Editorial Chapter

Ivan Mistrik  
Independent Consultant, Germany

Antony Tang  
Swinburne University of Technology, Australia

Rami Bahsoon  
University of Birmingham, UK

Judith A. Stafford  
Tufts University, USA

ABSTRACT

Architects (of any kind) need to understand the system, its environment, its stakeholders, their concerns, and the solution approach. The solution approach varies across different kinds of architecture. In software architecture, the solution approach is software-oriented. For system architecture, the solution approach is hardware- and platform-oriented. For enterprise architecture, the solution approach is people- and business-oriented. The goal of this introductory chapter is to clarify the relationships among different kinds of architecture, explore and identify areas of commonalities and difference, and to discuss the current challenges and future directions in aligning these architectures.

WHAT ARE ENTERPRISE, SYSTEM, AND SOFTWARE ARCHITECTURE?

When we use the term architecture, we mean that architecture as a product resulting from an application of the science of architecting to a system or other entity of interest.

Enterprise, System, and Software architectures play a critical role in developing software-intensive complex systems. However, the scope and focuses of each type of architecture design is different but overlapping. Enterprise architecture is a complete expression of the enterprise; it describes the alignment of business processes with IT, the composition of software systems and subsystems, and their relationships with the external environment, and the guiding principles for the design and evolution of an enterprise (Giachetti, 2010). Enterprise architecture acts as a collaboration platform between aspects of business planning such as goals, visions, strategies and governance principles; aspects of business
operations such as business terms, organization structures, processes and data; aspects of automation such as information systems and databases; and the enabling technological infrastructure of the business such as computers, operating systems and networks (ISO/IEC, 2010).

System architecture is defined as a formal description of a system, or a detailed plan of the system at component level to guide its implementation (The Open Group, 2009). It is also the structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time. System architecture comprises the design of both software and hardware systems, design issues arising from such a system architecture design often concern both systems.

As the field of Software architecture enters its third decade of formal study it finds itself moving from its traditional focus on software structures to the more general notion of software architecture as the set of design decisions made to ensure the software requirements will be met. Consistent with this view is the trend toward focusing software architecture documentation on supporting stakeholders in understanding how the software solution satisfies their concerns. These stakeholder concerns can be viewed roughly along two axes: concerns related to the software itself and concerns about how the software relates to other systems with which it interacts. Over the last decade it has been widely recognized and accepted that architectural decisions must take into consideration not only the functions the software is expected to perform but also the quality attributes associated with the functions. In recent years the community has come to consensus that quality concerns related to security, performance, reliability, maintainability, etc. cannot be ensured unless they are considered and their realization mechanisms properly documented from the outset, and they cannot be coded in during development.

In the next decade we will see increasing focus on the second of our two axes: how systems software relates to the software of systems with which it interacts to support the quality requirements. Similarly, how software and systems can be architected to provide the flexibility and quick responses to market changes that are required by enterprises. Thus there will be increasing collaboration between those concerned with software architecture and those concerned with system and enterprise architectures.

The following definition seems equally applicable to enterprise, systems, software and other aggregations of interest:

**Architecture:** The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution (ISO/IEC, 2010).

The key ideas in the IEEE 1471 and ISO/IEC42010:2010 definition are these:

- Architecture embodies the fundamental concepts about a system, including its components, their relationships and governing principles.
- Architecture recognizes the role and influence on an architecture of the environment in which a system is embedded.
- Architecture is not merely about the static arrangement of a system parts but its evolution and principles governing that evolution.

All three architecture disciplines have been maturing over the past decades. Enterprise architecture has been evolving around the premise of aligning business processes with IT. It deals with the business management levels for planning purposes. Plans and designs are then translated into systems and software designs. Many methodologies and frameworks have been used in the industry: IEEE 1471 (IEEE, 2000) was first approved in 2000. Other frameworks address sys-
tem and/or enterprise concerns, such as GERAM (IFIP-IFAC Task Force, 1999) and ISO 15704 (ISO, 2000), Zachman Framework (J. Zachman, 1987; J Zachman, 1997), the US DoD Architecture Framework (Department of Defense, 2007) and related frameworks such as MODAF (Ministry of Defence (UK), 2012) and TOGAF (The Open Group, 2009). On the other hand, system and software architectures view setting requirements as an important part of the architecturally significant decision making process (de Boer et al., 2007). Architects in each field have different processes and methods for carrying out design and implementation. To date, these processes and practices are not seamless across the different levels of architecture design, and the integration in theory and practice is still a challenge.

We need to improve our current understanding of the relationship between the types of architecture, their related artifacts, their processes and the systems supporting their associated reasoning process. There is a disjoint between different types of architectures (enterprise, system, software) due to misunderstandings across the boundaries between architects. The rather simplistic view of segregating a development process into enterprise, system and software architectures cannot solve the issues in complex system development. An alignment of these architecture design processes is imminent to improve the current practice.

WHAT IS ALIGNING ENTERPRISE, SYSTEM, AND SOFTWARE ARCHITECTURE?

As business processes are supported and realized by software systems, the design of software and systems in turn can pose constraints and generate new requirements. The interrelationships between the three areas of architecture designs are intricate and interdependent. The current practice of treating the three architectures separately cannot work as Business-IT alignment remains a major issue (Luftman, Papp, & Brier, 1999). This issue is partially a result of misalignments between the three architectural practices. For instance, the contradictions in the scoping, feasibility, budget and schedule between enterprise architecture and software architecture are often experienced in practice when the two architecture planning are misaligned.

One of the reasons for the gap between enterprise, system, and software architecture, is that there have not been sufficient considerations of the commonality between them. Not only does each community have its own paradigm (metamodel, languages, methods etc.), they also publish their research results and experiences in different forums (conferences, journals, books), duplicating their efforts and results.

The last decade provides a rich set of architecture-related standards, motivated both by recognized practices and a hope that standards can effectively improve practice. There is a need for more structured and systematic treatment of the relationships between the three kinds of architectures.

From the industrial perspective, we observe that practitioners in the field deem the knowledge linking an enterprise to its architectures as a core knowhow, and invest major efforts in keeping track of the relations between enterprise, system, and architectural elements. For instance, software and enterprise architects regard business goals and their relation to architectural decisions as core information to be maintained for architecting. Although key business processes drive decision making for IT projects, organizations continue to experience major problems when the information aligning business processes and technical architectures is missing.

At various conferences/workshops and in several research projects many issues in enterprise (EA), system (SA), and software (SWA) architectures have been explored. Methodologies and approaches for defining, improving and aligning the three types of architecture activities have been
suggested. From these studies, e.g., (Bergey et al., 2009; WICSA/ECSA, 2009) a number of key questions have emerged:

- **What are the major activities involved in each kind of architecture?** Any architecture is motivated by a set of stakeholders, a set of concerns, a set of mandatory “architecturally significant requirements,” and a “vision” (something to use to make tradeoffs, to say how systems might evolve). These frame what the architect does to produce asset of viewpoints and instantiate them to show the architecture meets the concerns and complies with the vision. So the activities could be articulated as the following: understanding goals, context, and requirements; creating, evaluating, and documenting architecture; managing the architecture post-creation; and assisting in post-architecture activities.

- **What is the boundary between the different kinds of architecture?** The boundaries of the kinds can be understood in terms of the concerns of stakeholders for each kind of architecture. Questions to be asked when moving from one kind of architecture to another are: What stakeholders should be involved in each type of architecture? What are the common concerns of these stakeholders and how do their interests shift? What new concerns arise or are reinterpreted?

- **What common activities exist for each type of architecture (e.g., specification, evaluation)?** There are some basic activities that any kind of architecting would take (Emery & Hilliard, 2009). Some of these activities include defining the problem, understanding the context and environment in which problem and solution are situated, and working the problem within those bounds in terms of requirements analysis and specification, solution modeling and explorations. (Hofmeister et al., 2007) discuss a generic process for Software Architecting, aside from some terminology, there is no reason to believe the archetype would not work for other kinds, or architecting in general.

- **How do we capture and represent architecture in each kind?** The workshop on architecture genres (Bergey, et al., 2009) has summarized its findings on the documentation issue as follows:
  - **Enterprise Architect:** No ‘one size fits all’ no de facto standard.
  - **System Architect:** Usual approaches include block diagrams, use cases, context diagrams and versions of the DoDAF; value and objective models; and prototyping, simulation and analysis reports.
  - **Software Architect:** Standard approaches include Kruchten’s 4+1 Views approach (Kruchten, 1995), SEI’s Views and Beyond approach (Clements et al., 2002), ANSI/IEEE Std 1471-2000 approach (IEEE, 2000).

- **How can architectural frameworks be used?** There seems to be a clear consensus (WICSA/ECSA, 2009) that architectural frameworks, e.g., TOGAF (The Open Group, 2009), GERAM (IFIP-IFAC Task Force, 1999), DODAF (Department of Defense, 2007), are helpful in some areas, but are neither necessary nor sufficient to capture a high-quality rendition of architecture in any kind.

The following sections explore the means how to partially resolve the main issues in aligning enterprise, system, and software architectures.
ARCHITECTURE ALIGNMENT

Architects (of any kind) need to understand the system, its environment, its stakeholders, their concerns, and the solution approach. The solution approach varies across kinds. In software architecture, the solution approach is software-oriented. For system architecture, the solution approach is hardware- and platform-oriented. For enterprise architecture, the solution approach is people- and business-oriented.

To capture the relationships among the kinds graphically, we suggest that simple class diagram. For example, the diagram would show that all types of architecture are influenced by stakeholders' needs and concerns. It would show that system architecture heavily influences software architecture, that system architecture influences enterprise architecture in the case of acknowledged EA, and so forth.

Consequently we decided to use a normal association relationship between the entities in a UML class diagram, shown in Figure 1.

Stakeholders of a system include management, users and the technical people who design and implement the systems. These stakeholders contribute to the three kinds of architecture in one way or another. Their interactions with the architecture design are iterative in nature: a user may need to change her requirements if technology constrains what can be achieved, or an architect may need to reconsider a design if new non-functional requirements arise. Architects in each of the architectural areas also influence each other’s decisions. For instance, software architects designing for software reliability needs the design support of system architects; enterprise architects designing for system integration requires software and system architects to align and synchronize their designs.

CROSSING ENTERPRISE, SYSTEM, AND SOFTWARE ARCHITECTURE BOUNDARIES

It is imperative that the architecting process is intertwined and interleaved between requirements and architectures: much of the requirements communicate stakeholders’ and the enterprise technical, business, environmental and strategic concerns. These concerns need to be realized and accommodated by the architecture of the software system. Architecting software systems is process, which crosscuts business, technical and strategic concerns. In today’s world of rapidly changing information technology, organizations, and marketplaces, the requirements tend also to

Figure 1. Relating enterprise, software and systems architectures
change, and in ways that affect the stability of the architecture.

The landscape for managing the co-evolution of requirements and architectures is becoming more complex; as a result, it is becoming a trend to describe various architectural views relating to the enterprise, software and systems perspectives. Such as an approach is believed to cater for “modularity,” “separation of concerns” and manage complexity through “decomposition”; it follows the trend towards heterogeneity in reasoning and assist in development and evolution of long-lived complex software systems. However, reconciling these views and managing their evolution is a problem. For example, it is challenging to prioritize these views, weight their importance and trace their concerns to relevant stakeholders and software artifacts. It is also challenging to manage the conflicts arising from inconsistencies in reconciling these views and co-evolving the views with the associated artifacts (and as the requirements change).

Current architectural practices, however, do not provide full support for traceability from the requirements specification to the architectural description related to these views (e.g., which and (how) requirement(s) in the requirements specification an individual architectural element relate to and satisfy and vice versa). Maintaining traceability “links” between these views is necessary for managing the change, the co-evolution of both the requirements and the architecture, confining the change, understanding the change impact on both the structure and the other requirements, providing a support for automated reasoning about a change at a high level of abstraction. Further, such traceability “links” make it easier to preserve the enterprise strategy, the acquired knowledge of the team, the architectural knowledge through guided documentation. For example, this may then minimize the impact of personnel losses and may allow the enterprise to make changes in the software system without damaging the architectural integrity (and making the software system un-evolvable, Figure 2).

Figure 2. Architecting for enterprise, software, and systems goals
We envision that the architecting process tends to follow an intertwined and interleaved phases encompassing iterative phases of inceptions, refinements and realisations spanning enterprise, software and systems goals and their associated viewpoints. In practice, the realisation is often distilled into an “implementable” architecture and associated architectural knowledge. Relating the architecture to various architectural viewpoints and associated architectural knowledge (through the architecture viewpoints management) enforces traceability and provide primitives for managing changes and evolution (through the Goal Management layer).

ARCHITECTURE PROCESSES, TOOLS, AND TECHNIQUES

The process of architecting can be through of as a translation of system requirements into a solution with consideration for additional constraints that may be imposed by the enterprise. This translation process does not happen as a specific, well-defined phase during system development but rather is ongoing as negotiations among stakeholders take place and vision of the system is refined and evolved. Throughout these activities the architecture is a central reference for stakeholders at all levels. Given the central role of architecture throughout the planning, development, and maintenance phases of a system’s life it is critical that it be a faithful representation of the current state of the system.

Much progress has been made over the past two decades and some consensus has been achieved as to what to capture, how it should be modelled, and what will constitute good documentation. Architecture description languages have been developed that provide support for describing the components and connectors of a system as well as describing both static and behaviour related properties associated with, or required of, these architectural elements (Feiler, Gluch, & Hudak, 2006; Magee, Dulay, Eisenbach, & Kramer, 1995; Object Management Group, 2003). Standards for architecture documentation have been defined (ISO/IEC, 2010), and templates have been provided that support just in time capturing of that information (Clements et al., 2002).

The process of capturing and maintaining architectural information is on-going as it will be the primary carrier of architectural decisions, which as we have note previously must be traceable to enterprise-wide elements, artifacts, and concerns. Not only must the relationships among architectural artefacts be identified and maintained but also their mappings to other elements in the enterprise. It is not possible for any manual process to support flexibility on this scale thus tools support must be provided to manage the architectural models as well as other supporting documentation and linkage. Automation is critical to this process and support tools must be integrated with other tools that support the enterprise (Figure 3).

INDUSTRIAL PRACTICES

Enterprise, Systems, and Software Architecture share many commonalities and many of their concerns overlap, but they also have marked differences in their focuses and their approaches. Presently in the IT industry, it is common to see each architecture discipline solving problems for some specific stakeholders, they use different technologies and employ different practices. The focus on their respective solutions has made each them inward looking. Alignment of these topics is to shift the focus from the solution domain to problem domain, and to be able to communicate goals and design problems across the architecture disciplines.

In recent years, researchers have started to emphasize design decisions and knowledge management as core elements of architectural design. All architectural solutions are the results of design decisions, but how do these decisions come about
and how are the decisions justified? Avgeriou et al. (Avgeriou, Kruchten, Lago, Grisham, & Perry, 2007) suggest that identifying design issues and design concerns are central to rationalizing a solution. Tang et al. (Tang, Liang, Clerc, & Vliet, 2011) not only support the notion of design problem as a key element in architecture design, they outline that architecture design problems and solutions should co-evolve. The goals and the design problems are not all known up-front when architecture design activities begin. Many of the design problems are discovered during design. The process of architecture design can be problem-driven if one does not understand the problem domain very well or it can be solution driven if the solutions the problems are well-known to the architects, it is usually both.

The communication of the design contexts, design problems and potential solutions across different architecture disciplines is an essential part of bridging the knowledge gap between them. As architecture design decisions often have wide ranging implications to different parts of the business and technical architectures, these implications, mostly often issues, cannot be detected easily without communicating knowledge from another perspective. Architects from the three architectural disciplines can bring about such a broader perspective with wider coverage of the different stakeholders’ views (Figure 4).

Figure 3. Traceability of architecture elements, artifacts, and concerns

Figure 4. Co-evolving architecture design problems and solutions
CURRENT CHALLENGES AND FUTURE DIRECTIONS

System and Software architecture are defined as “the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution” (ISO/IEC, 2010). Bass and co-authors (Bass, Clements, & Kazman, 2003) describes architecture as a description of system structures, its properties and a set of early design decisions. Some important and common properties are quality attributes of a system. The description of a system structure and its properties are not and cannot be definitive, they can be subject to interpretation by different people and under different circumstances. What is structural to an architect may be considered as implementation details to another architect. What is an important property to one architect may be considered as obvious to another.

The authors in this book have explored many issues in enterprise, system and software architectures. Methodologies and approaches for defining, improving and aligning the three types of architecture activities have been suggested. From these studies, a number of challenges have emerged:

• Clements has argued that it is useful to have some ambiguities of different kinds of architecture. This concept is further explored by Woods and Rozanski in which the roles of architects might be a focal point. An architect must determine the extent, in terms of system scope as well as the problem-solution domain, to which his/her role lies. Making an appropriate situated-decision given the ambiguity of architecture boundaries requires a sound judgment.
• Axelsson describes the architecting process and the maturity model of embedded systems. One of the challenges suggested is how to synchronize architectures at the enterprise, software and system levels during their evolution. Zimmermann and Miksovic also describe evolution in terms of the architecture guidance model. They describe the challenge of reusing a guidance model to extended projects. The original guidance and principles may no longer be applicable in the new situation.
• Muller models architectures in a hierarchy. He suggests that the heterogeneity, uncertainties and complexity of real-world architectures post challenges to practical architecting.
• Alfred describes architecture challenge in terms of the architecture of problems. He argues that the essential issue is problem identification. It is all too common for practitioners to apply analogous solutions that are unfit to the problem. When architectural decisions are made under these circumstances, they become difficult to change.

The chapters in this book have provided some insights to address these challenges. Though there are similarities between architectures, situations also vary from system to system. An observation is that there are always ambiguities and challenges in architectural design. No single method can address that. We rely on architects to apply the right principles, the right guidelines, and making the right decisions.

FUTURE RESEARCH DIRECTIONS BEYOND THIS BOOK

We reflect on some challenges and future directions, which future research into enterprise, software and system architectures should consider. We explicate our focus on change management, through maintaining, managing and evolving traceability across various artifacts. We also
motivate the need for future research in change impact analysis and associated suites explicating the interconnection between these artifacts.

**Change Management: Traceability of Requirements to the Software, System, and Enterprise Architectures**

An important outcome of the initial development of the software system is the knowledge that the development team acquires the knowledge of the application domain, user requirements, role of the application in the business process, solutions and algorithms, data formats, strength and weakness of the architecture, and operating environment. From inception to evolution, software, system, and enterprise architectures depend on this knowledge. Such knowledge is crucial prerequisite for the co-evolution of the three artifacts. In particular, these architectures and the team knowledge make evolution and co-evolution possible. These artifacts, to a great extent, allow making changes in the large-scale and complex software systems possible without damaging the integrity of the architecture. Once one or the other architecture disappears and following a similar argument to that of (Bennet & Rajlich, 2000), the system will be difficult to evolve and enters the stage of servicing (also referred to as maturity by Lehman) (Bennet & Rajlich, 2000). At the servicing stage, only small tactical changes would be possible. For the business and the enterprise, the software is likely to be no longer a core product and the cost-benefit of the change becomes marginal. Quoting from Bennet and Rajlich (2000): “There is a positive feedback between the loss of software architecture coherence and the loss of software knowledge.” Less coherent architectures requires more extensive knowledge in order to evolve the system of the given architecture. However, if the knowledge necessary for evolution is lost, the changes in the software will lead to faster deterioration of the architecture. Very often on software projects, the loss of knowledge is triggered by loss in key personnel and the project slips into the servicing stage (Bennet & Rajlich, 2000). Hence, planning for evolution and stable software architectures urges the need for traceability techniques, which traces requirements and their evolution back and forth into the software, system and enterprise architecture; such traceability is believed to aid evolution, co-evolution of various artifacts and tend to “preserve” the team knowledge.

Davis (1993) gives the earliest definition of traceability. Davis defines traceability as “the ability to describe and follow (track) the lifetime of an artifact, in both a forward and a backward direction, i.e., from its origin to development and vice versa” (Davis, 1993). Gotel & Finkelstein (1995) have preserved the spirit of Davis’s definition of traceability. They, however, have scoped the definition on tracing a requirement through its “life.” The requirement life covers periods of a requirement origin, development and specification, deployment, use, and on-going refinement. They have defined requirements traceability as “the ability to describe and follow the life of a requirement in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases)” (Gotel & Finkelstein, 1995) have particularly discussed the importance of tracing requirements back to their source. These sources might be people, other requirements, documents, or standards.

Traceability is important for modeling dependencies among software objects and for managing the change across software artifacts. Traceability information records the dependencies between requirements and the sources of these requirements, dependencies between requirements themselves, and dependencies between requirements and the system implementation (Kotonya & Sommerville, 1998). Advances in software-development envi-
ronments and repository technology have enabled software engineers to trace the change in software using traceability techniques. According to (Gotel and Finkelstein, 1995), these techniques span a variety of approaches ranging from cross-referencing schemes (e.g., cross-referencing schemes, based on some form of tagging, numbering, indexing, traceability matrices, and matrix sequences), through document-centered techniques (e.g., Templates, hypertext, and integration documents), to more elaborate structure-centered techniques (e.g., assumption-based truth maintenance networks, constraint networks, axiomatic, key phrase, relational dependencies and/or social networks).

We define requirements to architectural traceability as the ability to describe the “life” of a requirement through the requirements engineering phase to the architecture phase, including systems, software and enterprise architectures, in both forwards and backwards. Forwards demonstrates which (and how) architectural element(s) satisfy an individual requirement in the requirements specification. Backwards demonstrates which requirement(s) in the requirements specification an individual architectural element relate to and satisfy. Current architectural practices, however, do not provide a support for traceability from the requirements specification to various architectural descriptions related to systems, software and enterprise (i.e., which and (how) requirement(s) in the requirements specification an individual architectural element relate to and satisfy and vise versa). Maintaining traceability “links” is necessary for managing the change, the co-evolution of both the requirements and the architecture artifacts, confining the change, understanding the change impact on both the structure and the other requirements, providing a support for automated reasoning about a change at a high level of abstraction. Further, such traceability “links” make it easier to preserve the acquired knowledge of the team through guided documentation. This may then minimize the impact of personnel losses, and may allow the enterprise to make changes in the software system without damaging the architectural integrity and making the software system unevolvable.

Change Impact Analysis Crosscutting System, Enterprise, and Software Architectures

Although change impact analysis techniques are widely used at lower levels of abstractions (e.g., code levels) and on a relatively abstract levels (e.g., classes in O.O. paradigms), little effort has been done on the architectural level (i.e., architectural impact analysis). Formal notations for representing and analyzing architectural designs generically referred to as Architectural Description Languages (ADLs) have provided new opportunities for architectural analyses (Garlan 2000). Examples of such analyses includes system consistency checking (Allen & Garlan, 1994; Luckham et al., 1995), and conformance to constraints imposed by an architectural style (Abowd et al., 1993).

Notable effort using dependency analysis on the architectural level includes the “chaining” technique suggested by Stafford and Wolf (2001). The technique is analogous in concept and application to program slicing. In chaining, dependence relationships that exist in an architectural specification are referred to as links. Links connect elements of the specification that are directly related. The links produce a chain of dependencies that can be followed during analysis. The technique focuses the analysis on components and their interconnections. A component may have a set of input and output ports (which correspond to the component’s interface). These ports may have been connected to one another to form a particular architectural configuration. Communication between components is accomplished by sending events to the component’s ports. Stafford and Wolf (2001) supports the approach with an analysis tool, Aladdin. Aladdin accepts an architectural
specification as input. A variety of computations can be then performed. The computations include unconnected component identification, change impact analysis (i.e., which components will be affected by an architectural change), and event dependence analysis (i.e., which components can send the following event to this port). These computations start at a particular component and/or port. Forward and/or backward chaining are then performed to discover related components. Forward and backward chaining is analogous in concept to forward and backward walk in the data-flow slicing. The applicability of this technique is demonstrated on small-scale architectures and could be extended to address current architectural development paradigms. For example, how such a concept could be refined to perform what-if analysis on large-scale software architectures describing enterprise, system and software? This is necessary for reasoning about how the change could impact the commonality, variability, cross-cutting concerns and their interdependence across various artifacts. These techniques could be then complemented by analysis tools, which could facilitate automated reasoning and provide a basis for what-if analyses to manage the change across instances of the core architecture. Understanding how the change could then ripple across different architecture artifacts might be feasible. These techniques could be complemented by automated reasoning to manage evolution and co-evolution of systems, software and enterprise architectures. When combined with traceability links, the combination could provide a comprehensive framework for managing the change and guiding evolution.

REFERENCES


