Analyzing the Architectures of Software-Intensive Ecosystems

Philip Boxer, Rick Kazman, *Member, IEEE*

Abstract— Software-intensive ecosystems include large numbers of independent software-intensive and human agents interacting with and responding to each other's demands in ways that are not amenable to traditional 'closed-world' assumptions. The paper describes the core-periphery structures of the systems participating in ecosystems, and approaches the analysis of their 'wicked' behavior from the perspective of the market behaviors that they are expected to support. It proposes that a key driver of the 'wickedness' is the accelerating tempo at which an ecosystem is expected to respond to new kinds of demand, making it necessary to extend the concept of 'architecture' to include the resultant processes of dynamic alignment. As a result, it becomes necessary to analyze architecture in a way that includes the operational contexts-of-use within which systems are being used. The paper proposes the use of a multi-sided matrix to represent the variety of forms of dynamic alignment demanded, and describes an extension to the Architecture Tradeoff Analysis Method as a means of discovering the risks inherent in architectural decisions made to support a software-intensive ecosystem.

Index Terms— D.2.11 Software Architectures, D.2.10.h Quality analysis and evaluation.

1 Introduction

An ecosystem is a community of managerially and operationally independent organizations interacting with each other and with their environment. Software-intensive ecosystems—ecosystems in which the behaviors of the participating organizations are themselves dependent on software because of the intensive use they make of it—are an increasingly important social, financial, and political force in the world. We find examples of software-intensive ecosystems in industries concerned with such things as transport, healthcare, defense, government, and communications.

These software-intensive ecosystems are different from traditional "closed-world" software systems that can be analyzed independently of the contexts in which they are embedded. Such ecosystems exhibit ultra-large-scale characteristics: they are constantly evolving, they have no centralized control, they have many heterogeneous elements, their requirements are inherently conflicting and unknowable, failures are normal, and the boundary between people and systems is blurred [1]. And the emergent properties of such ecosystems could not have been predicted by the designers of the software systems on which they depend.

A number of key drivers underlie this change, challenging the former "closed-world" perspective on software engineering. Amongst these are the tempo at which the ecosystems are themselves expected to evolve, the ubiquity and criticality of the software on which they depend, and the entanglement not only between software systems and the way they are used by people, but also between interoperating software systems that are themselves managerially and operationally independent of each other [2].

This paper will argue that, to understand the behaviors of such ecosystems, the analysis of their architectures must not separate the software systems from the organizational contexts-of-use that depend on them. We further argue that we must develop new ways of analyzing the patterns in how the parts of the ecosystem interact, and to what ends. This is the case even where the interactions between the organizations within the ecosystem primarily concern the use of software itself, such as is to be found in the Microsoft and iPhone ecosystems [3].

1.1 The challenge of 'wickedness'

Understanding the behavior of ecosystems presents a form of "wicked" problem [4]. Wicked problems have the following characteristics:

- There is no definitive formulation.
- They have no stopping rule.
- Solutions are not true-or-false, but good-or-bad.
- There is no immediate or ultimate test of a solution.
- They do not have a well-described set of potential solutions.
- Every implemented solution has consequences.
- Every wicked problem is essentially unique.
- Every wicked problem can be considered to be a

- *Philip Boxer is currently completing a PhD at Middlesex University and was until recently at the Software Engineering Institute, Carnegie Mellon, Pittsburgh, PA 15213. E-mail: pboxer@ optonline.net.*
- *Rick Kazman is with the University of Hawaii and the Software Engineering Institute, Carnegie Mellon, Pittsburgh, PA 15213. E-mail: kazman@sei.cmu.edu.*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

symptom of another problem.

- The causes of a wicked problem can be explained in numerous ways.
- The planner (designer) has no right to be wrong.

As Rittel and Webber said: "As we seek to improve the effectiveness of actions in pursuit of valued outcomes, as system boundaries get stretched, and as we become more sophisticated about the complex workings of open societal systems, it becomes ever more difficult to make the planning idea operational" [4]. We simply cannot draw a 'closed-world' box around the system and analyze it. This presents us with a challenge not just at the level of the ecosystem, but also at the level of the software systems on which it depends.

While this challenge appears to undermine our ability to do any meaningful analysis, simply not analyzing such ultra-large scale software-intensive ecosystems is an unpalatable option. Society is increasingly dependent on them, for example, the social networking communities enabled by the internet, the patterns of energy production, distribution and use enabled by the Smart Grid, the interconnected networks of health-care providers, users, and organnizations, or the forms of military response enabled by networked capabilities.

Let us briefly consider just one of these examples. The US Army, in considering the impact of such wicked problems concluded that a different approach to problem solving was needed that was *inductive* in nature, concerned with producing a well-framed problem hypothesis and an associated design for engaging with it—a conceptual approach for the problem [5]. Thus as much attention had to be paid to the way the problem was framed—the way the boxes were defined—as to the subsequent analysis of what was placed within and between those boxes. The conclusion reached was as follows:

> *The issue is whether a commander should begin by analyzing the mission objective, or whether complexity compels the commander to first understand the operational problem, and then—based upon that understanding—design a broad approach to problem solving. The answer to this question depends upon the problem and the mission objective. If the problem is structured so that professionals can agree on how to solve it, and the mission objective received from higher headquarters is properly framed and complete, then it makes sense to begin with the analysis of the mission objective (breaking it down into specified, implied, and essential tasks). However, if the problem is unstructured (professionals cannot agree on how to solve the problem), or the mission objective received from higher headquarters is not properly framed (it is inappropriate for this problem), or higher headquarters provided no clear guidance (permissive orders), then it is crucial to begin by starting to identify and understand the operational problem systemically.*

Another way of stating the challenge, therefore, is to improve our understanding of the organizational contexts-of-use into which software systems are being deployed, before analyzing any proposed architectures or architectural changes for such systems, to ensure that the software systems are as suitable as possible.

1.2 Analyzing complementarity within ecosystems

The early work on socio-technical systems defined enterprises as composite systems combining technological and social systems [6]. Analysis of the technological system involved "establishing a systematic picture of the tasks and task inter-relations required by a technological system", revolving around the establishment and control of the boundaries of this technological system and the way it interacted with its environment. But analysis of the social system involved establishing the nature of the social relations between the people engaged in the tasks defined by the technological system. The boundaries around these social systems related to the way in which meaning was shared within these boundaries [7]. The technological and social systems were therefore *complementary*: the behavior of each type of system was entangled in the behavior of the other, and the behavior of neither system could be defined independently of the other [8]. The composite system that was the enterprise therefore involved managing the alignment between these two systems to meet a shared goal. This socio-technical system was defined as being open, so that this alignment process had to be dynamic to be able to change and adapt to changes in its environment [9].

With software-intensive ecosystems we are dealing with multiple enterprises that are managerially and operationally independent of each other, but that are still necessarily socio-technical [10]; their software-intensive nature tells us something about the technological systems that they use. As a result, any analysis of software-intensive ecosystems has to be able to account for the behaviors of the complementary technological and social systems, and also has to be able to account for their alignment.

In this paper we examine the architectural characteristics that follow from this characterization of the problem of understanding ecosystems. We do this as a means of motivating how traditional architectural analysis can be extended to take account of complementarity. Such an analysis can give us insight into the properties of an ecosystem and can help us reason about the alignment of the technological systems within the ecosystem with the goals of the stakeholders within its many social systems (an example of the complexity of which can be seen with the Smart Grid [11]). For any organization that is planning on creating a substantial piece of an ecosystem, it is prudent to examine the architecture for this piece, to ensure that it will meet the organization's goals and will be appropriately adaptive to change from within and without.

2 The Metropolis Model and Core-Periphery Structures

The entanglement of the technological and social systems within an ecosystem demands an approach that can understand the impact of software not only on the technological systems, but also on the corresponding social systems with their associated alignment processes. This understanding is necessary to considering consequences for software development. The Metropolis Model [12] provides a starting point for understanding how software systems are constructed, maintained, and operated.

At the heart of this model is the distinction between the core and the periphery, which has been often noted as a key architectural construct in complex software systems [13]. The core-periphery architectural pattern provides the maximum opportunity for developers (and the producers and consumers—sometimes called *prosumers*—of content) at the periphery to embed the behavior of a system into their own contexts-of-use, enabling the activities of the eventual customers and end-users within the larger ecosystem. Examples of cores include the Linux kernel, the Android platform, Facebook's application platform, the Apache core, iPhone's iOS platform, Hadoop Common, and so forth. Each of these "cores" provides an abstraction upon which the actual end-user-facing functionality is built. The core itself provides little or no end-user value, but rather provides the architectural foundation upon which others build value. The core is typically relatively small, compared with the size of the periphery. It controls access to the platform's resources and hence provides the means to achieve the system's most important quality attributes (performance, modifiability, availability, security, etc.)

For example the core (iOS) platform of the iPhone enables application developers and users to develop uses independently that can in turn by taken up within a wide variety of contexts-of-use. For example, making GPS location data accessible to users and application developers on the iPhone platform has spawned a wide variety of location-based services. A simple diagram showing the relationships defined by the Metropolis Model is given in.

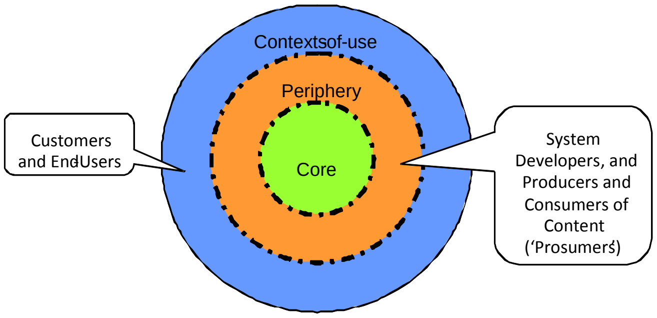
Contextsof-use
Periphery
Core
Customers and EndUsers
System Developers, and Producers and Consumers of Content ('Prosumers')

Figure 1: Metropolis Model Roles and Relationships

We can apply the core-periphery distinction to the way modularity and its associated inter-dependencies are defined for a software system. For modules in the core, tight coupling between them creates mutual dependencies. In the periphery, coupling is much looser creating a measure of independence in the way modules can be used. The strength of core-periphery pattern is therefore the relatively few constraints at the periphery on how modules may be used and combined, and how their use may be modified. Consequently a core-periphery structure is less over-determining of its use at the periphery than it is at the core. This under-determination frees stakeholders at the periphery to use systems in ways that have not been anticipated in their original design. This can enable novel compositions of systems (such as the mashups that appear regularly in web-based and open-source applications), but it also leads to emergent system properties.

The core-periphery distinction applied to the software describes patterns of *necessary* dependency. It can also be applied to the social system within an ecosystem, describing the ways stakeholders align their behaviors to each other. For example, according to the mirroring hypothesis [14], differences in the ways software modules are coupled will also tend to be mirrored by the organizations that develop them, even if these "organizations" are open-source communities or other ad hoc collections of interested individuals. And in open-source communities there is a subset of all developers, called *committers*, who have primary responsibility for the core functionality, integrity, and evolution of the overall system [3]. The core-periphery distinction applied to the social system describes patterns of *chosen* dependency, reflecting choices made by the stakeholders.

Together, the chosen and necessary dependencies described by the core-periphery distinction reflect the complementary forms of dependency associated with the technological and social systems within an ecosystem. The challenge to architecture analysis of the ecosystem is therefore to describe how necessary and chosen dependencies interact, bringing their complementarity within a unifying framework.

3 The Challenge to Architecture Analysis

Architecture analysis has been used for over two decades as a risk analysis and risk mitigation technique. The central tenet of architecture analysis is that one can profitably analyze the proposed architecture for a software system before it has been built, or before major changes to it are made [15]. In doing so, potential risks to the system may be discovered and mitigated at a low cost (relative to the cost of these risks going undiscovered until the system is developed or fielded). This has been shown over many years, by many researchers (e.g. [16], [17], [18]).

The Architecture Tradeoff Analysis Method (ATAM) [19] and its various progeny ([20], [21], [22]) have adopted the use of software system scenarios to elaborate and reify the quality attributes demanded of the system. This technique has shown itself to be well suited to understanding and analyzing the architectural characteristics of the single systems (and of the directed and acknowledged systems-of-systems[1]) needed to deliver those attributes. It

[1] Both directed and acknowledged systems-of-systems have a single design authority over the development of its composition from its (managerially and operationally independent) constituent systems. When

has been employed countless times over the past decade, in hundreds of large companies and government organizations.

A quality attribute scenario is a description of a stimulus-response pair: some portion of the system is stimulated and the system responds in a specified, measurable way. The impact of scenarios, like use-cases, can be traced through a software system. These scenarios, in addition to describing the *direct* interaction of some end-user with the software system, also describe a quality attribute of the system (a latency goal, for example) within the context-of-use defined by the scenario. The achievement of these quality-attribute-related responses brings benefit to the stakeholders in the behavior of the system. For example, a correct response might be of great value to an end-user stakeholder if it arrives predictably within 100 ms., of moderate value if it arrives predictably within 1 second, and of little value if it arrives unpredictably or if it predictably arrives in 10 seconds.

We can use scenarios to trace through the core-periphery interactions when trying to identify, measure, and understand the benefits radiating out to the end-users at the perimeter on the basis of their direct uses of the system. Scenarios such as these are necessary to analyze a shared infrastructure, such as a core. Such scenarios might illuminate contention for resources; consider, for example, understanding the load on an operating system kernel when multiple processes are operating and competing for shared resources. On a larger scale, cloud computing platforms such as Amazon's EC2 support large numbers of simultaneous users, each of which could request thousands of server instances [http://aws.amazon.com/ec2/]. And social computing platforms such as Facebook have over 250 million active users each day [http://www.facebook.com/press/info.php?statistics], each of whom will consume core resources, potentially interacting through applications that are built on top of Facebook's core application platform..

This is depicted in Figure 2, where three different scenarios are shown at the periphery. Each scenario sits within a particular context-of-use, and connects some end-user input—a stimulus—to an output (perceived by a potentially different end-user). Consider, for example, the connecting lines for each scenario: $\alpha_1\beta_1$, $\alpha_2\beta_2$ and $\alpha_3\beta_3$. These might represent different contexts-of-use in which end-users are making voice-over-IP (VoIP) connections, for example conferencing, or saying hello to a colleague on the other side of the world, or giving a seminar. The end-users within each scenario might be employing different software, but in the end they must use, and share, some resources at the core—name servers, protocol translators, routers, satellite links, IP stacks, fiber-optic cables, etc. Furthermore, the stakeholder perspective on each of these scenarios could be that the VoIP call would only be of value if latency and jitter were both kept within specified ranges.

In a software-intensive ecosystem, the under-determining effects of core software systems on their periphery free end-users to act independently of each other, but also enable new kinds of interactions between end-users. The scenarios associated with these interactions are not necessarily directly related to their use of any given system, and may involve multiple core systems that can be operationally and managerially independent of each other. For example, Facebook's location features allow users and service providers to interact with each other in ways that are only made possible by their use of a smartphone.

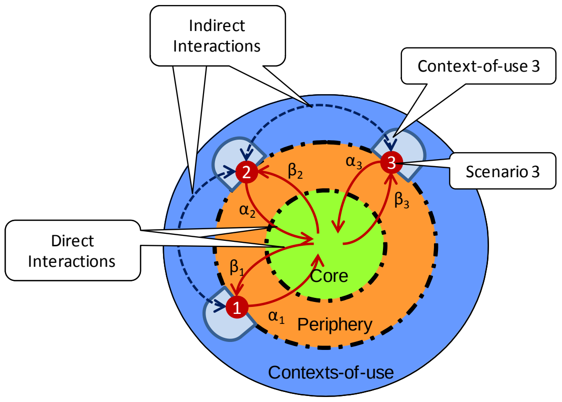

Figure 2: Scenarios over a Core-Periphery Structure

This creates a much larger and more complex environment defined by interactions that are *indirect* (from the perspective of the core software systems). For example, the actors within scenarios 1, 2 & 3 may belong to different research institutions, but are all involved in a single research collaboration for which a number of core systems become a key enabling element.

3.1 Multi-sided interactions

This more complex environment is represented in Figure 3 as a matrix of *multi-sided interactions* in which the sets of columns correspond to sets of direct scenarios associated with direct use of particular core technological systems (for example particular uses of VoIP or screen-sharing) that are operationally and managerially independent of each other. The rows correspond to the indirect scenarios associated with particular forms of social collaboration across multiple core systems (for example a research collaboration, or a marketing project). Thus the two axes of the matrix are used to represent the interactions between the technological and social systems within the larger ecosystem.

A new challenge arises therefore for the architecture analysis: what is the impact of these *indirect* interactions on the architecture of the operationally and managerially independent core systems (VoIP, Screen-sharing, etc.)? From the perspective of the core systems, each row is a collaborative system of systems in which *indirect effects* dominate its behaviors.

For example, consider, as another example, in Figure 3

deployed into an operational space, these systems-of-systems operate concurrently alongside other systems-of-systems using overlapping sets of constituent systems, collectively forming a collaborative system-of-systems [17].

the indirect effects that might arise where the end-users in the research collaboration are combining the use of VoIP, screen-sharing, and file sharing systems. Multiple X's in a column represent the architectural challenges facing the technological system. For example, many users of the VoIP systems will be competing for resources, such as computation and bandwidth, affecting latency. Or the developers of those applications may have chosen a common protocol affecting the development time of individual systems. Or charging schemes may affect an end-user's choice of combined systems to employ (for example as a result of differential charging schemes for relatively time-sensitive internet-traffic such as VoIP packets, and relatively non-time-sensitive traffic, such as file sharing).

Such considerations are of real concern to core providers because these indirect interactions can collectively have a huge effect on the performance and behavior of the core. As an example, Google Maps imposes resource restrictions on the services that it provides to other organizations, in an effort to moderate worst-case resource usage [http://www.zdnet.com/blog/google/google-maps-api-team-says-stop-it/429].

But decisions made affecting the capabilities of core systems will also impact on the way they can participate in the indirect interactions chosen by stakeholders. The variety and scale of these indirect interactions arising between stakeholders within the social system will determine the emergent qualities of the ecosystem. But multiple X's in a row represent social collaborations that also present architectural challenges: how will the systems interoperate, and how will the collaboration manage that interoperation within its larger context?

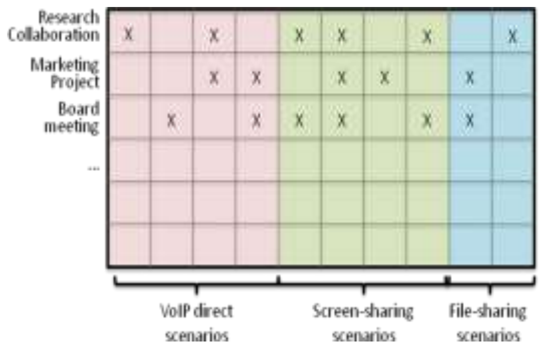

Figure 3: Multi-sided interactions

To summarize, the presence and impact of these indirect interactions is an *essential* characteristic of software-intensive ecosystems, arising from the entanglement of the technological and social systems. To understand these ecosystems, and to analyze them, we therefore need to be able to characterize a statistically representative set of the interactions between many independent actors across independent systems. But if we just analyze the direct impact of the αβ paths of each vertical scenario on the core-periphery architecture of any given system--what we do in traditional architectural analysis—we will not understand the collective impact of the indirect interactions between many such scenarios across many systems caused by the social collaborations taking place within the larger socio-technical ecosystem. We will thus not be in a position to analyze the potential emergent effects and behaviors arising from the way the core systems are used within this larger context. We must therefore be able to analyze the multi-sidedness of demand. This analysis will always be a sample of the demand, and hence effort must be taken when creating the multi-sided matrix to ensure that the indirect scenarios collected are statistically representative of the population of users and usages.

4 Extending Architectural Analysis

To be able to respond to this challenge, a modeling approach is needed that can represent the way the multiple stakeholders within a socio-technical ecosystem interact with each other. The multi-sided ways in which the technological and social systems align to each other within the ecosystem needs to be described and analyzed. To give three examples from our own experience:

- The suppliers of orthotic services within the UK National Healthcare System wanted to change the ways in which orthotics clinics were managed and regulated in order to improve the quality of care that they provided and reduce the underuse of their services by the larger ecosystem [23].
- A supplier of unmanned aerial vehicles (UAVs) needed to understand the way the use of these UAVs was changing within the context of the missions arising within operational theatre. Their understanding of this dynamic then determined what kinds of architectural change needed to be made to the UAV systems themselves [24].
- A government department supplying on-line search capabilities in support of an eGovernment initiative needed to understand how citizens' questions were changing. The department's understanding of how this dynamic impacted on the way departments needed to collaborate then determined what kinds of architectural change was needed to the search capabilities [25].

In each case, it was necessary to analyze the way the indirect interactions between enterprises (clinics, mission commanders, collaborating departments) and their customers (patients' conditions, adversaries' threats, citizens' questions) created multi-sided demands on the supporting technological systems. This involved distinguishing the stakeholders in the technological systems supplying products and services, and the stakeholders in the social systems identified with the horizontal collaborations creating indirect demands on those products and services. And it involved being able to model the impact of new and anticipated forms of collaboration on demand as well as on the way the supporting technological systems were used [26]. Again, this modeling can never be 100% accurate, but our intention is to represent a statistically valid sample of the known and anticipated forms of demand, to drive the analysis.

But these software-intensive ecosystems are 'wicked': any models of multi-sidedness are also hypotheses about what forms of multi-sidedness should be supported by any given technological systems within it on the basis of the view they take of the larger ecosystem. And this view depends on a prior analysis both of way the current technological systems can be aligned to the social systems of demand, and also of the way these demands are themselves changing. For example, Facebook makes its application platform available to developers who then create applications that add value to Facebook users. But these applications often weave together much more than just Facebook resources—they might incorporate Google maps, demographic data from the census.gov, Twitter feeds, personal information from other web-sites, and so forth. The forms of alignment are constantly changing and evolving. We can never completely master such evolutionary pressures, of course (as Rittel and Weber warned in their original discussion of wickedness, nearly 40 years ago), but the analysis that we do endeavors to capture the dimensions and degrees of wickedness as accurately as we can.

4.1 Eliciting models of multi-sidedness

An analysis of a software-intensive ecosystem starts from an analysis of the way forms of value are created within the ecosystem. That is, we must understand value as it pertains to the social systems *creating demand* rather than for the technological systems supplying products and services [27]. Consider, for example, the case of the orthotics clinics. The value was in the way treatments impacted on the patient's condition through the life of the condition within the context of the patient's life. For the UAVs, the value was in the way the UAVs could be combined with other assets to produce operational effects on threats with much greater timeliness and proportionality. And for the government departments responding to citizens' questions, their value was in the ability to respond to greater numbers and varieties of questions without commensurate increases in staffing levels.

These forms of value arise within specific contexts, giving rise to indirect demands on the supporting technological systems. The scope of an analysis of an ecosystem is therefore bounded by the analysis of the relationships between these indirect demands, the direct demands they give rise to, and the activities of the technological systems through which the demands are ultimately satisfied. This scoping starts with defining the relevant customer situations giving rise to indirect demands. The varieties of collaboration that respond to these indirect demands are then analyzed in terms of the multi-sidedness of demands they generate on the supporting technological systems [28]. In the case of the orthotics clinics, this meant understanding the referral pathways through which demands for treatment were defined. For the UAVs, it involved understanding how the tempo and nature of threats were changing within the context of unconventional warfare. And for the government it meant understanding how the nature of the questions varied over time with respect to the changing concerns of citizens.

The corresponding analysis of the supporting technological systems and the processes by which their use is aligned to the multi-sidedness of demands involves modeling tasks, resources, organizational processes and governance. Different methods of conceptual or structural modeling are constrained by the different categories of things and relationships between things that they make it possible to represent [29]. These determine the forms of knowledge that can be represented in particular domains [30, 31]. Many such frameworks exist, for example Zachman [32], DoDAF [33], or Federal Enterprise Architecture [34]. The characteristic of all of these frameworks is that they model physical and digital structures and behaviors, and the accountability hierarchies under which these operate. Thus they assume a single unifying architecture for a single entity (whether virtual or not).

Our method of analysis used for eliciting models of multi-sidedness admits multiple entities, adding representations for network relationships across these entities, and for the organization of customers' contexts-of-use [35]. This enables the relationships between multiple entities to be modeled, aligned to different definitions of demand, for example reflecting different ways of organizing multi-episode treatments [23], using UAVs in concert with other assets [24], or collaborating across government [25].

This extended representation [36] allows us to analyze the different forms of alignment with respect to different forms of demand through analyzing patterns of simple relations across the underlying models. These patterns enable us to represent the way underlying technological systems are aligned to the ultimately social contexts-of-use through a set of layers, producing a stratified analysis [23, 37]. The structural characteristics of this stratification then reveals different kinds of risk associated with the way their constituent parts within and between layers interoperate [38].

The multi-sided matrix therefore represents one layer within this stratification, being the intersection between these two analyses: of the way the technological systems respond to direct demands, and of the way the social systems of collaboration generate indirect demands.

4.2 Multi-sided ATAM

The nine steps of the ATAM, as it currently exists, are as follows:

Phase 1:

1. Present the ATAM.
2. Present business drivers.
3. Present architecture.
4. Identify architectural approaches.
5. Generate quality attribute utility tree.

6. Analyze architectural approaches.

Phase 2:

7. Brainstorm and prioritize scenarios.
8. Analyze architectural approaches.
9. Present results.

To analyze a software-intensive ecosystem a number of steps must be modified in order to take into account the complementary nature of its technological and social systems. The presentation of the architecture (step 3) will always be a snapshot, since the ecosystem is continually changing, but like the analysis of business drivers, it will be based on the prior elicitation process. Thus the analysis of the technological systems and the social systems of collaboration from which the multi-sided matrix is derived will need to explicitly include documentation of the way in which architectural constraints are placed on them and the processes of dynamic alignment between the two.

The presentation of business drivers in the ATAM (step 2) has previously assumed a single technological system directly supplying products or services. In an ecosystem, there is another set of "business drivers" that are more difficult to elicit and prioritize as they emerge from the social collaborations of end-users. These indirect business drivers are associated not just with the customer's business, but also with the other stakeholders with which the customer collaborates in responding to the customer's customers. Identifying them means elaborating the architecturally significant characteristics of the *collaborations* in which the customer participates, represented by indirect scenarios reflecting the different forms of collaboration supported by the technological systems.

Both the underlying technological systems and the social systems defining these collaborations are outputs of the elicited models of multi-sidedness described earlier. The multi-sided matrix therefore represents the particular ways in which these systems complement each other, representing the layer in the stratification where these two systems meet. From the perspective of a multi-sided ATAM therefore, the social layers of the overall stratification represent the indirect scenarios giving rise to the direct demands on the technological systems.

The most important change in a multi-sided ATAM is therefore this elicitation of the indirect scenarios, followed by their prioritization. In the ATAM there were two techniques for eliciting and prioritizing scenarios: the quality attribute utility tree (step 5) and scenario brainstorming and prioritization (step 7). In the multi-sided ATAM, these steps have to be done with two sets of scenarios—direct and indirect—distinguishing the interests of the stakeholders in the technological systems and social systems of collaboration respectively. The rows of the multi-sided matrix therefore define indirect scenarios presenting a *set* of stimuli and responses in the same way as the columns, but for different kinds of scenario. For the orthotics clinics, these indirect scenarios were the different kinds of referral situation emerging from the referral pathways through which demands for treatment were defined. For the UAVs, the indirect scenarios were different types of threat situation needing to be countered, and for eGovernment it meant understanding the different types of collaboration needed to respond to different types of questions from citizens.

Returning to the small-scale example in Figure 3, each column for VoIP stream, screen-sharing, and file-sharing session must have direct scenarios with associated stimuli and responses defining their direct scope across the rows. But since the rows themselves represent indirect scenarios (the point of which is to satisfy a social demand arising in the larger ecosystem, potentially for which its technological systems had never been designed), we must consider both dependencies and alignment processes between the direct scenarios along a row, each of which presents an interoperability requirement and a potential contention for shared resources between operationally and managerially independent systems. Consider, for example, the Research Collaboration in the first row in Figure 4.

Perhaps an end-user in this interaction wishes to capture the content of a screen-sharing session and share that with other end-users. Such a scenario imposes an interoperability requirement between the screen-sharing and file-sharing systems. This might be accomplished via a separate system function (i.e. this interoperability requirement might have been anticipated by the architects and implemented) or it might be accomplished via a user interaction (e.g. the user saves a file from the screen-sharing session, and then copies this file to a separate device from which it is file-shared with another user). If the former, then the system may be said to have an *indirect scope* defined by the systems it can enable to interoperate, shown in Figure 4 as a separate set of columns.

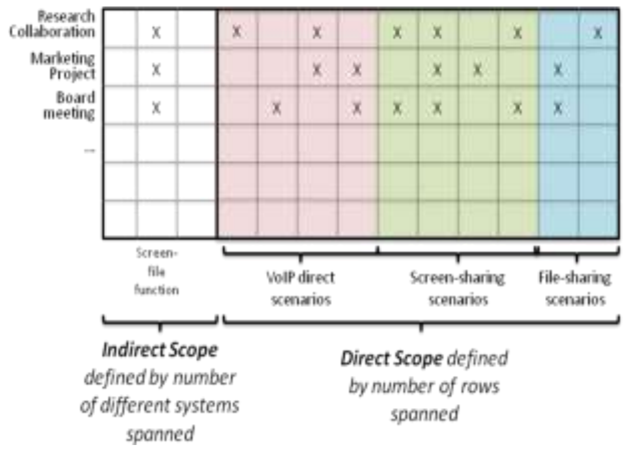

Figure 4: Functions with Direct and Indirect Scope

Whether the response to this inter-system portion of the indirect scenario is automated or manual, it has a number of important effects on the quality attribute responses of the collaboration as a whole that must be taken into consideration:

- its latency must be considered when analyzing the performance of the ecosystem as a whole.
- if the interoperation is accomplished by a system function, that function might be a single point of failure, thus compromising availability
- if the interoperation is accomplished by a user, that use might be a security risk for the system (by writing the information on a sheet of paper, or by transferring it using unsecured channels)
- if the interoperation is anticipated to change frequently, this might be a modifiability risk for the system

Furthermore, the set of rows (the indirect scenarios) in the multi-sided matrix define the *environment* under which each individual row is evaluated. Referring once again to Figure 4, the Research Collaboration will be sharing and competing for resources with the Marketing Project, the Board Meeting, and perhaps other indirect interactions which collectively may be modeled as stochastic processes.

The consideration of such a complex set of indirect scenarios then proceeds as any other scenarios would be analyzed in the traditional version of the ATAM: by mapping each scenario to a documented set of architectural approaches, and probing for risks associated with the mapping. In the above case we might probe for interoperability, availability, security, and performance risks. However, there is one important difference with the traditional ATAM scenario analysis: the rows of the multi-sided matrix describe the environment within which each scenario is analyzed. This analysis process will be exemplified in the next section.

5 An Example Analysis

In this section we provide an example showing how the multi-sided ATAM was approached, using a real-world system that was recently analyzed by researchers at the Software Engineering Institute. For the purposes of this paper, the prior elicitation process from which the matrix was derived has been omitted.

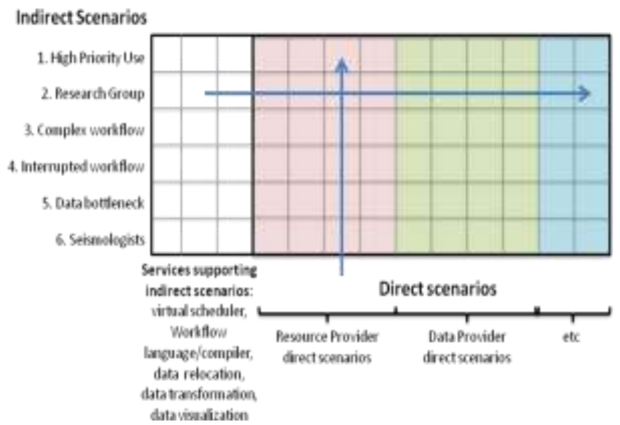
Indirect Scenarios
1. High Priority Use
2. Research Group
3. Complex workflow
4. Interrupted workflow
5. Data bottleneck
6. Seismologists
Services supporting indirect scenarios: virtual scheduler, Workflow language/compiler, data relocation, data transformation, data visualization
Direct scenarios
Resource Provider direct scenarios
Data Provider direct scenarios
etc

Figure 5: The Elicited Multi-Sided Matrix

This system is a proposed integrated grid supercomputing infrastructure for distributed high performance computing, in which the emphasis is on the variety of collaborations that can be supported. While the specific identity and details of the system have been altered, but the technical challenges identified are representative of those faced by the real system as it builds on the existing capabilities of the TeraGrid, particularly with reference to the role of Science gateways in supporting collaboration across science communities supported from gateways [https://www.teragrid.org/web/science-gateways/home].

To analyze this system, a multi-sided matrix was elicited through structured interviews with strategic stakeholders. A portion of the multi-sided matrix is shown in Figure 5. The indirect scenarios will be described in section 5.1.

Each row of the multi-sided matrix describes an indirect scenario with a set of end-to-end stimuli and responses addressing different quality attribute concerns. But recall that each row requires the orchestration of independent systems. An X in a cell of the matrix describes a single use of a single system (which includes its stimulus-response behavior). The entire row is a collection of these systems, appropriately interoperating, via human or system mediation. Finally, each column in the matrix describes a set of potential simultaneous demands on the systems—a set of simultaneously occurring stimuli with associated response goals. We will use this information in probing for risks in the ecosystem architecture.

Consider the example view of the grid-based high performance computing architecture presented in Figure 6, in which the services to end-users all have an indirect scope spanning multiple Data Providers (DPs), for example the research institutions within the Southerna Earthquake Centre; and Resource Providers (RPs) for example facilities provided by TeraGrid, each of which is operationally and managerially independent. An example of the type of scenario being considered is physics-based earthquake wave propagation simulations [39], with their associated challenges of managing large-scale scientific workflows [40] with their associated semantics [41]. Data transformation is concerned with aligning time-series physics calculations across different mesh sizes, and data visualization involves animated renderings of terrain deformations [42].

5.1 Example Indirect Scenarios

Consider the following scenarios which were a subset of those collected from the Utility tree and scenario brainstorming exercises:

1. A new job request from a high priority user occurs that spans multiple RPs. Within 15 minutes, an administrator reviews current usage and availability of resources suitable to accommodate this request, and enters the job into the job queue with high priority. The job is dispatched within 3 minutes and

completes execution within its deadline (6 hours).

2. An international research group schedules a regularly run job that employs six RPs and two DPs, all from different organizations. The schedule contains legacy-based estimates of the resources required from each of the providers, and the bandwidth required to move the data so that it is available "just in time" at each RP. The job executes as scheduled and terminates within 4 hours.

3. Three collaborating scientists submit a workflow entailing three related jobs at different remote sites, each of which needs different processing capabilities (speed, memory, cache size, etc.). The system automatically matches two of the three jobs to resources that satisfy their execution requirements. The third job requests resources beyond what is available. The system matches this request to the best available RP and notifies the user of the anticipated response time, given this allocation, within 30 minutes. The user can then choose to accept the allocation or cancel the workflow.

4. While a long-running collaborative scientific simulation employing 4 RPs at remote sites is executing one RP is shut down because it has lost its air conditioning. Jobs executing on this RP are suspended, relocated to an alternate RP, and restarted. Users of the workflow are notified within 10 minutes and the workflow is reconstituted within 20 minutes. Users can then choose to accept this new workflow or resubmit the job (or a portion of it, resuming from a checkpoint).

5. A team of scientists collaborating on an analysis requiring intensive simulation submit a 10 hour (scheduled) job for execution that runs 100 interdependent jobs at three different RP sites. This job involves a large amount of data staging from six other RP and DP sites, and transfers of computed data between these sites. The job was scheduled to use 50% of the resources at site 1 throughout its execution cycle, but, due to slow arrival of data, a site monitor determines that site 1 is consistently only using 5% of its resources over the first hour of the job, and alerts the users. The users in turn contact a system administrator who suspends the job, determines that the source of the bottleneck is a single DP, re-allocates two additional DPs to share the load, and restarts the job from the latest checkpoint. Execution completes with less than a 2 hour delay.

6. A team of seismologists concerned about an impending earthquake in a highly populated area needs to use multiple sets of sensor data to anticipate forthcoming events. These scientists need to run concurrent simulations of 1000 predictive models at different levels of fidelity within 6 hours, to generate recommendations for civil defense authorities.

5.2 Example Analysis

To see how this elicited information is employed let us first consider the evaluation of indirect scenario 6 above and in Figure 5. The architect traced out the steps involved in satisfying this scenario using the architectural representation depicted in Figure 6:

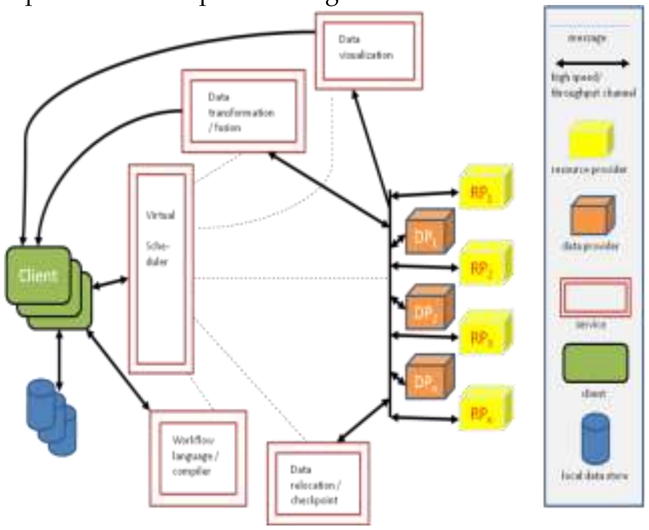

Figure 6: Architectural representation

- a user first interacts with the Workflow language and compiler component to define the programs to be run and their (data) dependencies; the Workflow language/compiler provides an executable script that ensure that programs are executed in the correct order and that data is transferred, stored, and transformed appropriately
- this script is transferred to the Virtual Scheduler, which begins allocating resources in an attempt to meet the specified 6 hour deadline
- the Virtual Scheduler allocates jobs to a set of organizationally independent resource providers and data providers.
- the Data relocation/checkpoint service is used to transfer data among the independent resource providers and to perform periodic checkpoints (to provide snapshots of stable intermediate results in case of processing failures or interruptions)
- the Data transformation/fusion service is used to translate data formats and to fuse data sets so that different programs, resource providers, and data providers can exchange data and interoperate
- when the jobs have all completed the results are sent to the Data visualization service and then returned to the requesting user

The purpose of the multi-sided ATAM is to discover risks in the architectural decisions that have been made and in decisions that have not been made as they apply to the row collaborations. In the course of evaluating scenario 6 a number of risks were noted:

1. the virtual scheduler employs "best effort" schedul-

ing; it allocates jobs fairly, according to the pool of jobs to execute, the pool of available resources, and the priorities of the jobs. However, the policies for job prioritization are unclear, particularly since the system is not "owned" by a single organizational entity

2. the virtual scheduler is a single service and hence represents a single point of failure for the system
3. the performance characteristics of Workflow language/compiler are unknown for very large jobs; it is possible that the calculation of the workflow itself could consume significant time, putting the deadline for the job at risk
4. combining data relocation and fusion into a single service component adds unnecessary complexity and if the component fails both services are disabled
5. the data relocation/checkpoint service is a single point of failure and potential performance bottleneck; if this service is split and/or distributed into multiple service components then it will become a potential complexity risk for the system as a whole
6. the data transformation/fusion service is a single point of failure and a potential performance bottleneck
7. there is no system-wide standard protocol for reporting service failures and outages

These risks could be discerned purely through an examination of the indirect scenario as it was mapped onto the architectural representation. Additional risks can be found when considering the "environment" of scenario 6, which is to say the other scenarios (1-5) that may be competing with scenario 6 for resources. Based on a consideration of this environment, additional risks were discovered. The risks identified below emerged during the process of elicitation itself as a consequence of bringing the separate parts of the analysis together. (The emergence of such insights is intrinsic to an ATAM-like process. More detailed analysis of risks arising from the absence of interoperabilities across the wider ecosystem is beyond the scope of this paper, but an example of their analysis within a different ecosystem is detailed in [43]):

8. the worst-case performance characteristics of the system are not well understood. Scenarios 1, 2, and 5 all have deadlines. Scenarios 3 and 4 will impose additional loads on the system. Without a system-wide policy and processes for resource reservation, negotiation, arbitration, or bidding, it is unclear at best how the worst-case behaviors of the system will be controlled within the context of concurrent research collaborations.
9. scenarios 3 and 4 both involve user responses. While waiting for these responses resources (that could otherwise have been used) may be allocated but not used. This represents a performance and availability risk for the system.
10. the anticipated distribution of job requests that the system will face is not well understood. This needs to be studied and statistically characterized in order to understand how the variability in indirect demands on the system will impact on its performance. The impact of such variability will be on the economics of supporting emergent forms of collaboration, risks to existing collaborations arising from new forms of interoperability, and of course performance issues arising from changing loads on the different parts of the system.

What else can we learn from this collection of scenarios? The risks identified above relate to the systems supporting horizontal collaborations. These systems are themselves socio-technical, so that a number of risks arose during the analysis due to a consideration of the *human* elements in these collaborations. For example, scenarios 1 and 5 require the actions of system administrator. This person is, then, a potential performance bottleneck if these scenarios, and others like them, occur concurrently. In scenario 3, the actions of the collaborating scientists could lock valuable resources while the system waits for the users to make a decision, preventing other users in the ecosystem from employing them. In the analysis of scenario 4 it was noted that some RPs require data in different formats. This presents an interoperability risk (since a human would need to introduce a format translation as a repair mechanism), and a performance risk (since the computation will need to be suspended until the human has acted and the data transformation has completed). The analysis therefore presents us with the challenge of levels of detail: how much refinement is needed to satisfy the interests of the stakeholders in the analysis process itself? This brings us back to the importance of prioritization. The methods described here have to serve those priorities, but cannot determine them.

5.3 Reflections

Clearly the analysis that we have just described builds upon the existing ATAM techniques. But the analysis goes beyond what would be done in an ATAM in several important ways, beginning with the multi-sided matrix. The elicitation of the multi-sided matrix causes the analysts to consider not just direct scenarios (which the traditional ATAM considered), but also indirect scenarios. A consideration of indirect scenarios requires analysis of the system as a socio-technical ecosystem, including humans as both principals and agents, and including many resources that are not under direct control of any single authority. These humans both determine and become part of the execution of the system and may themselves represent performance, availability, security, interoperability or other kinds of risks.

Each of these changes is a departure from the traditional ATAM, which was focused on analyzing the architectures of systems under the control of a single entity (or a known, finite number of entities) with reasonably well understood requirements. The analysis of ecosystem architecture forces us to relax those assumptions and, in doing so, creates new obligations for analysis.

6 Conclusions/Future Research

In this paper we have examined the architectural characteristics of software-intensive ecosystems, to suggest how traditional architectural analysis can be extended to address the complementary nature of their technological and socials systems, reflected in the multi-sidedness of demands on the use of technological systems. We have argued that, in fact, such analysis *must* be extended. It must be extended to account for how dynamic processes of alignment, in response to rapidly changing demand, impacts on the wicked (ill-structured) nature of ecosystems. We have proposed the use of a multi-sided matrix to represent the variety of forms of dynamic alignment and, given this information, have proposed an extension to the ATAM that allows us to analyze software ecosystems.

Such an analysis can give us insight into the properties of an ecosystem and can help us reason about the alignment of the ecosystem with the goals of its many stakeholders. In the early years of ATAM, the challenge was to define effective architectures for the use of software systems. Now this challenge extends to analyzing the effectiveness of embedding these software systems within socio-technical ecosystems.

In addition to the application of ATAM to multi-sidedness, we see several other potential forms of analyses based on the data elicited to generate the multi-sided matrix:

1. a structural analysis of potential risk areas in both technological and social system architectures, using a dependency structure matrix (DSM) to identify core-periphery characteristics in both technological and social systems [13]
2. a stratification analysis of the processes aligning the technological and social systems to new and/or different forms of demand. This enables the risks to be identified in aligning systems to different forms of demand (for example in evaluating the risks facing military operational capabilities facing a changing threat environment [44]).
3. an economic analysis of the consequences of changes in demand on the use of the architecture (for example in assessing the value of investment in eGovernment search capabilities to support greater responsiveness to citizens [45], and in assessing the value of investing in unmanned aerial vehicle capabilities to support greater agility in military force structures [43]).

References

1. Northrop, L., et al., *Ultra-Large-Scale Systems: The Software Challenge of the Future*. 2006, Pittsburgh: Software Engineering Institute, Carnegie Mellon University.
2. Coyle, L., et al., *Guest Editor's Introduction: Evolving Critical Systems.* Computer, 2010. **43**(5): p. 28-33.
3. Jansen, S., A. Finkelstein, and S. Brinkkemper. *A Sense of Community: A Research Agenda for Software Ecosystems.* in *31st International Conference on Software Engineering (New Ideas and Emerging Results Track)*. 2009. Vancouver, Canada: IEEE CS Press.
4. Rittel, H. and M. Webber, *Dilemmas in the General Theory of Planning.* Policy Sciences, 1973.
5. TRADOC, *Commander's Appreciation and Campaign Design*. 2008.
6. Emery, F.E. and E.L. Trist, *Socio-technical systems*, in *Management Science, Models and Techniques*, C.W. Churchman and M. Verhulst, Editors. 1960, Pergamon: London. p. 83-97.
7. Miller, E.J. and A.K. Rice, *Systems of Organization: The Control of Task and Sentient Boundaries*. 1967, London: Tavistock.
8. Atmanspacher, H., H. Romer, and H. Walach, *Weak Quantum Theory: Complementarity and Entanglement in Physics and Beyond.* Foundations of Physics, 2002. **32**(3): p. 379-406.
9. Bertalanffy, L.v., *The theory of open systems in physics and biology*. Science, 1950. **111**: p. 23-29.
10. Walker, G., et al., *A Review of Sociotechnical Systems Theory: A Classic Concept for New Command and Control Paradigms*. 2007, Human Factors Integration Defence Technology Centre.
11. National Energy Technology Laboratory, *Advanced Metering Infrastructure*. 2008, NETL.
12. Kazman, R. and H.-M. Chen, *The Metropolis Model: A New Logic for the Development of Crowdsourced Systems.* Communications of the ACM Vol 52 No 7, 2009: p. 76-84.
13. MacCormack, J., C. Baldwin, and J. Rusnak, *The Architecture of Complex Systems: Do Core-Periphery Structures Dominate?* MIT Sloan School of Management, 2010. **Working Paper 4770-10**.
14. Colfer, L. and C. Baldwin, *The Mirroring Hypothesis: Theory, Evidence and Exceptions.* Harvard Business School Working Paper, 2009.
15. Falessi, D., et al., *Decision-making Techniques for Software Architecture Design: A Comparative Survey.* ACM Computing Surveys, 2010.
16. Bengtsson, P. and J. Bosch. *Architecture level prediction of software maintenance*. in *3rd European COnference on Software Maintenance and Reengineering (CSMR 99)*. 1999. Amsterdam, Netherlands.
17. Lassing, N., D. Rijsenbrij, and H. van Vliet, *How well can we predict changes at architecture design time?* Journal of Systems and Software, 2003. **65**: p. 141-153.
18. Tarvainen, P. *Adaptability evaluation of software architectures; A Case Study*. in *31st Annual International Computer Software and Applications Conference*. 2007. Beijing.
19. Clements, P., R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. 2001: Addison-Wesley.
20. Chen, H.-M., R. Kazman, and A. Garg, *BITAM: An Engineering-principled Method for Managing*

Misalignments between Business and IT Architectures. Journal of Science of Computer Programming, 2005. **57**(1): p. 5-26.

21. Barbacci, M., et al., *Quality Attribute Workshops*, in *Technical Report*. 2003.
22. Moore, M., et al. *Quantifying the Value of Architectural Design Decisions: Lessons from the Field*. in *Proceedings of the 25th International Conference on Software Engineering (ICSE 25)*. 2003. Portland, OR.
23. Cohen, B. and P. Boxer, *Why Critical Systems Need Help To Evolve.* Computer, 2010. **43**(5): p. 56-63.
24. Boxer, P., *Valuing Multi-Sided Systems*. 2009, CMU/SEI-2009-SR-012 unlimited distribution (in draft): Pittsburgh.
25. Boxer, P. and H. Sassenburg, *The Swiss eGov Case: "Metadata 2010"*. 2010, CMU/SEI-2010-SR-003 Unlimited distribution: Pittsburgh.
26. Baldwin, C.Y., *Where do Transactions come from? Modularity, transactions, and the boundaries of firms.* Industrial and Corporate Change, 2007.
27. Prahalad, C.K. and V. Ramaswamy, *The Future of Competition: Co-Creating Unique Value with Customers*. 2004, Boston: Harvard Business School Press.
28. Boxer, P., et al. *Systems-of-Systems Engineering and the Pragmatics of Demand*. in *Second International Systems Conference*. 2008. Montreal, Que.: IEEE.
29. Sowa, J.F., *Knowledge Representation: Logical, Philosophical, and Computatinal Foundations*. 2000: Brooks/Cole.
30. Amin, A. and P. Cohendet, *Architectures of knowledge: firms, capabilities and communities*. 2004: Oxford University Press.
31. Hopkins, R. and K. Jenkins, *Eating the IT Elephant: Moving from Greenfield Development to Brownfield*. 2008: IBM.
32. Zachman, J.A., *A Framework for Information Systems Architecture.* IBM Systems Journal, 1987. **26**(3): p. 276-292.
33. U.S. Department of Defense, *DoD Architecture Framnework Version 2.0*. 2009.
34. The Office of Management and Budget, *Federal Enterprise Architecture Consolidated Reference Model V2.3*. 2007.
35. Boxer, P. and S. Garcia. *Limits to the use of the Zachman Framework in Developing and Evolving Architectures for Complex Systems of Systems*. in *SATURN*. 2009. Pittsburgh: SEI.
36. Boxer, P.J. and S. Garcia. *Enterprise Architecture forComplex System-of-Systems Contexts*. in *3rd International Systems Conference*. 2009. Vancouver, BC: IEEE.
37. Boxer, P.J. *Building Organizational Agility into Large-Scale Software-Reliant Environments*. in *3rd International Systems Conference*. 2009. Vancouver, BC: IEEE.
38. Anderson, W.B. and P. Boxer. *Modeling and Analysis of Interoperability in Systems of Systems Environments*. in *IDGA Systems of Systems Engineering Forum*. 2009.
39. Deelman, E., et al. *Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The CyberShake Example*. in *IEEE e-Science and Grid Computing*. 2006. Amsterdam, Netherlands.
40. Deelman, E. and Y. Gil. *Managing large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges*. in *IEEE e-Science and Grid Computing*. 2006. Amsterdam, Netherlands.
41. Gil, Y., et al. *Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows*. in *OWL: Experiences and Directions 2006*. 2006. Athens, GA.
42. Chourasia, A., et al., *Insights gained through visualization for large scale earthwuake simulations: Discovering the Unexpected.* Computer Graphics and Application Journal, 2007. **2**(6): p. 28-33.
43. Anderson, W. and P. Boxer, *Modeling and Analysis of Interoperability in Systems of Systems Environments.* CrossTalk, 2008.
44. Anderson, W., P. Boxer, and L. Brownsword, *An Examination of a Structural Modeling Risk Probe Technique*. 2006, CMU/SEI-2006-SR-017: Pittsburgh.
45. Boxer, P., P. Kirwan, and H. Sassenburg. *The Impact of Governance Approaches on SoS Environments*. in *IEEE 4th International Systems Conference* 2010. San Diego, California: IEEE.

Philip Boxer is currently completing a PhD at Middlesex University, London and was until recently a Senior Member of the Technical Staff at the Software Engineering Institute of Carnegie Mellon University. His research interests include the economics and architectural and risk characteristics of the sociotechnical ecosystems within ultra-large-scale systems. Boxer received a BSc in electrical and electronic engineering from King's College in London University and an MSc in business administration from the London Graduate School of Business Studies. He is a member of IEEE, the International Council on Systems Engineering, and the Institute of Business Consulting.

Rick Kazman is a Visiting Scientist at the Software Engineering Institute of Carnegie Mellon University and Professor at the University of Hawaii. His primary research interests are software architecture, design and analysis tools, software visualization, and software engineering economics. He is the author of over 100 papers, and co-author of several books, including "Software Architecture in Practice", and "Evaluating Software Architectures: Methods and Case Studies". Kazman was one of the creators of the SAAM (Software Architecture Analysis Method) and the ATAM (Architecture Tradeoff Analysis Method). Dr. Kazman received B.A. and M.Math degrees from the University of Waterloo, a M.A. from York University, and a Ph.D. from Carnegie Mellon University. Dr. Kazman is a member of the IEEE and the IEEE Computer Society.