The Journal of Systems and Software 85 (2012) 1511–1519

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Scaling up software architecture analysis

Rick Kazman [a,b,*], Michael Gagliardi [b], William Wood [b]

[a] Dept. of Information Technology Management, University of Hawaii, 2404 Maile Way, Honolulu, HI 96825, USA
[b] Software Engineering Institute, Carnegie Mellon University, 4500 5th Ave., Pittsburgh, PA 15213, USA

ARTICLE INFO

*Article history:*
Received 30 April 2010
Received in revised form
24 November 2010
Accepted 16 March 2011
Available online 2 April 2011

*Keywords:*
Software architecture
Analysis
System of systems
Software-intensive ecosystems

ABSTRACT

This paper will show how architecture design and analysis techniques rest on a small number of foundational principles. We will show how those principles have been instantiated as a core set of techniques. These techniques, combined together, have resulted in several highly successful architecture analysis and design methods. Finally, we will show how these foundations, and the techniques that instantiate them, can be re-combined for new purposes addressing problems of ever-increasing scale, specifically: addressing the highly complex problems of analyzing software-intensive ecosystems.



## 1. Introduction: the value proposition for architecture

Many large software-intensive system failures can be traced to one of three root causes: (1) inadequate software architecture education, (2) inadequate software architecture practice, or (3) no meaningful software architecture evaluation early in the system development life cycle (Bass et al., 2003; Clements et al., 2002). In each case, the reasons for the failure were avoidable. Using architecture-centric practices throughout the lifecycle and lifetime of a system leads to a number of important benefits:

- early identification of important product qualities
- predictable product quality achievement supporting the business and mission goals, which translates into competitive advantage
- early identification and mitigation of design risks resulting in fewer downstream problems
- cost savings in integration and test, and in evolution

In this paper we will show how architecture-centric practices and, in particular, architecture analysis can be employed in a variety of ways to find and address risks early in a system's life cycle. We, and others in the software architecture community, have created a number of methods for the design, documentation, and analysis of software and architectures over the past 10 years. These methods share many of the same underlying principles and are realized in practice by a small set of techniques. The point of this paper is to name, illustrate, and illuminate these 7 principles and show how these principles – while they were initially created to aid in the design, analysis and construction of single systems – in fact apply to systems of *all* scales, from single-system software architectures to system architectures to system-of-system (SoS) architectures, which are kinds of software ecosystems.

Some researchers (e.g. Bosch, 2009) have defined a software ecosystem as a core platform with a surrounding community of developers who create value by building applications on top of this platform. We take a slightly different view of software ecosystem. We define it, following (Messerschmitt and Szyperski, 2003) as a set of independent but related organizations that function as a unit, with a shared market for software, systems, and services. Because our definition includes inter-organizational concerns, we must address ecosystems as socio-technical constructs. Thus the methods for dealing with these ecosystems must consider technical concerns, business concerns, as well as social and organizational issues. The consistent consideration of such concerns has led to the development of our core principles.

These principles are:

1. The software architecture of a system is the fundamental artifact that guides development (Bass et al., 2003; Bengtsson et al., 2004; Hofmeister et al., 1999)
2. Systems are built to satisfy business goals (Bengtsson et al., 2004; Hofmeister et al., 1999; ISO/IEC, 2001)

* Corresponding author at: Software Engineering Institute, 4500 5th Ave., Pittsburgh, PA 15221, USA. Tel.: +1 412 268 1588; fax: +1 412 268 5758.
*E-mail addresses:* kazman@sei.cmu.edu (R. Kazman), mjg@sei.cmu.edu (M. Gagliardi), wgw@sei.cmu.edu (W. Wood).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.03.050

3. Architecture design is based on a set of architecturally significant requirements, derived from business goals (de Boer and van Vliet, 2009; ISO/IEC, 2001; Kazman and Bass, 2005)
4. Quality attribute requirements exert the strongest influence on architectural design (Bass et al., 2003; de Boer and van Vliet, 2009; Hofmeister et al., 2007; van Lamsweerde, 2001)
5. Architecture design can be made tractable by considering a small number of primitives, called tactics (Bachmann et al., 2007; Bass et al., 2003; Scott and Kazman, 2009)
6. Architecture design can and should be guided by analysis (Bengtsson et al., 2004; Clements et al., 2002; Dobrica and Niemela, 2002)
7. Architectures are developed by people within an organizational and business context; so economic and organizational concerns shape and constrain architecture (Paulish, 2002)

## 2. Implications for software architecture design and analysis

The degree to which a system meets its quality attribute requirements is dependent on architectural decisions. As a consequence these quality attributes should be designed into the architecture. To be effective, the architecture-centric design and analysis activities must: directly link to business and mission goals, explicitly focus on quality attributes, and explicitly involve system stakeholders. This sounds obvious. But how do we ensure this occurs in a predictable, repeatable manner? Repeatability and predictability can only come with documented methods and through training in the proper use of those methods.

Thus we have developed, refined, and industrially tested a set of methods and techniques[1] for aiding in the achievement of each of these objectives.

### 2.1. Identifying and prioritizing business goals

The first task in the creation of any system, however, complex, is to identify the goals of that system. However these goals are often implicit, held in the minds of the stakeholders. The consequence of having not explicitly recorded and prioritized the business goals is that they are frequently fuzzy and open to interpretation and endless debate. Similarly, if the business goals have not been prioritized, much effort may be wasted simply by having different members of the project operating under different assumptions of what is "important". When asked to identify the business goals of a system, project leaders often describe a set of functions that the system needs to provide, and a set of qualities that it needs to exhibit (e.g. the system should be easy to use, fast, reliable, easy to modify, etc.). When asked which of these are most important, the business leader will invariably reply "they are *all* most important!".

But engineering is inherently about tradeoffs, and so while the business leader might *want* to achieve all of these goals, the reality is that some will be subordinated to others by the decisions that the architects and implementers make. For this reason, eliciting and prioritizing the business goals is the foundation for a sound practice of architecture analysis, because everything else in the project should flow from, and be aligned with, these goals. This principle has long been a foundation of requirements engineering methods (e.g. van Lamsweerde, 2001 and Nuseibeh and Easterbrook, 2000) but is frequently overlooked in practice, by organizations doing architecture-centric development.

To achieve this we need to: identify the project's key stakeholders; identify the business goals of the stakeholders; and prioritize these business goals. Stakeholders often find expressing and/or prioritizing their business goals to be difficult. We have devised a number of techniques to facilitate this process. Mining architectural analyses, we have created a taxonomy of business goals (Kazman and Bass, 2005) to aid in stakeholder elicitation, as shown in Fig. 1.

Each of the categories shown in Fig. 1 is broken down into subcategories. For example "Reduce Total Cost of Ownership" is broken down into: reduce cost of development, reduce cost of deployment and operations, reduce cost of maintenance, and reduce cost of retirement/moving to a new system. These sub-categories are further dissected into sub-sub-categories. For example, "reduce cost of deployment and operations" is broken down into: ease of installation, and ease of repair.

Finally, these business goals must be *prioritized*. We typically ask project managers, system owners, customer representatives, and other important stakeholders to create this prioritization, using whatever prioritization technique they prefer. While this prioritized set of business goals provides a basis for all that follows, it is not sufficient to directly support construction and analysis; business goals, even when sub-sub-categorized, are still vague. To understand the implications of business goals on a project, as mediated by an architecture, we need to delve more deeply. We do this by aiding the ecosystem's stakeholders in translating from their unique (and sometimes parochial) business goals to a set of *architecturally significant requirements* that can be applied to the entire ecosystem.

### 2.2. Architecturally significant requirements

But where do we get architecturally significant requirements from? While these requirements must emanate from the system's stakeholders, it is fruitless to just ask ecosystem stakeholders what their requirements are: stakeholder thinking about architecturally significant requirements is typically fuzzy. And so we have, over the years, developed techniques to facilitate stakeholders in forming, validating, and prioritizing these requirements. We have packaged these techniques into a method that we call a *Quality Attribute Workshop* (QAW). The QAW is a facilitated method that engages system stakeholders early in the life cycle (Barbacci et al., 2003) to discover and prioritize the driving quality attribute requirements of a software-intensive system. And, as we will discuss in Section 3.2, the principles of the QAW can be scaled up to analyze the quality attribute requirements of ecosystems. This scaled-up version of the QAW is called a "Mission Thread Workshop".

The QAW is a system-centric, stakeholder focused, scenario based method that is intended to be used *before* the software architecture has been created. It results in a number of well-documented benefits: increased stakeholder communication and participation, clarified quality attribute requirements, and an informed basis for architectural design decisions. The heart of the method rests in carefully and unambiguously characterizing quality attributes.

Quality attribute names by themselves – performance, modifiability, security, testability, availability, and so forth (ISO/IEC, 2001) – are inadequate to represent anything but the most coarse architectural concern (e.g. "The system should be secure"). Heated debates often revolve around the quality attribute to which a particular system behavior belongs: one group of people might classify a denial of service attack as a security problem, while another might classify it as a performance problem. Such debates are, in the end, pointless. The problem lies in the fact that the vocabulary describing quality attributes varies widely, and stakeholder thinking about quality attributes is seldom carefully elicited and captured. Requirements documents are filled with requirements such as "The architecture shall be secure and robust".

A solution to the characteristic ambiguity found in attempts to describe quality attributes is to use *quality attribute scenarios* to

[1] In this paper we will use the convention that *methods* contain *techniques*.

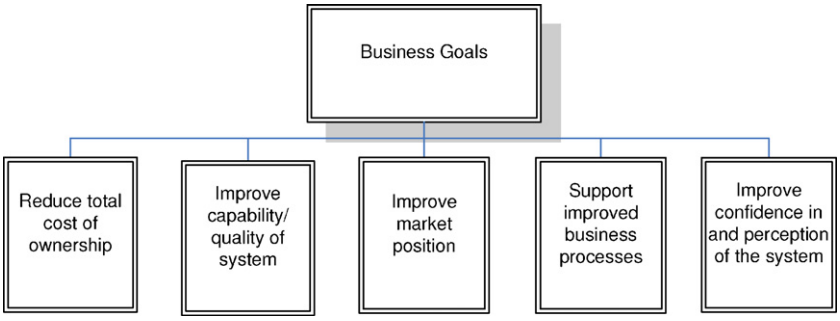


**Fig. 1.** First level of the business goals taxonomy.

better characterize them. A quality attribute scenario is a short description of how a system is required to respond to some stimulus. As such, it is a falsifiable hypothesis—the system either will or will not respond to the stimulus by producing the desired response.

Quality attribute scenarios are extremely useful because they are *architecture test cases*. They can be applied proactively, at analysis time, to determine the likelihood of a proposed architecture meeting its quality goals. Quality attribute scenarios are used in precisely this way in the Architecture Tradeoff Analysis Method (Clements et al., 2002). Or these scenarios can be used retrospectively, to determine if the system that has been produced (based on the specified architecture) actually meets its quality attribute requirements.

More formally, a quality attribute scenario consists of six parts:

1. *source*: an entity that generates a stimulus
2. *stimulus*: a condition that affects the system
3. *artifact*: the part of the system that was stimulated by the stimulus
4. *environment*: the condition under which the stimulus occurred
5. *response*: the activity that results because of the stimulus
6. *response measure*: the measure by which the system's response will be evaluated

Quality attribute scenarios, thus captured and prioritized, form the backbone of everything that we do in architecture analysis and design. They define what it means for an architecture to be successful. But these scenarios are simply a more precise statement of the drivers; the architecturally significant requirements. How do we link these drivers to design decisions?

### 2.3. Tactics

Every ecosystem has an architecture, or more likely a set of architectures, at its core. Such architectures must be created by humans—software, system, and SoS *architects*. The core of the architecture design process consists of the architect generating a hypothesis and then testing that hypothesis against some criteria. If the hypothesis does not pass the test, another hypothesis is generated and tested. We have just described how the criteria may be unambiguously described as quality attribute scenarios. But where do the architect's design hypotheses come from? Or, to put it another way, faced with the blank page, how does an architect know what to design and where to begin?

We propose that the architectural design of a single system, or an ecosystem, begins with *tactics*. A tactic is a design decision that is influential in the control of a quality attribute response. Tactics are scale-invariant design primitives that have a role in the generation of hypotheses and the validation of a design against some assessment criteria. Each tactic is a single design option for the architect. For example, consider the set of availability tactics (Scott and Kazman, 2009) shown in Fig. 2.

For example, to promote availability, an architect might choose a Fault Detection tactic such as Ping/Echo, and Fault Repair tactic such as Active or Passive Redundancy, and a Reintroduction tactic such as Shadow operation. Such a tactic might be used within a single system (for example, a server might have a hot spare—an instance of Active Redundancy) or between multiple systems in an ecosystem (system A might provide similar functionality to system B and it can be activated in case system B fails, providing Passive Redundancy). (These tactics and their implications are discussed in greater detail in (Scott and Kazman, 2009).)

Architectural patterns are packages of tactics. For example, a pattern that supports availability will typically use some form of a Redundancy tactic to achieve higher availability. Consider the Triple Modular Redundancy pattern (Storey, 1996). This can be understood as a combination of Active Redundancy and Voting tactics. Similarly, the Layers pattern is composed from several distinct Modifiability tactics: Maintain Semantic Coherence, Abstract Common Services, Raise Abstract Level, Use Encapsulation, Restrict Communication Paths, and Use an Intermediary (Bachmann et al., 2007).

Tactics categorizations exist for the following six quality attributes categories—availability, modifiability, performance, security, testability, and usability (Bass et al., 2003). Tactics are not just a theoretical construct: they have substantial practical application and have formed the basis for architecture design methods and tools (e.g. van Lamsweerde, 2001) that have been industrially adopted.

The choosing and documenting of a set of tactics and patterns are the primary activities of architecture design. But design and anal-

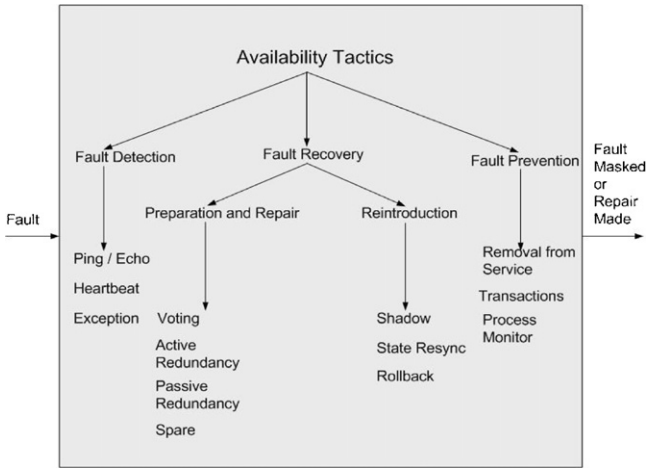


**Fig. 2.** Availability tactics.

ysis are two sides of the same coin. To validate a design, it must be analyzed and evaluated. And the techniques that lead us to choose a design are exactly the same as those that inform analysis. In the remainder of this paper we will show how the "Principles of Software Architecture" – and their realizations as a set of techniques – have been combined into several evaluation methods, such as the ATAM (Architecture Tradeoff Analysis Method). But, perhaps more significantly, we will show how these principles can be recombined into new techniques with different scales and scopes: a *Mission Thread Workshop*, a *System ATAM*, and a *System-of-Systems Architecture Evaluation*.

## 3. Relation to architecture evaluation

Why do we evaluate architectures? Quite simply: because so much depends on the architecture. An unsuitable architecture will precipitate disaster in a project. And architecture determines the structure of the project. Documented, repeatable methods provide a low-cost risk mitigation activity that can be employed early in the software development life cycle. It is becoming increasingly common that architecture evaluation is a standard part of any methodology for developing large complex systems.

We have been analyzing architectures using the ATAM for over 10 years (it has been used in countless evaluations by major companies and government organizations such as Boeing, Raytheon, GM, Ford, the US Army, Siemens, Fidelity Investments, Bosch, Pitney-Bowes, HP, General Dynamics, Philips, Visteon, Wells Fargo, UPS, Daimler, Mellon Financial, . . .). The stated purpose of the ATAM is *to assess the consequences of architectural decisions in light of quality attribute requirements and business goals*. To do this the ATAM brings together three key groups in an evaluation: a trained evaluation team; an architecture's senior decision makers (the architect, senior designers, the project manager); and representatives of the architecture's stakeholders.

The ATAM leaders cannot hope to be experts on the details of the architectures that they examine. Hence their purpose is to help stakeholders ask the right questions, and to use these questions to guide investigations that may lead to the discovery of potentially problematic architectural decisions. The discovered risks can then be made the focus of risk mitigation activities such as further design, further analysis, and prototyping.

### 3.1. ATAM led to the development of other methods

The ATAM has been discussed and documented thoroughly elsewhere (Bass et al., 2003; Clements et al., 2002). Our intention here is not to repeat that discussion, but rather to show how the 7 fundamental principles behind the ATAM has not only motivated everything presented thus far but has been extended to larger scales and scopes to create three new methods, namely:

- *Mission Thread Workshop (MTW)*: Augment existing end-to-end mission threads with quality attribute considerations that shape the SoS ecosystem architecture, with inputs from key SoS stakeholders. Identify SoS architectural challenges as well as capability and engineering issues at the SoS level, early in the SoS life cycle.
- *System ATAM*: Expanded the scope of the ATAM to address system architecture issues. Ability to evaluate a system and software architecture early in its life cycle. In an SoS ecosystem context, this can be used to identify architectural risks within individual systems of an ecosystem architecture.
- *SoS Architecture Evaluation*: a "first pass" identification of architectural risks within the SoS ecosystem and across the individual systems using mission threads augmented with quality attribute concerns. This analysis may be profitably undertaken early in the SoS life cycle in that it is relatively inexpensive to perform and it can find risks prior to deployment of major portions of the ecosystem.

The relationships between these methods are illustrated in Fig. 3.

#### 3.1.1. Background

Quality attributes are the key to mission success. In addition to providing needed capabilities all systems, and all ecosystems, must satisfy quality attributes that are essential to mission success and achievement of business goals. Examples of such quality attributes are performance, availability, reliability, security, usability, testability, safety, interoperability, maintainability, and spectrum management. If these quality attributes are not met, the ecosystem is of little value to its stakeholders.

Severe integration, interoperability, and operational problems can arise from inconsistencies, ambiguities, and omissions in addressing the quality attributes of SoS architectures. In many cases, the root causes of interoperability and integration problems in SoSs can be traced to a failure to address quality attributes in the SoS and its constituent system/software architectures. For example, in the US Department of Defense (DoD) SoS context, system use cases are common architecture design practices that are used to identify and develop required SoS capabilities. However, these practices do not adequately address cross-cutting quality attributes from end to end in a SoS. To be sure that SoSs will satisfy both their functional and quality attribute requirements, developers need to consider quality attributes in the context of end-to-end mission threads.

We have developed a SoS Architecture Engagement approach (depicted in Fig. 3) that builds upon the 7 fundamental principles of the ATAM to address: early elicitation of SoS quality attribute considerations; early identification and addressing of SoS architecture challenges (e.g. candidate legacy system architecture evaluation); early identification of candidate legacy system/software architectural risks for inclusion in the SoS architecture; and early identification and mitigation of SoS architectural risks, once the SoS architecture is sufficiently defined and documented. Each of the methods that comprise this Architecture Engagement will be described in the following subsections.

### 3.2. Mission Thread Workshop

The purpose of the Mission Thread Workshop (MTW) is to augment end-to-end mission threads with quality attribute considerations and to identify architecture, engineering, and capability challenges early in the definition of a system of systems architecture. A MTW brings together key stakeholders representing a variety of organizations, roles, and points of view. Together, these stakeholders augment end-to-end mission threads by considering and capturing quality attribute, mission, and capability needs. The information elicited in an MTW is then made available to the SoS architecture development, integration, and test activities. This information is used to: (1) determine a set of SoS architectural, capability, and engineering challenges, (2) inform SoS, system, and software architecture development and acquisition activities and (3) drive early risk reduction activities.

In the DoD a *vignette* is a description of the geography, force structure and mission, strategies and tactics, enemy forces and their attack strategies and tactics including timing. There may be associated measures of performance (MOP) and measures of effectiveness (MOE). A vignette provides context for one or more *end-to-end mission threads*. An end-to-end mission thread is defined as "a sequence of activities and events beginning with an opportunity to detect a threat or element that ought to be attacked and ending

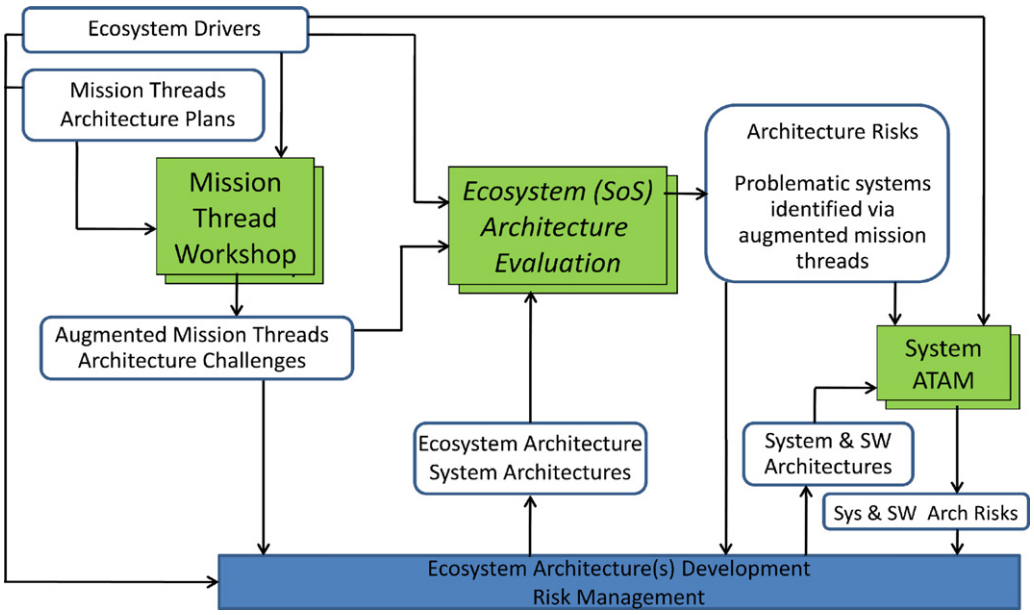


**Fig. 3.** ATAM-inspired architecture evaluation methods and their relationships.

with a commander's assessment of damage after an attack." (C4ISR, 2006). There are many other types of mission threads, e.g. developmental, sustainment. Mission threads are somewhat similar to use cases and user stories employed in traditional systems analysis and design methods, but they apply at the SoS level, spanning multiple constituent systems, and they are annotated with quality attribute information, like a scenario. A vignette from which we derive a mission thread is illustrated in Fig. 4.

Example vignette:

Two ships (Alpha and Beta) are assigned to integrated air and missile defense to protect a fleet containing two high-value assets (HVA). A surveillance aircraft (SA) and 4 un-manned air vehicles (UAV) are assigned to the fleet and controlled by the ships. Two UAVs flying as a constellation can provide fire-control quality tracks directly to the two ships. A three-pronged attack on the fleet occurs:

- 20 land-based ballistic missiles from the east
- 3 min later 5 aircraft-launched missiles from the south
- 5 min later 7 submarine-launched missiles from the west

The fleet is protected with no battle damage.

A mission thread is given as a series of steps, like a behavioral trace. An operational mission thread shows how the vignette, or part of the vignette, might "play out" among the systems that are participating in the SoS. The following is an example of an operational mission thread's steps that are derived from the vignette above:

| Step | Description |
|---|---|
| 1 | Alpha develops the Air defense plan and Rules of Engagement and sends them to Beta. The plan assigns to Alpha the Area of Regard (AOR) to the west, and Beta the AOR to the east. Alpha configures surveillance and weapons systems to support eastern engagements. |
| 2 | The SA aircraft detects that the 5 enemy aircraft have changed course and are heading towards the fleet at low altitude. |
| 3 | SA informs both Alpha and Beta of the change. |
| 4 | Alpha (and Beta) go to General Quarters. |
| 5 | SA detects that 5 missiles have launched from the southeast enemy aircraft and informs Alpha and Beta. |
| 6 | Alpha assigns its pair of eastern UAVs to track the missiles. |
| 7 | The 2 Alpha controlled UAVs send tracks for the 5 missiles to both Alpha and Beta. |
| 8 | Alpha assigns 3 missile engagements to itself and 2 to Beta. |
| 9 | Alpha receives tracks (continuously) for the missiles and initiates engagements for 3 missiles. |
| 10 | Alpha and Beta each launch 2 missiles. (Alpha and Beta must track the missiles.) |
| 11 | SA detects submarine launched 7 missiles from the southwest. |
| 12 | SA informs both Alpha and Beta. |
| 13 | Alpha commands Beta to use its 2 UAVs to track these submarine launched missiles. |
| 14 | Alpha launches a third missile against the aircraft initiated attack. |
| 15 | Alpha assigns itself 3 of the 7 missiles and Beta the other 4 for the submarine initiated attack. |
| 16 | Alpha and Beta each launch 2 missiles against the submarine attack. |
| 17 | Alpha analysis of its UAV track data shows that one aircraft launched missile was not killed. |
| 18 | Alpha picks up the "leaker" missile on its own radar. |
| 19 | Alpha launches another missile at the "leaker". |
| 20 | Ships radar confirms that the "leaker" was killed. |
| 21 | Alpha and Beta launch remaining missiles at submarine launched targets. |
| 22 | Kill Assessment Performed—all launched missiles are destroyed. |

The above example is DoD-specific. But mission threads occur in all complex systems. For example, in commercial systems missions threads are realizations of business processes that cross inter-organizational and inter-firm boundaries, linking supply chains or sets of services.

### 3.2.1. MTW phases

A MTW has three phases: preparation, augmentation, and roll-up/follow-up.

In the *preparation phase* (realizing principles 1–3), analysts meet with the SoS architect and program or project manager to determine the vignettes and mission threads to be developed and augmented; reach agreement on scope and series of MTWs; identify stakeholders; and work out the logistics. We have found the following criteria useful for the development and selection of vignettes and mission threads: capability coverage; new requirements or capabilities; stresses on the SoS, touching all constituent systems and communications.

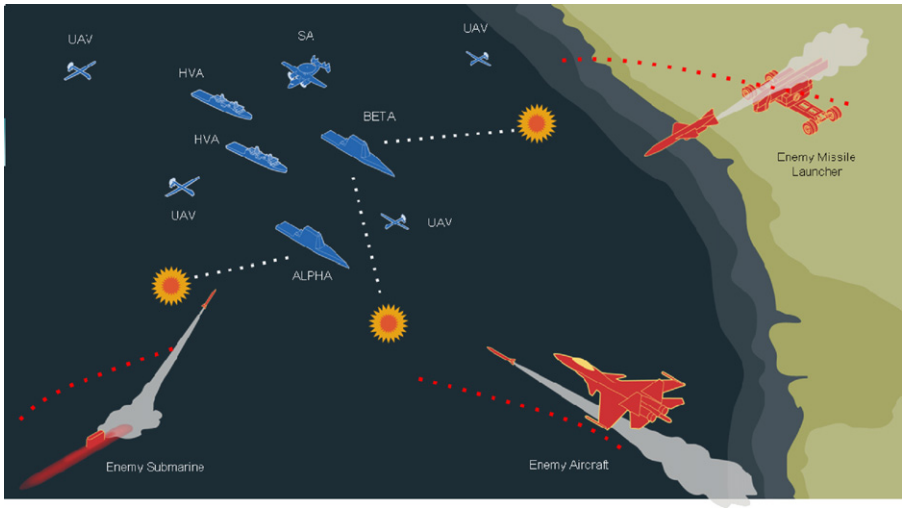


**Fig. 4.** A sketch for a vignette showing "own force" assets and enemy assets.

The *augmentation phase* (realizing principles 3 and 4) involves augmenting the end-to-end mission threads with quality attribute considerations and identifying SoS architectural challenges based on stakeholder inputs and the dialogue between the stakeholders and the architects. During the augmentation, overarching quality attribute considerations – as well as step-specific quality attribute considerations for each quality attribute of the SoS – are elicited and documented for each mission thread, just as a quality attribute scenario is elicited and annotated in the ATAM or QAW. The facilitation team ensures that the following are captured during the MTW: step-by-step and end-to-end quality attribute considerations; over-arching assumptions, engineering issues, challenges, additional use cases and mission threads (with a quality attribute context); capability, engineering and/or mission issues that arise; and a "parking lot" of issues concerning organization, programmatic, or non-technical issues that arise (and which will not be further pursued in the MTW).

In the *roll-up/follow-up phase* (realizing principle 4), two outputs are produced from the activities in the execution phase. One output includes the quality attribute considerations for each step of each mission thread selected for the MTW and the end-to-end quality attribute considerations for each mission thread. This is contained in a MTW template that is filled in during the augmentation phase. The other output is a briefing which describes the challenges identified during a post-MTW analysis session generated by the facilitation team. These challenges typically include architectural, engineering and capability issues. Each challenge contains a description, an estimation of its impact, and a set of recommendations.

At the end of the series of MTWs, an annotated summary briefing of rolled-up SoS challenges is developed by the facilitation team and vetted with the SoS stakeholders and finalized. This provides input to subsequent System ATAMs or SoS Architecture Evaluations.

### 3.3. System ATAM

Since the majority of software architecture evaluations using the ATAM have been done in the context of *systems*, it seemed natural to enhance the method to explicitly include system architecture considerations, to identify system and software architecture risks. The System and Software ATAM (System ATAM) was developed, based on minor enhancements to the ATAM, to evaluate a system and software architecture. Because the System ATAM has required only minor enhancements, the integrity of the proven ATAM remains intact. The System ATAM is not intended to be used to evaluate a SoS or enterprise architecture directly. However, it can play an important role in evaluating a SoS's constituent system and software architectures, to identify risks in the context of the overarching SoS architecture.

#### 3.3.1. Essential similarities

The System ATAM and the ATAM are similar in purpose and process. The purpose of both is: (1) to assess the consequences of architectural decisions in light of quality attribute requirements and business goals and (2) to discover risks created by architectural decisions. Both methods follow the same major phases and steps and generate same types of output. Some specific similarities include:

- Stakeholder-elicited scenarios remain a centerpiece (principle 3) and the format of these scenarios, their elicitation and prioritization is unchanged (including the use of the quality attribute utility tree (Bass et al., 2003; Clements et al., 2002)
- Architect uses the documented architecture to explain how the architecture satisfies the specific scenario under consideration (principle 6)
- Architectural approaches remain the focus of much of the analysis in the System ATAM (principle 5)
- The evaluation team still employs attribute-based questions focused on known architectural patterns and tactics to probe the architecture (principles 4–6)

#### 3.3.2. Essential differences

There are, however, a number of important differences between the ATAM for a software architecture and the System ATAM, specifically in the following areas:

- *System architecture notations and specifications*—a system architecture typically has notations and diagram types that are not included in software architecture documentation (e.g., functional block diagrams, system control and data flow diagrams, electrical diagrams, mechanical drawings).
- *Engineering considerations*—system architecture brings additional architectural and engineering considerations in several areas (e.g. electrical, mechanical, chemical engineering).
- *Quality attribute concerns*—there are also new quality attributes that are system architectural drivers (e.g., physical robustness, power supply continuity, fieldability).

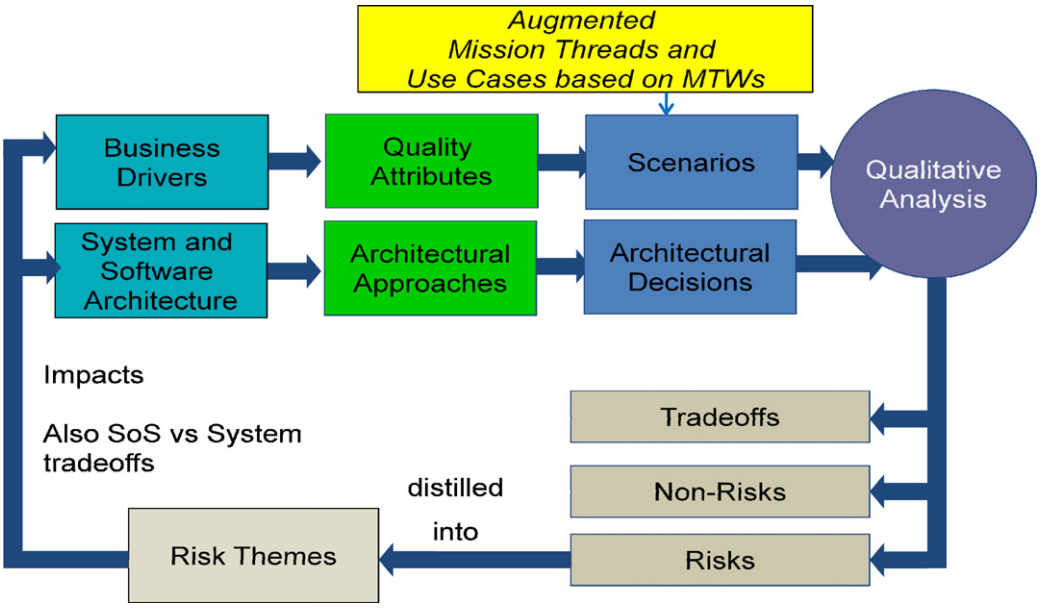


**Fig. 5.** Conceptual flow of the System ATAM.

- *Architectural approaches*—there are many additional architectural approaches for the additional engineering considerations.
- *Scope*–there are additional scoping considerations for a System ATAM that must be determined and agreed on.
- *Fast-tracking Subject Matter Experts (SMEs) onto the evaluation team*—the System ATAM lead must identify the need for an SME in a particular engineering discipline to participate on the evaluation team to handle additional system architecture/engineering considerations.

#### 3.3.3. Using System ATAM in a SoS Architecture Context

The System ATAM can also be used in a SoS architecture development context to help identify architectural risks associated with including a candidate legacy system into an SoS architecture. The augmented mission threads and use cases from prior MTWs can be used to generate system specific scenarios from the SoS context.

In preparation for such an evaluation, the System ATAM lead meets with SoS and the architects of individual systems to discuss what is "in" and "out" of scope concerning the system under analysis and if appropriate documentation exists. An agreement is reached on the scenarios to be considered (based upon the augmented mission threads) with the understanding that additional scenarios can be added during the system architecture evaluation. This process is depicted in Fig. 5, which is a minor variant of the conceptual flow of the ATAM:

### 3.4. System of systems/ecosystem architecture evaluation

We have thus far shown that the principles underlying the ATAM can be expanded to larger scope, such as analyzing systems (System ATAM) and to larger scale, eliciting requirements for SoSs (the MTW). Now we will show an example of how these principles allow us to undertake an analysis that is both larger in scale *and* scope – the SoS Architecture Evaluation – a technique that allows us to find risks in software-intensive ecosystems. The purpose of the SoS Architecture Evaluation is to evaluate the SoS architecture's ability to satisfy its mission goals as represented by a set of quality attribute requirements. This evaluation identifies architectural risks, using the augmented mission threads and architectural challenges developed in previous MTWs. The evaluation provides early identification of architecture risks and identifies problematic systems and SoS components for further evaluation. The architects, along with project management, can then identify, prioritize, and mitigate risks early in the lifecycle, before integration. Working early with project management to develop a sound architecture-centric acquisition strategy and associated artifacts motivates developers and contractors to do the right thing architecturally. It also gives project management visibility into the progress of SoS architecture definition.

The SoS Architecture Evaluation follows an ATAM-like approach, but is applied to an ecosystem—a collection of independent but interconnected systems. Like the ATAM, appropriate SoS architecture documentation is a prerequisite of the evaluation, and key stakeholder input is crucial to the success of the evaluation. Unlike the ATAM, we have found it necessary to conduct a *series* of SoS Architecture Evaluations focused on specific mission threads, attended by different sets of stakeholders. It is necessary to hold a *series* of evaluations to ensure that the proper stakeholders are present while not requiring other ecosystem stakeholders to be present for mission threads that do not affect their systems of concern. A typical evaluation is 1–2 days long and evaluates 3–4 mission threads.

A SoS Architecture Evaluation consists of three phases: preparation, execution, and roll-up/follow-up. In the preparation phase the following activities are performed:

- Review results of MTW, noting the architectural challenges.
- Identify the mission threads for the SoS Architecture Evaluation with the SoS architect(s).
- Identify the series of SoS Architecture Evaluations to be executed, associating stakeholders for each.
- Develop and review the SoS business/mission drivers and the SoS and system/software architecture presentations.
- For each SoS Architecture Evaluation:

  - Review results of relevant MTWs, noting the architectural challenges
  - Review SoS and system architecture documentation
  - Identify and invite stakeholders
  - Set up logistics and send out read-ahead with invitations
  - Walk-through one mission thread for practice
  - Identify evaluation team

As with the ATAM, the evaluation process begins with a set of presentations from key stakeholders: a SoS Architecture Evaluation Overview presented by the evaluation leader, a SoS business/mission driver presentation presented by a project manager, a SoS Architecture Presentation presented by the lead architect. Then the architects walk each end-to-end mission thread through the SoS architecture, explaining how the SoS architecture and constituent systems will satisfy (or be modified to accommodate) the mission thread. Since each mission thread evaluation might require the analysis of numerous constituent systems, the system architect for each system must explain the relevant architectural significant approaches employed without delving into implementation details. The SoS architects will also be asked to explain how the SoS architecture explicitly addresses the relevant architecture challenges indentified in the MTWs. During the explanations the evaluation team probes for risks, guided in particular by a knowledge of architectural patterns, tactics, and quality attribute considerations, all of which are scale-free. As in an ATAM, SoS architecture risks are "rolled up" into risk themes, which identify problematic areas (including identifying problematic constituent systems) for more focused architecture evaluation. These risk themes pose the greatest overall threats to the health of the ecosystem.

Finally, during the roll-up/follow-up phase the evaluation team aggregates risks into risk themes, reviews any left-over issues from the evaluation (what are termed "parking-lot" issues, which did not get addressed during the evaluation meeting), and delivers a summary of findings, risk themes and issues. Based on this analysis the evaluation team identifies problematic areas (including identifying problematic constituent systems) for more focused architecture evaluations (System ATAMs). Essentially the ecosystem analysis process has a fractal-like structure where, at each level of analysis information may be analyzed in the aggregate or, when necessary, a more detailed level of analysis may be invoked. But this more detailed level of analysis follows a virtually identical set of steps, and uses the same underlying techniques (principles 1–7).

## 4. Reflections on the analysis of software-intensive ecosystems

We have now executed over 25 MTWs, augmenting over 75 end-to-end SoS mission threads. We have also executed three System ATAMs and one SoS Architecture Evaluation. The majority of these were in the context of DoD mission-critical SoS architecture development efforts, but we have also analyzed commercial systems using these methods. The following contains categorized lessons learned in the application of the methods, based on the foundational principles discussed in Section 1.

### *4.1. The software architecture of a system is the fundamental artifact that guides development*

During the application of the methods, it was found that SoS Architecture Guidelines and Precepts were universally absent. These are, however, required to properly document the architectural approaches and rationale at the SoS architecture level. The system stakeholders agreed, and SoS Architecture Guidelines and Precepts document templates have since been adopted.

### *4.2. Systems are built to satisfy business goals*

In SoS that we have analyzed, the *only* end-to-end documented functional and capability requirements were the end-to-end mission threads that we developed and augmented in the MTWs. An SoS business/mission driver presentation is required for each MTW and all challenges identified in the MTWs have traceability back to the SoS business/mission drivers. In addition, legacy system/software architecture risks can be mapped back to SoS business/mission goals using the System ATAM method seeded with MTW outputs for scenario generation. In this way we are able to keep the stakeholders focused on risks and benefits in the SoS architecture, as they relate back to their individual and shared business goals.

### *4.3. Architecture design is based on a set of architecturally significant requirements, derived from business goals*

Augmented end-to-end SoS mission threads are the vehicle that couples SoS end-to-end functionality, derived from business goals, with quality attribute considerations elicited from the SoS stakeholders. Because of the chain of reasoning, from business goal to mission thread to architecturally significant requirement, these requirements are accepted by the SoS and constituent system architects and have been found useful and necessary for SoS architecture development.

The analysis of these architecturally significant requirements reveals many *system* risks: we were initially surprised that the MTWs uncovered as many capability and engineering risks as quality attribute and architecture-related risks. However, after many executions this appears to be the norm in current SoS architecture development efforts. Simply put, no one group appears to be taking ownership of the health of the *ecosystem*. The mission threads help to bring this risk to light.

### *4.4. Quality attribute requirements exert the strongest influence on architectural design*

For most SoSs, quality attribute requirements were initially not documented. The MTWs forced the stakeholders to document which SoS quality attributes were important to the overall health of the ecosystem, and this was driven by a consideration of the quality attribute implications of the mission threads. In addition, the System ATAM uses the quality attribute augmented mission threads as seed scenarios when evaluating the constituent system and software architectures in the context of a SoS Architecture Evaluation.

The quality attribute considerations that arise from consideration of a mission thread extend beyond that which is typical in software architecture analysis. Examples of new QA concerns:

- Survivability: Need to contain compartmental flooding in a critical compartment resulted in discussion on using new pump technologies to avoid flooding.
- Availability: Need to address a failure of a generator that has a massive impact on all ship operations and mission.

Despite the fact that the quality attributes and architectural primitives were new to us, the methods remained unchanged—the quality attributes desired still depended on the architectural decisions made, and our analysis revealed this dependency, and revealed risks in cases where the architectural design was lacking.

### *4.5. Architecture design can be made tractable by considering a small number of primitives, called tactics*

When analyzing systems of ecosystem scale, scalable analysis methods are crucial so that design and analysis remains tractable and so that we do not succumb to "analysis paralysis". Tactics are scale-free design primitives that can be identified by system or

SoS architects and analyzed with respect to the applicable scenarios and mission threads for risks. In practice what we have seen is that tactics-based analysis helps to provide strategic direction to the overall analysis effort: based on the results of a SoS or System ATAM analysis, deeper and more detailed analyses have been enacted.

### *4.6. Architecture design can and should be guided by analysis*

It is important to develop architectural challenges after each MTW and justify them from the thread augmentation discussions (and provide mapping between challenges and MTW. It is also important to vet MTW output challenges with the principles and reach agreement to impacts and recommendations. system/software architecture scenarios are easily generated from mission thread augmentations. System ATAMs can be easily seeded from previous MTWs to help identify legacy system/software architectural risks for inclusion within the SoS context.

The challenges identified in these analyses become risks if they are not addressed. In each case the outputs of analysis (MTW, System ATAM, or SoS Architecture Evaluation) guide further design and engineering activities (e.g. modeling and simulation, prototyping, engineering studies, etc.).

### *4.7. Architectures are developed by people within an organizational and business context; so economic and organizational concerns shape and constrain architecture*

Organizational stovepipes – often a barrier to effective interoperation between systems – were broken down using the scaled-up architectural analysis approaches described in this paper. Mission threads allow stakeholders from different operational areas and segments to inter-relate and understand each others' concerns. And stakeholders from diverse engineering disciplines were able to actively participate, using a mutually comprehensible language (at the level of architecture, which abstracts much of the detail that often obstructs effective communication) and get benefits.

Strong and impartial facilitation is, however, critical to the success of these methods, including the *independent* development of architectural challenges. This was cited numerous times by stakeholders as critical to success.

## 5. Conclusions

We began this paper by reviewing the foundations upon which over 10 years of architecture methods (and tools) have been built. Over time, the challenges facing the world of system and software architecture have only grown. Systems requirements have become more complex and the systems (and architectures) built to meet these requirements have grown correspondingly larger and more complex, leading us to a consideration of software-intensive ecosystems.

But the foundational principles of software and system architecture – quality attribute models, quality attribute scenarios, tactics, traceability from business goals to quality attribute requirements, etc. – have remained virtually unchanged over the past 10 years. And these principles have now been composed in a variety of ways into methods that address systems at all scales. The techniques underlying these methods have been proven to provide deep insights into architectures, irrespective of their scale. This suggests that these techniques are truly foundational.

## Acknowledgments

The authors would like to thank the staff of the RTSS Program at the Software Engineering Institute, who have collectively contributed and refined many of the ideas in this paper.

## References

Bachmann, F., Bass, L., Nord, R., 2007. Modifiability Tactics, CMU/SEI-2007-TR-002. Software Engineering Institute, Carnegie Mellon University.
Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice, 2nd ed. Addison-Wesley.
Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C., Wood, W., 2003. Quality Attribute Workshops (QAWs), CMU/SEI-2003-TR-016, 3rd ed. Software Engineering Institute, Carnegie Mellon University.
Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H., 2004. Architecture-level modifiability analysis (ALMA). Journal of Systems and Software 69 (1–2), 129–147.
Bosch, J., 2009. From software product lines to software ecosystems. In: Proceedings of the 13th International Software Product Line Conference (SPLC 2009) , San Francisco, CA, pp. 111–119.
C4ISR, 2006. C4ISR for Future Naval Strike Groups. National Research Council, ISBN: 978-0-309-09600-3.
Clements, P., Kazman, R., Klein, M., 2002. Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley.
de Boer, R., van Vliet, H., 2009. On the similarity between requirements and architecture. Journal of Systems and Software 82, 544–550.
Dobrica, L., Niemela, E., 2002. A survey on software architecture analysis methods. IEEE Transactions on software Engineering 28 (7), 638–653.
Hofmeister, C., Nord, R., Soni, D., 1999. Applied Software Architecture: A Practical Guide for Software Designers. Addison-Wesley.
Hofmeister, C., Kruchten, P., Nord, R., Obbink, H., Ran, A., America, P., 2007. A general model of software architecture design derived from five industrial approaches. Journal of Systems and Software 80 (1), 106–126.
ISO/IEC 9126-1: 2001 Software Engineering—Product Quality. Part 1: Quality model, http://www.iso.org/iso/iso_catalogue_tc/catalogue_detail.htm?csnumber=22749, 2001.
Kazman, R., Bass, L., 2005. Categorizing Business Goals for Software Architectures, CMU/SEI-2005-TR-021. Software Engineering Institute, Carnegie Mellon University.
Messerschmitt, D., Szyperski, C., 2003. Software Ecosystem: Understanding an Indispensable Technology and Industry. MIT Press.
Nuseibeh, B., Easterbrook, S., 2000. Requirements engineering: a roadmap. Proceedings of The Future of Software Engineering (ICSE 00), 35–46.
Paulish, D., 2002. Architecture-Centric Project Management. Addison-Wesley.
Scott, J., Kazman, R., 2009. Realizing and Refining Architectural Tactics: Availability, CMU/SEI-2009-TR-006. Software Engineering Institute, Carnegie Mellon University.
Storey, N., 1996. Safety-Critical Computer Systems. Addison-Wesley, Reading, MA.
van Lamsweerde, A., 2001. Goal-oriented requirements engineering. Proceedings of the Fifth IEEE International Symposium on Requirements Engineering, 249–262.

**Rick Kazman** is a Professor at the University of Hawaii and a Visiting Scientist at the Software Engineering Institute of Carnegie Mellon University. His research interests are software architecture, design and analysis tools, software visualization, software engineering economics, and human-computer interaction. Kazman has created several highly influential methods and tools for architecture analysis, including the SAAM (Software Architecture Analysis Method), the ATAM (Architecture Tradeoff Analysis Method) and the Dali architecture reverse engineering tool. He is the author of over 100 papers, and co-author of several books, including Software Architecture in Practice. When not writing about software, Kazman may be found cycling, playing the piano, gardening, practicing Tae Kwon Do, or (more often) flying back and forth between Hawaii and Pittsburgh.

**Mike Gagliardi** has more than 25 years of experience in real-time, mission-critical software architecture and engineering activities on a variety of U.S. Department of Defense systems. He currently works in the SEI Research, Technology, and System Solutions Program on the Architecture-Centric Engineering Initiative, and is leading the development of architecture evaluation and quality attribute specification methods for system and System of System architectures.

**Bill Wood** has worked at the SEI for the past twenty five years, and has held various technical andmanagerial positions over this time period; performing a combination of research activities andcustomer-interactive activities. He currently works in the Research, Technology and System SolutionsProgram at the SEI with a focus on developing and implementing methods that can be used to expose risks in a System of Systems Architecture and/or a System Architecture. The evaluation methods arebased on the Architecture Trade-off Analysis Method (ATAM). Much of his time is engaged directly withSEI clients. Mr. Wood previously worked for Westinghouse Electric Corporation in developing softwarefor a variety of real-time systems, such as: electric power distribution centers, subway command andcontrol systems, power plants, and chemical plants.