

Updates Assessment Method (e.g., Standards and Procedures)

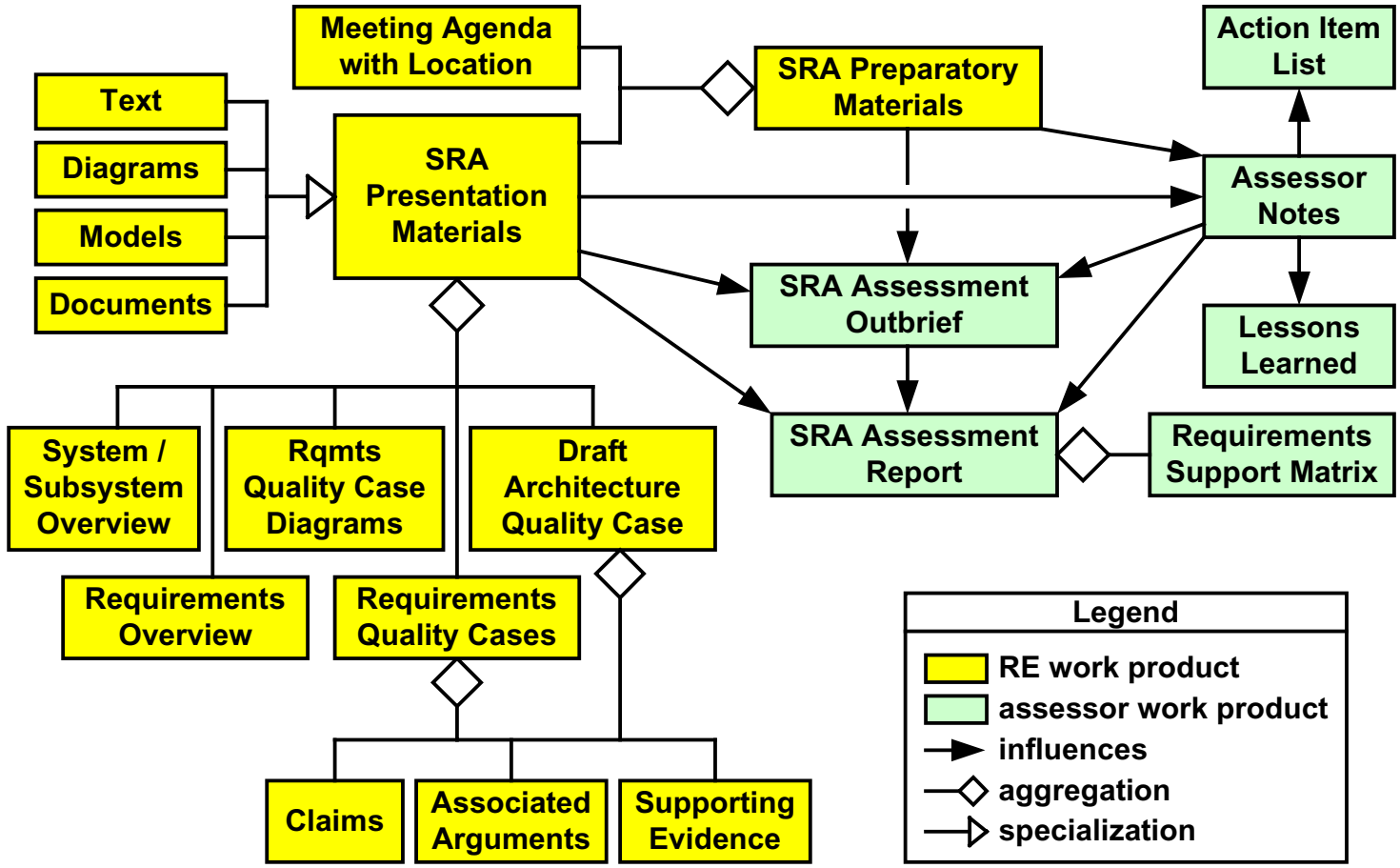
Training Team:

Updates Training Materials (if appropriate)

Phase 2) SRA – Work Product Workflow



Phase 2) SRA – Work Products



Phase 2) SRA – Team Memberships

Subsystem Requirements Team:

- Subsystem Requirements Engineers
- Subject Matter Experts (if appropriate):
  - Specialty Engineering Experts
  - Application Domain Experts

Subsystem Architecture Team:

- Subsystem Architects
- Subject Matter Experts (if appropriate):
  - Specialty Engineering Experts
  - Application Domain Experts

Phase 2) SRA – Assessment Team

Subsystem Assessment Team:

- Assessment Team Leader
- Meeting Facilitator
- Subsystem Liaisons
- Subject Matter Experts
- Acquisition *Observers*
- Scribe

Must include members having Experience and Expertise in:

- Requirements Engineering including Quality Requirements
- QUASAR (with all members having been trained in the method)
- Subsystem Application Domain(s)
  (e.g., avionics, sensors, telecommunications, or weapons)

Phase 2) SRA – Lessons Learned

Select, Define, and Prioritize Quality Factors and Quality Subfactors (e.g., as Critical, Important, Desirable, or Relevant).

Concentrate on Quality-related *Requirements*
(i.e., Merely Listing Quality Factors is Not Sufficient).

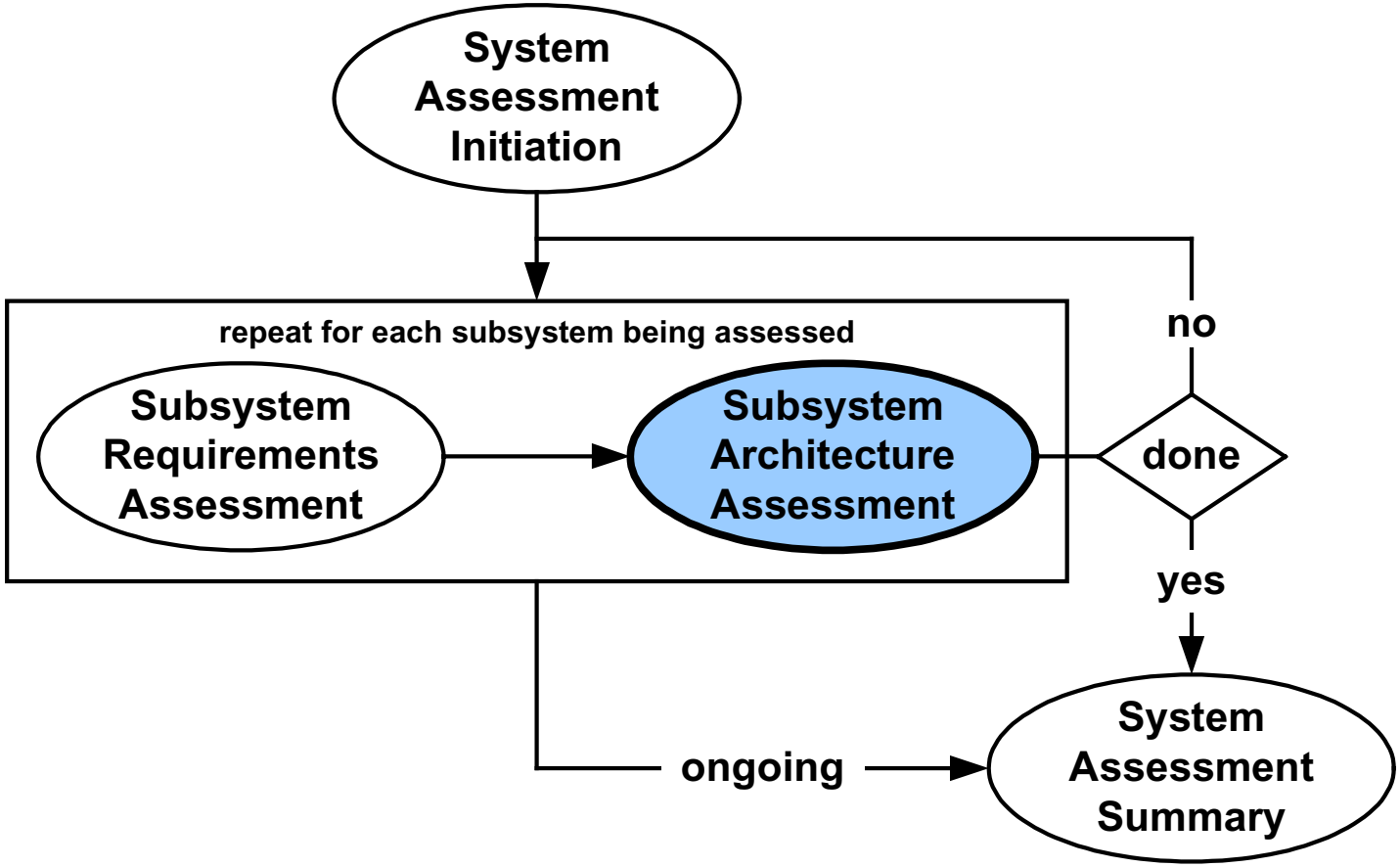
Architecturally-Significantly Quality Requirements must have certain Properties.

Iterative/Incremental Development implies Iterative/Incremental Requirements Assessments.

Hold Meeting Sufficiently Early for Quality Requirements to Drive the Architecture.

Phase 3: Subsystem Architecture Assessment (SAA)

Subsystem Architecture Assessment (SAA)



Phase 3) SAA – Topics

System Architecture Assessment (SAA) Phase:

- Objectives
- Principles
- Challenges
- Tasks:
  - Preparation
  - Meeting
  - Follow-Through
- Primary Work Products
- Team Memberships
- Lessons Learned

Phase 3) SAA – Objectives

Assess Quality of Subsystem Architecture in terms of:

- Architectures Support for its Derived and Allocated Architecturally-Significant Requirements
- Architectural Quality Cases

Phase 3) SAA – Principles

The Subsystem Architects should know:

- What Quality Goals and Requirements *drove* the Development of their Architectures.
- What Architectural Decisions they *made* and Why.
- Where they *documented* their Architectural Decisions.

The Subsystem Architects should already have Documented this Information as a *Natural* Part of their Architecting Method

Little *New* Documentation should be Necessary for the Subsystem Architects to make their Cases to the Subsystem Assessment Team.

The Subsystem Architects are Responsible for making their own Cases that their Architectures Sufficiently Support their Derived and Allocated Quality Requirements.

Phase 3) SAA – Challenges$_1$

Architects may not have developed Quality Cases as a Natural Part of their Architecting Process:

- Architectural Documentation are typically not organized by Quality Factors.
- Quality Case Evidence is often buried in and scattered throughout massive amounts of architectural documentation.
- Architectural Models (e.g., UML) often do *not* address Support for Quality Requirements.

Architecture Assessments may not be:

- Mandated by Contract or Development Process
- Scheduled and Funded

Phase 3) SAA – Challenges$_2$

Managers feel Schedule Pressures do *not* allow time for Architecture Assessments.

Architects often do *not* understand how to prepare for a Subsystem Assessment:

- Too Busy
- Not Trained
- No Standards Exist
- Bias against Assessments/Audits

Architecturally-Significant Requirements are Rarely Well Engineered.

Architectural Documentation often varies Widely in Quality and Completeness.

Phase 3) SAA – Challenges$_3$

Architecturally-Significant Requirements (esp. Quality Requirements) are *rarely traced* to the Architectural Elements that Collaborate to Implement Them.

Architectures are *rarely assessed* to Determine if they *truly* meet their Poorly-Specified Architecturally-Significant Requirements.

Phase 3) SAA – Preparation Task

- Subsystem Assessment Team provides Assessment Checklist
- Subsystem Architecture Team gathers (generates) and makes Available *Preparatory* Materials:
  - Subsystem Architecture Overview
  - Updated Quality Requirements
  - Quality Cases including Claims, Arguments, and Evidence
- Subsystem Architecture Team gathers (generates) and makes Available *Presentation* Materials
- Subsystem Assessment Team:
  - Reads Materials
  - Generates RFIs and RFAs
- Teams collaborate to Organize Assessment Meeting (Attendees, Time, Location, Agenda, Invitation)

Phase 3) SAA – Meeting Task

- Subsystem Architecture Team:
  - Introduces Subsystem Architecture
    (e.g., Purpose, Location, Context, and Major Functions)
  - Briefly reviews Architecturally-Significant Requirements
  - Briefly introduces Subsystem Architecture
    (e.g., Most Important Architectural Components, Relationships, Decisions, Mechanisms, Trade-Offs, and Assumptions)
  - Present Architectural Quality Cases
    (i.e., Claims, Arguments, and Evidence)
- Subsystem Assessment Team:
  - Probes Architecture (Architectural Quality Case by Quality Case)
  - Manages Action Items

Phase 3) SAA – Follow-Through Task

Subsystem Assessment Team:

- Develops Consensus regarding Subsystem Architecture Quality
- Produces, reviews, and presents Meeting Outbrief
- Produces, reviews, and publishes Architecture Assessment Report
- Captures Lessons Learned
- Manages Action Items

Subsystem Architecture Team:

Addresses Risks Raised in Architecture Assessment Report

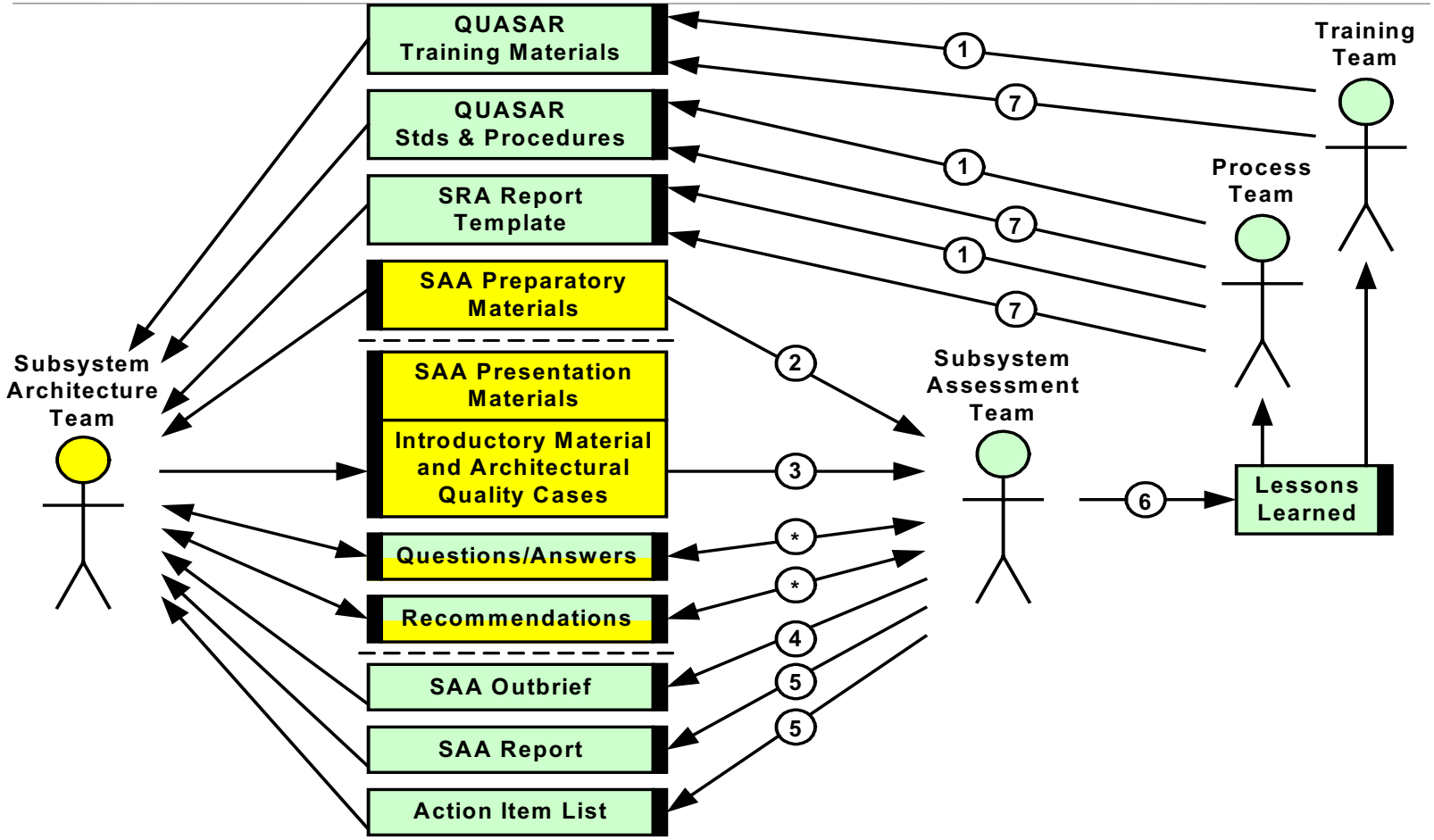
Process Team:

Updates Assessment Method (e.g., Standards and Procedures)

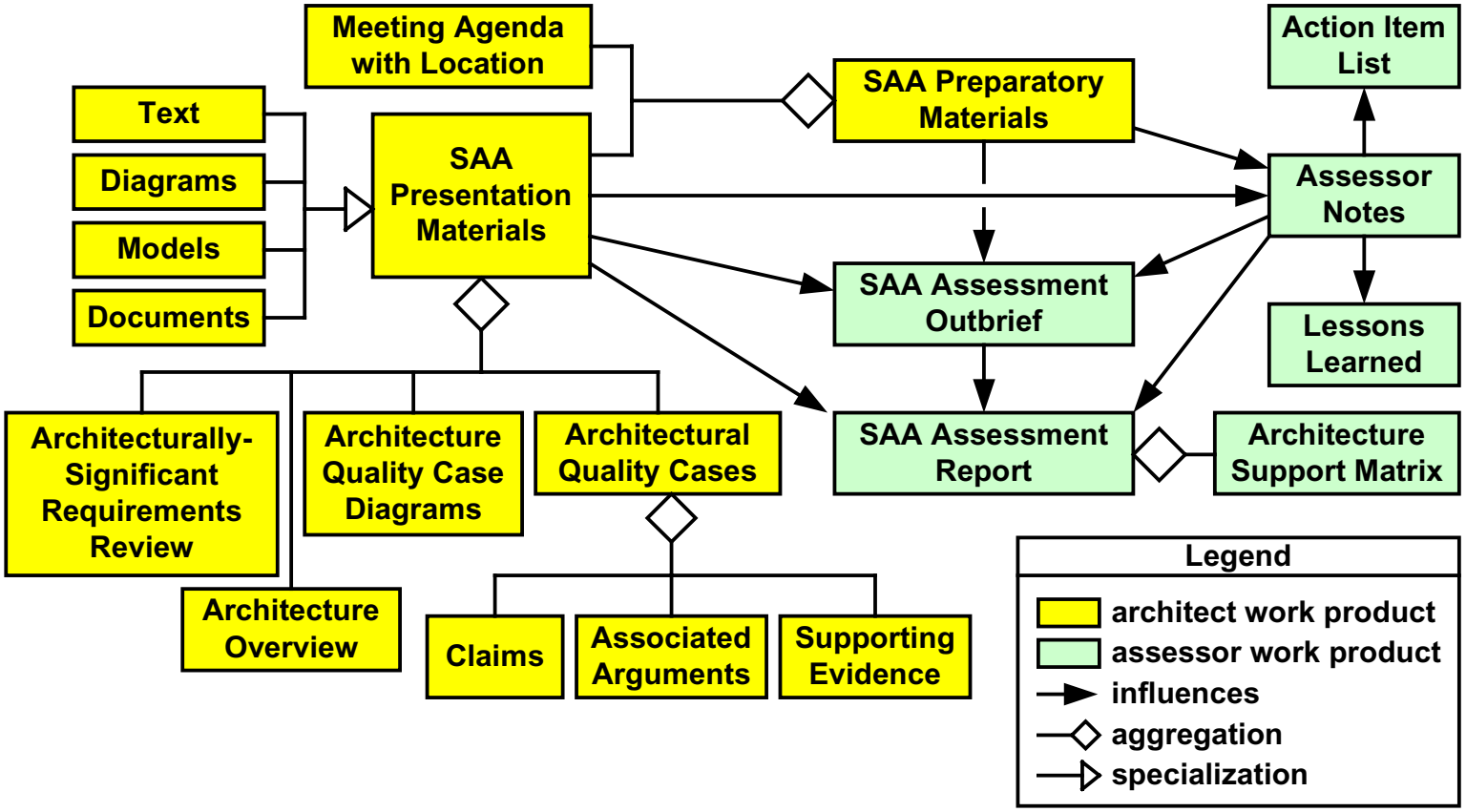
Training Team:

Updates Training Materials (if appropriate)

Phase 3) SAA – Work Product Workflow



Phase 3) SAA – Primary Work Products



Phase 3) SAA – Team Memberships

Subsystem Architecture Team:

- Subsystem Architects
- Subject Matter Experts (if appropriate):
  - Specialty Engineering Experts
  - Application Domain Experts

Subsystem Assessment Team

Subsystem Assessment Team:

- Assessment Team Leader
- Meeting Facilitator
- Subsystem Liaisons
- Subject Matter Experts
- Scribe

Must include members having Experience and Expertise in:

- System Architecting and System Architectures
- QUASAR (with all Members having been trained in the Method)
- Subsystem Application Domain(s) such as Avionics, Sensors, Telecommunications, or Weapons

Phase 2) SAA – Lessons Learned$_1$

Iterative/Incremental Development implies Iterative/Incremental Architecture Assessments.

Provide Initial Overview of Subsystem Architecture:

- Keep Overview Short
- Present Most Important Architectural Decisions, Trade-Offs between Quality Factors and Subfactors, and Assumptions
- Mount Diagrams on Meeting Room Walls (and Leave Them Up!)
- Highlight Primary Architectural Decisions

Focus on Assessing the Existing Architecture

Avoid a “Trust Me” Mentality

Phase 2) SAA – Lessons Learned$_2$

Organize Presentation by:

- Quality Factors and Quality Subfactors
- Architectural Components within the Subsystem

Do Not Restrict Evidence to Scenarios.

Present Both Logical (Functional) Architecture and Physical Architectural Structure.

Keep Evidence Presented and Requested within Assessment Scope.

Ensure Availability of Actual Architects.

Architects must have Electronic Access to Evidence to present Existing, Official Documentary Evidence.

Phase 2) SAA – Lessons Learned$_3$

Take Development Cycle, Project Schedule, and Architectural Maturity into Account.

Emphasize Assessment Results over Recommending Architectural Improvements.

Ensure Reasonable Assessment Size and Schedule.

Ensure Adequate Pre-Meeting Preparation.

All Architectural Tiers are not Equal:

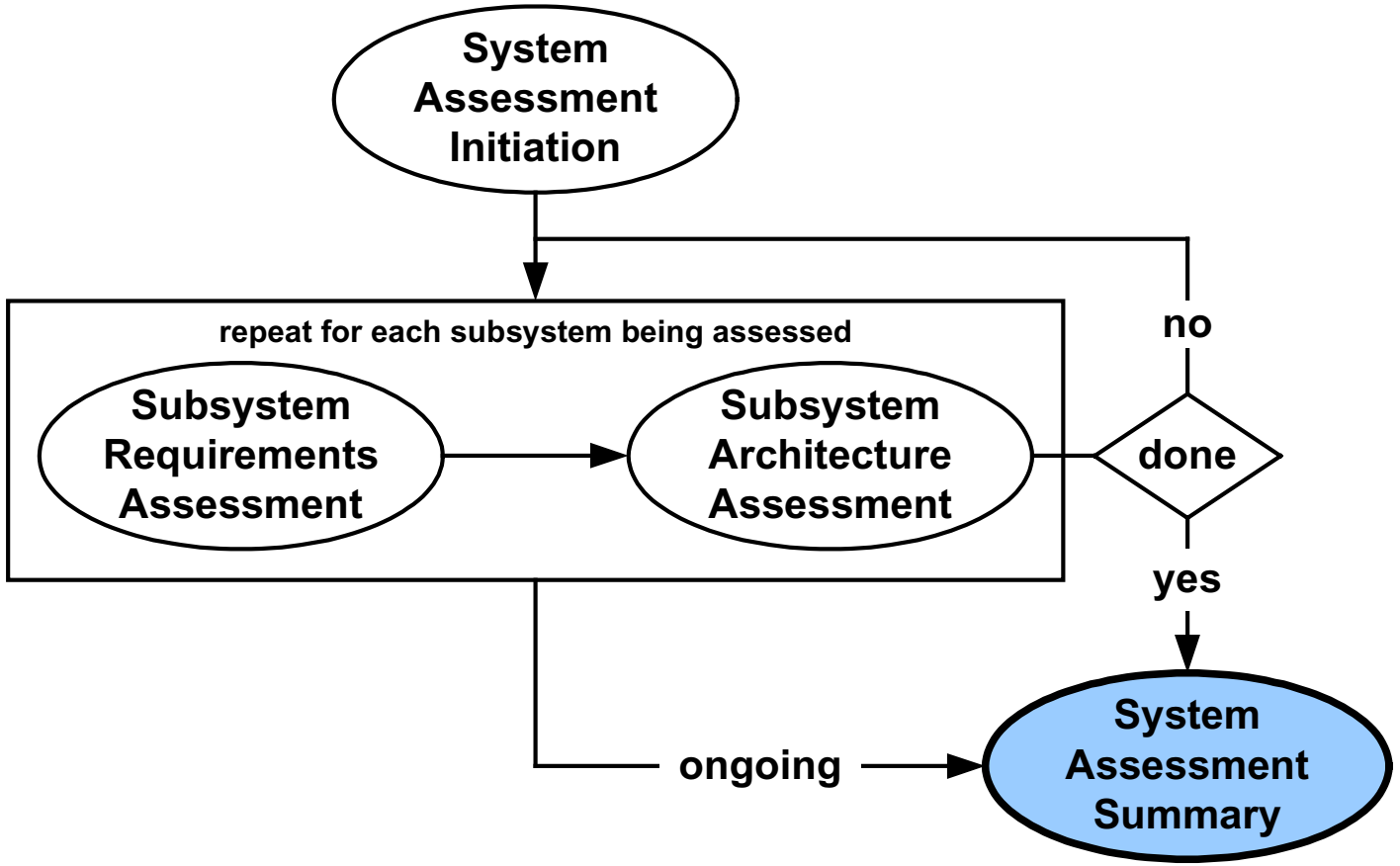
- Size, Complexity, Criticality, and Quality Factors/Subfactors
- Apply Different Emphasis at Different Levels of the Hierarchy.

Differentiate Architecture from Design.

Use Scenarios for Testing rather than introducing the Architecture.

Phase 4:
System Assessment Summary (SAS)

System Assessment Summary (SAS)



Phase 4) SAS – Topics

System Assessment Summary (SAS) Phase:

- Objectives
- Principles
- Challenges
- Tasks:
  - Preparation
  - Meeting
  - Follow-Through
- Primary Work Products
- Team Memberships
- Lessons Learned

Phase 4) SAS – Objectives

Collect previous Subsystem Architecture Assessment Results

Create System Architecture Assessment Summary Results

Capture Method Lessons Learned

Update Assessment Method and Training Materials

Phase 4) SAS – Principles

All Subsystems are Not Equally Important.

All Quality Factors and Subfactors are Not Equally Important for Different Subsystems.

Different Stakeholders want Different Summaries.

Phase 4) SAS – Challenges

How should Subsystem Findings be Summarized without ending up Comparing Apples and Oranges?

- Across Subsystems:
  - Average Subsystem Quality
  - Worst Subsystem Quality
  - Union of Subsystem Qualities (i.e., show all subsystems)
- By Quality Factor and Quality Subfactor:
  - Average Value
  - Worst Value
  - Union of Values (i.e., show all values)

Executive management may Demand Simplistic Single Number Summary of System Requirements and Architecture Quality.

Phase 4) SAS – Preparation Task

System Assessment Team:

- Collects Subsystem Assessment Results
- Summarizes Subsystem Assessment Results
  - Develops Subsystem Support Matrix
- Identifies Primary Stakeholders
- Produces, Reviews, and Distributes:
  - System Quality Assessment Summary Report
  - Preparatory Materials
  - Meeting Agenda
- Organizes Meeting

Phase 4) SAS – Meeting Task

System Assessment Team:

- Restates Assessment Objectives
- Summarizes QUASAR Method
- Summarizes Assessment Scope
- Summarizes Quality of System and Subsystem *Requirements*
- Summarizes Quality of System and Subsystem *Architectures*
- Solicits Feedback

System Requirements and Architecture Teams:

Provide Feedback

Phase 4) SAS – Follow-Through Task

System Assessment Team:

- Develops Consensus regarding System Requirements and Architecture Quality
- Produces, reviews, and publishes System Assessment Report
- Captures Lessons Learned
- Manages Action Items

System Requirements and Architecture Teams:

Address Risks Raised in System Assessment Report

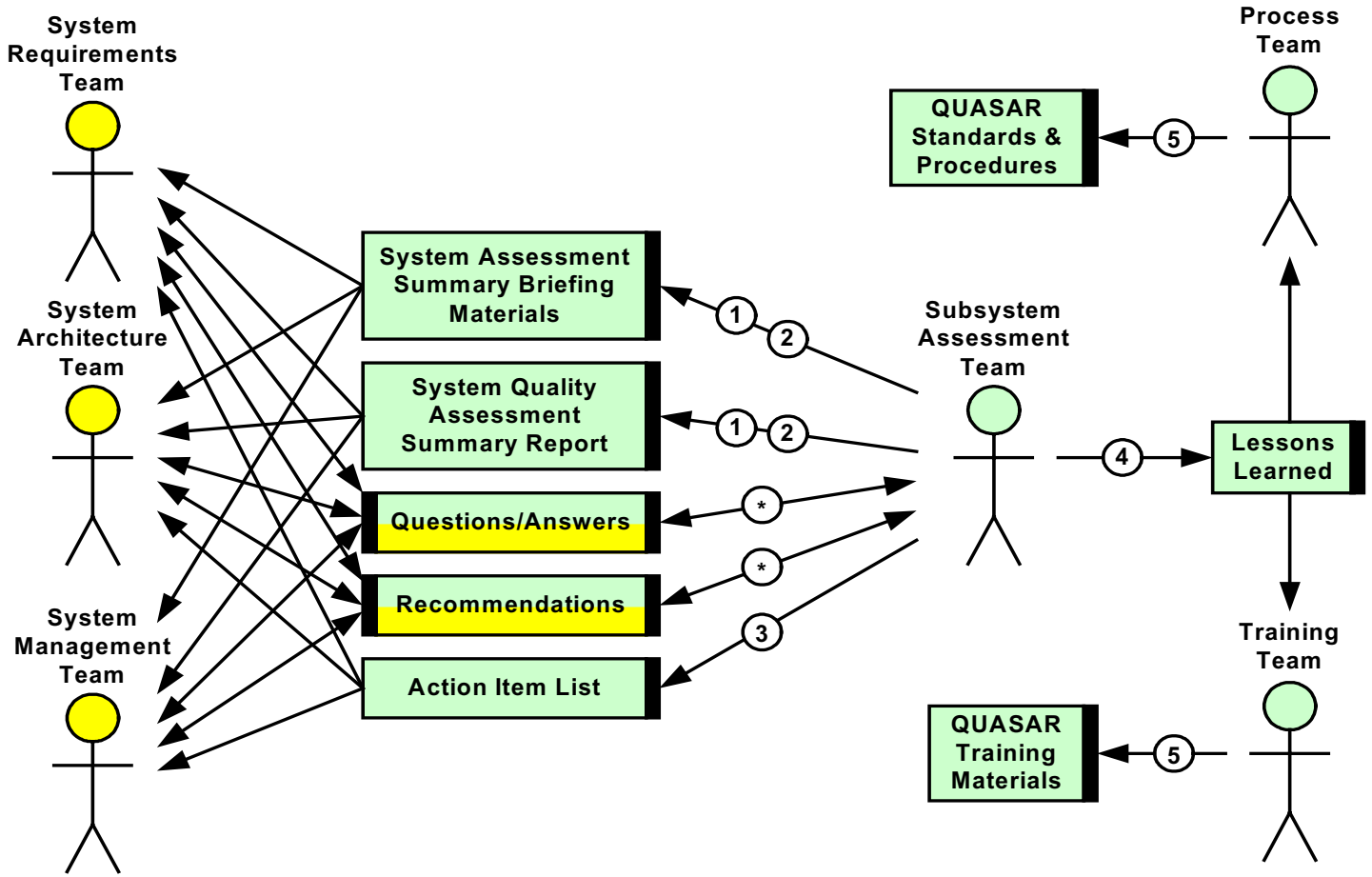
Process Team:

Updates Assessment Method (e.g., Standards and Procedures)

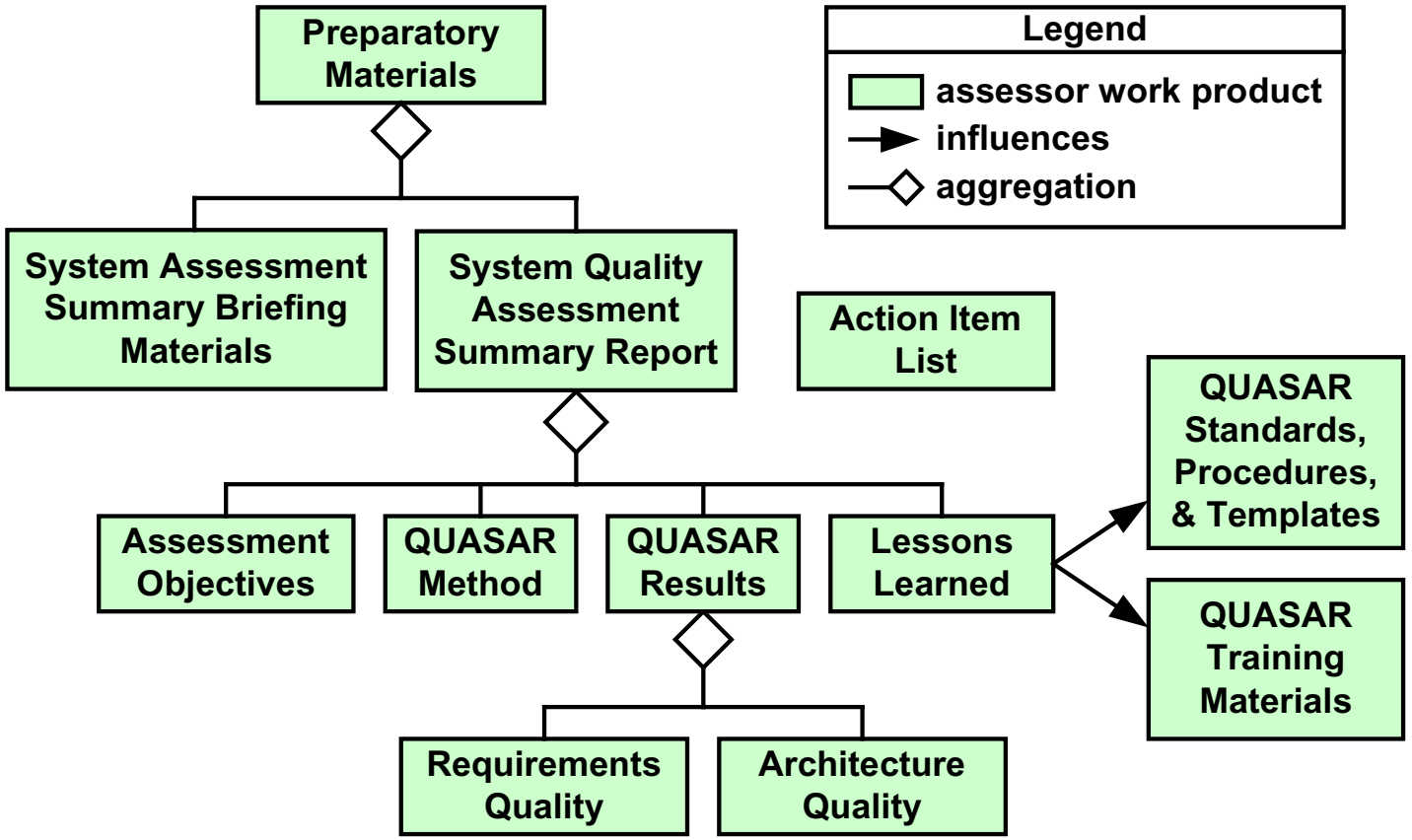
Training Team:

Updates Training Materials (if appropriate)

Phase 4) SAS – Work Product Workflow



Phase 4) SAS – Work Products



Phase 3) SAS – Team Memberships

System Requirements Team (Requirements Engineers):

- System Chief Requirements Engineer
- System Requirements Engineers
- Subsystem Requirements Engineers

System Architecture Team (Architects):

- System Chief Architect
- System Architects
- Subsystem Architects

System Management Team:

- System Program Manager
- System Technical Leader

Phase 3) SAS – Team Memberships

System Assessment Team:

- Assessment Team Leader
- Meeting Facilitator
- Subsystem Liaisons
- Subject Matter Experts
- Scribe

Phase 4) SAS – Lessons Learned

A Single Overall Summary Assessment Result can be Overly Simplistic.

Identify Current Problem/Risk Areas so that they can be Fixed.

System Assessment Summation should probably be *Ongoing* as part of an Incremental, Iterative Development Cycle.

QUASAR Benefits:

What you can expect to gain

QUASAR Benefits$_1$

Provides Acquirer Visibility into (and supports oversight of) the Quality of the Requirements and Architecture

Supports Certification and Accreditation

QUASAR Benefits$_2$

Supports Process Improvement:

- Solves Major Requirements and Architecture Problems

Provides Flexibility:

- Any Effective Requirements Engineering and Architecting Methods
- Uses Existing Requirements and Architecture Work Products
  (i.e., almost no new work products required)
- Any Subsystems based in Need and Risk
  (i.e., fits any system size, budget, schedule, and tier)
- Any Quality Factors and Quality Subfactors

QUASAR:

Today and Tomorrow

QUASAR Today

In-use on Largest DoD Acquisition Program

QUASAR Version 1 Handbook Published
http://www.sei.cmu.edu/publications/documents/06.reports/06hb001.html

Provided as SEI Service by Acquisition Support Program (ASP)

Tutorials at Conferences

QUASAR Handbook

Intended Audiences:

- Acquisition Personnel
- Developers (Architects and Requirements Engineers)
- Subject Matter Experts (domain, specialty engineering)
- Consultants
- Trainers

Objectives:

- Completely Document the QUASAR Method (Version 1)
- Enable Readers to start using QUASAR

Description:

- *Very* Complete
- Too Comprehensive to be Good First Introduction

QUASAR Tomorrow – Technical Plans

Quality Factors across Multiple Subsystems:

- Multiple Cross-Cutting Structures and Models
- Multiple Subsystems Collaborate to Achieve Quality Requirements

Development of Catalog of Quality Factor-Specific Architectural Styles, Patterns, and Mechanisms to use as Standardized Quality Case Arguments

Improve Objective Determination of “Sufficient Quality”

Expand Quality Cases Beyond Requirements and Architecture

QUASAR Tomorrow - Productization

More Conference Tutorials and Classes

Expanded QUASAR Training Materials

QUASAR Articles

Use and Validation on more Programs

QUASAR Book

How the SEI Can Help You

QUASAR is Ready for Use *Now*.

QUASAR Handbook and Training Materials can be downloaded from SEI Website.

The SEI Acquisition Support Program (ASP) offers QUASAR as a Service:

- Consulting and Training
- Facilitation of QUASAR Assessments
- Recommended RFP and Contract Language

Contact Information

For more information, contact:

Donald Firesmith

Acquisition Support Program

Software Engineering Institute

dgf@sei.cmu.edu

Software Engineering Institute | Carnegie Mellon