Chapter I
Towards a Systematic Method for Solutions Architecting

Tony C. Shan
Bank of America, USA

Winnie W. Hua
CTS Inc., USA

ABSTRACT

Solutions architecting method (SAM) is defined as a methodical approach to dealing with the architecture complexity of enterprise information systems in IT solution designs. This comprehensive method consists of eight interconnected modules: framework for e-business architecture and technology, prescriptive artineering procedure, technology architecture planning, architecture stack and perspectives, rapid architecting process, architecture readiness maturity, generic application platform, and Tao of IT development & engineering. Collectively, these modules form a holistic discipline guiding the process of developing architectured solutions in an enterprise computing environment. Several unconventional concepts and thinking styles are introduced in this overarching structure. This systematic method has been customized and adapted to be extensively applied in one form or another to develop various IT solutions across a broad range of industrial sectors. Reference solutions are presented and articulated to illustrate the exemplary implementations of some key elements in SAM. Best practice and lessons learned as well as future trends are discussed in the context.

INTRODUCTION

The e-business models in today’s fast-paced on-demand business world mandate increasing flexibility of information systems applications. It is compulsory for the information technology (IT) group to provide a higher level of services at a lower cost for the business to compete and succeed in a globalized economy. The reality is that IT must build more complicated, flexible, scalable, extensible, and forward-thinking technical solutions, to satisfy the ever-growing business needs.

In large organizations like worldwide financial institutions, virtually hundreds, if not thousands of IT applications and systems have been built, acquired, or purchased through the years, to provide both external customers and internal employees with
Towards a Systematic Method for Solutions Architecting

reliable electronic services, utilizing heterogeneous technologies and architectures to satisfy diverse functional requirements from different lines of business. In the financial services industry, the banking business processes generally involves different business divisions that address retail, commercial, investment, wealth management, treasury, and capital markets. In particular, services are delivered via different channels. To effectively manage the architecture assets and design top-quality IT solutions in such a diverse environment, a highly structured methodology is of critical importance to achieve an array of goals—separate concerns, divide responsibilities, encapsulate the complexity, utilize patterns, leverage best practices, control quality, ensure compliance, and establish execution processes.

BACKGROUND

The computing paradigm has gone through several generations of evolution in the last five decades: monolithic, client/server, multi-tier, structured methods, object-oriented, component-based, service-oriented, and event-driven model. The overall solution architecture has become increasingly complicated and thus hardly manageable through a traditional waterfall process. Previous work over the past few years has strived to address the complexity issue in the architecture design and process. A pioneer effort in this space was the Zachman framework (Zachman, 1987), which is a logical structure to classify and organize the descriptive representations of an enterprise computing environment, which are important to the development of the enterprise systems and the enterprise management. In a form of a two-dimensional matrix to symbolize the enterprise architecture environments, it has achieved a substantial level of penetration in the domain of business and information systems architecture as well as modeling. Though it is primarily used as a planning or problem-solving tool, it tends to implicitly align with data-driven and process-decomposition methods and processes, and it operates above and across the individual project level. A similar approach is taken in the extended enterprise architecture framework (E2AF) (IEAD, 2004) with a scope of aspect areas containing business, information, system, and infrastructure in a 2-D matrix. Rational unified process (RUP) (Kruchten, 2003) overcomes these shortcomings by taking a use-case driven, object-oriented and component-based approach, using a standard notation—unified modeling language (UML). The concept of 4+1 views offers multi-perspective interpretations of the overall system structure. RUP is more process-oriented, and to some extent is a waterfall approach. RUP has little to address software maintenance and operations, and lacks a broad coverage of physical topology and development/testing tools. It generally operates at the individual project level. Enterprise unified process (EUP) (Nalbone, 2005) attempts to extend the RUP to cover the entire IT lifecycle. An open source unified process (OpenUP/Basic) is also under development in Eclipse (OpenUP, 2007).

Another heavyweight approach, the open group architecture framework (TOGAF) (Open Group, 2007), is a comprehensive framework with a set of supporting tools for developing an enterprise architecture to meet the business and information technology needs of an organization. The three core parts of TOGAF are architecture development method (ADM), enterprise architecture continuum, and resource base. The scope of TOGAF covers business process architecture, applications architecture, data architecture, and technology architecture.

All these approaches are heavyweight methodologies, which require a fairly steep learning curve to get started. On the other hand, model-driven architecture (MDA) (OMG, 2007) takes a lightweight approach. MDA aims to separate business logic or application logic from underlying platform technology. The core of MDA is the platform-independent model (PIM) and platform-specific model (PSM), which provide greater portability and interoperability as well as enhanced productivity and maintenance. The primary focus of MDA is on software modeling in the development life-cycle process.

Quite a few agile methods are available such as extreme programming (XP), dynamic systems development method (DSDM), agile modeling (AM), feature driven development (FDD), crystal, adaptive software development (ASD), scrum, and
Towards a Systematic Method for Solutions Architecting

test-driven design (TDD). Agile methods typically fit well with small- or medium-size projects developed by highly-skilled people. It is noteworthy that in DSDM (DSDM, 2007) time is fixed for a project, and resources are fixed as far as possible. The consequence of these restrictions in DSDM is that the requirements have to be allowed to change.

In the business process management (BPM) space, the business process management initiative (BPMI) organization (BPMI, 2007) has been working to develop open, complete, and royalty free XML-based BPM standards such as business process modeling language (BPML), business process modeling notations (BPMN), and business process query language (BPQL), but there have been only a handful of implementations so far. The business process execution language (BPEL) is specified by the OASIS group and has been gradually endorsed by many vendors. The extended business modeling language (xBML) (Business Genetics, 2007) differs from other approaches in that it dissects the business into separate and distinct “atomic” dimensions and then reintegrates these individual elements to produce a representative model of the entire business process that supports the business purpose. This proprietary language is only used in the tools supported by the vendor. A lack of interoperability is a big challenge in the BPM space.

Other related work on enterprise architecture frameworks is largely tailored to particular domains. These are useful references when an IT group creates its own models for the organization. The C4ISR architecture framework (DoD, 1997) provides comprehensive architectural guidance for the various commands, services, and agencies within the United States Department of Defense, with the purpose of ensuring interoperability and cost effectiveness in military systems. The treasury enterprise architecture framework (TEAF) (Treasury Department, 2000) gives the direction and guidance to all bureaus and offices of the treasury department for structuring enterprise architecture. The federal enterprise architecture (FEA) framework (Federal Office, 2007) aims to guide the planning and development of enterprise architecture in U.S. federal agencies. The Purdue enterprise reference architecture (PERA) (Purdue University, 2007) is aligned to computer integrated manufacturing. ISO/IEC 14252 (IEEE Standard 1003.0-1995) (IEEE, 2007) is built on open systems standards as an architectural framework. Aiming at standardizing the open distributed processing, the ISO reference model for open distributed processing (RM-ODP) (Putman, 2001) is a coordinating framework that integrates the support of distribution, interconnections, portability, and interoperability, with four elements and five viewpoints.

In recent publications on the architecture process, practices, planning, patterns, methods, and platforms, the majority of these studies (Shan, 2004; Wada, 2006; Shan, 2006) focus on the high level or specific layers of enterprise architecture.

A new architecting model is introduced in the next section, with the key artifacts and features of various modules described in the subsequent sections, followed by the future trends and conclusion sections.

COMPREHENSIVE APPROACH

As discussed in the foregoing section, most of the previous methods reveal the architectural aspects of an information systems application to some extent from a single viewpoint or limited perspectives. The necessity of a comprehensive approach to architecting the end-to-end IT solutions becomes more and more evident, demanding a systematic disciplined way. A highly structured method is thus designed in this chapter to meet this ever-evolving need, and present a detailed and holistic view of all architectural elements, components, knowledge, platforms, planning, practices, and their interrelationships. Design processes are established accordingly in this approach to facilitate the creation, organization, and management of the architecture assets and solutions at different levels in a large organization.

Design Philosophy

The following design principles and philosophies are applied in developing the disciplined method, partly adapted from TOGAF (Open Group, 2007) but significantly modified/expanded to tailor to the services-oriented architecting process.
Towards a Systematic Method for Solutions Architecting

1. Business Principles
   a. **Primacy of principles:** All groups and stakeholders within an organization must follow these principles of solutions architecting.
   b. **Maximize benefits:** Maximum benefits will be achieved to the entire organization at the corporate level.
   c. **Business continuity:** Business operations are not interrupted in spite of system changes.
   d. **Active engagement:** All stakeholders that participated in the process will accomplish business objectives.
   e. **Compliance with regulations:** The architecting processes comply with all relevant regulations, policies, and laws.
   f. **IT accountability:** The IT group accounts for owning and implementing IT solutions, platforms, infrastructure, and processes to ensure that the deliverables satisfy business requirements for functionality, quality, service levels, cost, and delivery timelines.
   g. **Innovations:** The stimulation and protection of the corporate innovations is enforced in the IT architecture, management, standardization, planning, and governance processes.

2. Technical Principles
   a. **Flexibility:** The technical model is agile and nimble to be adaptive in response to future business needs.
   b. **Responsive change management:** Modifications to the corporate architecture/infrastructure environment are planned and implemented in a phased approach.
   c. **Requirement scope control:** Manage scope creeping effective.
   d. **Iterative process:** Use incremental development method rather than a waterfall approach.
   e. **Technology standardization:** Technological diversity is controlled to minimize immature and proprietary solutions and products.
   f. **Interoperability:** Software, hardware, network, and infrastructure must conform to industry and corporate standards that promote compatibility for applications, services, communications, integration, data, security, and technology.

3. Solution Principles
   a. **Ease of use:** Solutions are user friendly, with the underlying technology transparent to users, so they can concentrate on tasks at hand.
   b. **Technology independence:** Technical solutions are independent of specific technology/platform selections and decision, ensuring portability on different technology platforms.
   c. **Common services and components:** Minimize the redundant development of similar functionalities to promote common service and components across the organization.

4. Data Principles
   a. **Data asset:** Data is a corporate asset that has value to the enterprise and is managed accordingly.
   b. **Data ownership:** Each data element owned by an entity accountable for the data quality.
   c. **Common vocabulary and metadata:** Data is defined consistently throughout the organization, following common business taxonomy, and the metadata are standardized and accessible for all relevant users.
   d. **Shared data:** Data is shared across lines of business for individual applications and systems to perform their duties.
   e. **Data access:** Authorized users can access relevant data to perform their functions.
   f. **Data security:** Data is protected from unauthorized access, disclosure, use, and distribution. Regulatory requirements are strictly enforced. Sensitive and proprietary information is also protected,
Towards a Systematic Method for Solutions Architecting

particularly in international business, to comply with regional rules and laws.

Architecting Method

The solutions architecting method (SAM) is designed in this chapter as a systematic approach to developing and managing architecture of IT solutions. It utilizes a variety of conventional and unconventional problem-solving techniques such as the divide-and-conquer, recursive decomposition, and strengths-weaknesses-opportunities-threats (SWOT) strategies to abstract concerns, separate responsibilities, and manage complexities at different levels and from various standpoints. SAM is a methodical way to develop IT solution architecture, organize and visualize the architectural artifacts, identify static architectural aspects, specify dynamic architectural behaviors, and further help verify and optimize the strategy, resources, process, systems, applications, tools, and technology. As illustrated in Figure 1, SAM comprises eight key modules:

- **Framework for e-business architecture and technology (FEAT):** A 4-pillar framework for solution architecture design and governance.
- **Prescriptive artineering procedure (PAP):** A prescriptive method to combine both art and engineering approaches to designing and developing IT solutions.
- **Technology architecture planning (TAP):** A systematic approach to construct the IT blueprint and roadmaps with strategic alignment and agility.
- **Architecture stack and perspectives (ASAP):** A multi-perspective pyramid model for architectural artifacts and views.
- **Rapid architecting process (RAP):** A multidimensional procedural approach to develop orthogonal models in IT solutions.
- **Architecture readiness maturity (ARM):** An analysis method to assess architecture lifecycle (review-refactoring-reengineering-rearchitecting) and levels of capability maturity.
- **Generic application platform (GAP):** A modular platform addressing prominent development facets as an IT solution baseline.
- **Tao of IT development & engineering (TIDE):** A holistic best-practice model as guidance to effective IT solution constructions.

SAM MODULES

The modules in the SAM model are articulated in great detail in the next few subsections.

Framework for E-business Architecture and Technology (FEAT)

The framework for e-business architecture and technology is a high-level service-oriented framework for architecting IT solutions, composed of 4 core pillars—design methodology, systematic process, solution patterns, and integrated maturity & performance management, as shown in Figure 2.

The design methodology pillar deals with the approaches used to dissect business requirements and design architectures of IT solutions. The systematic process pillar specifies the step-by-step procedures in planning, designing, and maintaining architectural assets. The solution patterns pillar covers architecture models, common platforms, and reference solutions. The integrated maturity & performance management pillar is with regard to managing and assessing the maturity and performance of the artifacts in the framework.

FEAT is a foundation of the core scope of SAM. The modules discussed in the subsequent sections all fall into this overarching framework. In the design methodology pillar, practical methods are defined to consolidate the key techniques used in various schools of methodologies and to migrate the existing approaches to the next level, combining the art, engineering, and science disciplines. Based on these methods, pragmatic processes are constructed in the systematic process pillar, to form a cookbook approach to executing the methods in a step-by-step procedure. The solution patterns pillar serves as a
repository of best practices templates and standards. Anti-patterns are also cataloged to capture lessons learned and pitfalls to avoid in the solution design. The integrated maturity & performance management pillar bridges the system quality requirements with the overall architecture aspects. The quality attributes are measured qualitatively and, more importantly, quantitatively in a repeatable way.

**Prescriptive Artineering Procedure (PAP)**

The great majority of the architecting processes today are in the form of an art. UML provides a mechanism to document the key artifacts in the application development in a standardized way. However, UML is largely limited to the software part of an IT solution, primarily towards object-oriented analysis and design. Some research work has been conducted to extend the UML to support other architectural concepts, but there has been little standardization that combines these activities into a single extension. The current design and documentation of IT architecture are more or less free-form, using a wide variety of illustrative “box-and-line” diagrams and formats. Multiple efforts were funded by DARPA to develop architecture description languages (ADLs), but many of the ADLs are in the format of programming languages rather than specification languages. Portability and interoperability are the hurdles to overcome as no standardization exists to combine results from one description into another or to verify a representation in one language by recasting it in another language.

To make the matter worse, due to the lack of standards in architecture descriptions, there is no single standardized method to be applied to architect an IT solution. The consequence becomes twofold. First, almost every application has a “unique” architecture designed and documented in distinctive ways. The applications may generally be classified to categories at a very high level. Some common grouping criteria are: Java EE or .NET applications, Struts or JSF as the MVC framework, and Hibernate/Cayenne or Java Persistence API as object-relational mapping technology. Nonetheless, it is extremely hard to conduct apple-to-apple comparisons at this conceptual level. Second, the development procedures are to
Towards a Systematic Method for Solutions Architecting

a large extent spontaneous and ad hoc, behind the perceptive rationale that each application development environment has its own “individual” needs. Making objective assessments and quantifying the effectiveness measurements of the development procedures are out of the question.

A prescriptive method is proposed, which is adapted and expanded from our previous work (Shan, 2006a), to standardize the solution development in both art and engineering approaches. The method is coined as prescriptive artineering procedure, which implies a combination of art and engineering disciplines in the design practice.

This comprehensive solution method is composed of the following key components, as shown in Figure 3:

- Service patterns
- Architecture process
- Hybrid design methodology
- Service-oriented enterprise model
- Service-oriented solution platform

The five components of this method address the primary focus areas in the solution design. The service patterns map the application requirements to a category in the problem domain for a pattern-based analysis and high-level modeling. The architecture process defines a multi-layer structure to abstract architecture complexity and cope with the architecture assets and responsibilities in a systematic manner. The hybrid design methodology provides a balanced approach to design a flexible and agile technical solution model. The service-oriented enterprise model presents a holistic view of the cross-portfolio and cross-application IT service model with service-oriented architecture (SOA), service-oriented process (SOP), service-oriented integration (SOI), and service-oriented management (SOM). The service-oriented solution platform offers a common baseline for all application solutions in a particular vertical market sector.

The hybrid design methodology utilizes both top-down and bottom-up approaches to take advantage of the benefits of the two schemes. The composite services & process layer is an important bridge between the two camps, as illustrated in Figure 4. Another key in this methodology is the mandatory deliverable of a platform-independent technical model, on which the platform-specific technical model is built and based, but not the opposite way around. In addition, existing applications can be service-enabled and migrated via pattern-based techniques—wrapping, decomposing, consolidating, and transforming, rather than being thrown away in order to transit to a SOA paradigm.

This comprehensive method has been successfully applied in the e-banking business domain to derive a service-oriented architecture model for the Internet-channel financial services space (Shan, 2004).

Technology Architecture Planning (TAP)

Figure 5 shows a triangle model for architecture planning. The simplicity of the model is its usefulness and strength of abstraction at a high level. The “as-is” element reveals the current architecture...
we have built or acquired. The “to-be” element represents the future architecture we wish to have. The third element is all of the people who have impacts on or are significantly affected by the architecture changes. The stakeholders include the business users, analysts, architects, developers, testers, system administrators, DBAs, subject matter experts, engineers, specialists, vendors, standards organizations, and legal entities. They may be for a change, be against it, be neutral, or not even know about it yet. Due to the different interests, relevant expertise, and roles played from various parties, there are conflicting forces and factors in the decision-making process. And it is not uncommon that wrong tools are inadvertently misused from time to time in investigating various aspects in different domains by different parties. It is, however, crucial to leverage appropriate tools to look into the details of the elements for analysis and synthesis. The analogy illustrated in the diagram demonstrates that, when evaluating the “to-be” element, a binocular is needed to see things in the distance, whereas a telescope must be used to observe things further far away. Likewise, in order to inspect greater details of the “as-is” element, a microscope is necessary as a magnifying glass would not reveal the fine points at the level of granularity desired. Moreover, the internal structure may have to be disclosed via an X-ray or MRI image. The metaphor indicates that we need not only “do things right,” but also “rely on right tools to help do things right.”

There is a fourth element in the model—balanced approach, which is the architecture planning that balances the pros and cons, and resolves the conflicts with tradeoffs and compromises to lay out a reasonable path to the needed changes.

Based on this triangle model, a quadrant is constructed to identify the key artifacts in architecture planning, as illustrated in Figure 6. The “as-is” element is mapped to the current state assessment, which collects the input data and analyzes the existing environment. In the same way, the “to-be” element is mapped to the future state definition. The future state specification serves as a blueprint of the desired form. The gap analysis is subsequently conducted to bridge the two states. A roadmap is thus defined to transit the current state to the future state in a

Figure 3. Prescriptive artineering procedure
controlled manner. Usually, the changes are staged in multiple phases in an incremental fashion. Depending on the timeframe desired, the future state vision is typically strategic and targeted to the long term. However, tactical solutions are often needed to deliver required business functionalities in a short period of time. It is of vital importance to have the future state well defined, so that the strategic goals are not deserted when pragmatic approaches are crafted for the short-term goals.

This approach serves as a use-case driven pattern-based procedure for iterative architecture planning. Round-trip architecture engineering is enforced to articulate the multi-perspective views, using the industry standards on visual modeling and semantic ontology, such as UML, SysML/DSL, BPMN, BPEL, and OWL. Loosely-coupled interface and integration are imposed in the principle of design by contract. The 80-20 rule is leveraged in the use case drill-down for objective sizing and empirical estimation in the service-centered paradigm.

To provide better abstraction and individual agility, the solution domain is divided into two areas: business process domain and IT solution domain. The business process domain captures the information pertinent to the static structure on which the business runs—the business process, the operations method, the dynamic process, the data requirements, and the usage scenarios. The IT solution domain describes the IT systems supporting and realizing the business process. The key artifacts in the IT solution domain are application domain pattern, conceptual architecture, architectural style, logical architecture, and technology model. The gap between the business model and IT model is daunting in almost every large organization. There have been constant conflicts in the alignment of IT models with business models due to the delivery schedule, resources, skillset, risks, and budgets. The strategy to alleviate this pain is to seamlessly integrate the models in these two areas. The engaging of end-to-end round-trip engineering principles with traceability and auditability is a necessity to make IT models nimble and adaptive.

A pragmatic procedure is further defined in Figure 7 to facilitate the current state evaluation and future state specification. There are eight steps in the procedure:

- Operations model
- Scope/functional analysis
- Use case model
- Business process flow
- Business data requirements
Towards a Systematic Method for Solutions Architecting

Application/architecture pattern
Conceptual model
Technology model

Last but not the least, the SWOT technique is employed to conduct a thorough analysis on all major factors and options. The analysis reveals the requirement conflicts and design constraints as well as presumptions and limitations from a particular stakeholder’s point of view. Focus on a single factor is meant to minimize risks only from that attribute perspective at a fine-grained level. This is often insufficient when a variety of conflicting attributes play an equivalently important role or have non-distinctive impacts in the system, so that a balance between these interacting attributes must be established. The outcome of the SWOT exercise is the key to the unbiased justification of tradeoffs in a balanced approach. The harmonious approach resolves the conflicts in local/individual attributes to reach a global optimization.

Architecture Stack and Perspectives (ASAP)

Various architectures have been used in application design practices, such as application architecture, data architecture, network architecture, and security architecture. The need for an organization scheme of multiple architectures within the enterprise is evidently indispensable as the scheme represents progressions from logical to physical, horizontal to vertical, generalized to specific, and an overall taxonomy. The architecture stack in the technology and information platform (TIP) model, shown in a pyramid shape in Figure 8, provides a consistent way to define and understand the generic rules, representations, and relationships in an enterprise information system. It represents taxonomy for classifying architecture artifacts as an aid to organizing reusable solution assets. It assists communication and understanding, within enterprises, between enterprises, and with vendor organizations. It has occurred frequently that IT professionals talk at cross-purposes when discussing architecture because they are referencing different points in the architecture stack at the same time without realizing it. The stack helps avoid these misunderstandings and miscommunications.

The generic architecture stack (GAS) in the TIP model comprises seven interrelated layers:

- Layer 1—enterprise business architecture
- Layer 2—enterprise technical architecture
Towards a Systematic Method for Solutions Architecting

Layer 3—cross business-line architecture
Layer 4—channel specific architecture
Layer 5—application solution architecture
Layer 6—aspect-oriented architecture
Layer 7—component technology architecture

The TIP model also provides multi-perspective views of the architecture assets in a large organization from both business and technical standpoints. The contextual spectrum is depicted in Figure 9, which contains four core parts: process, abstraction, latitude, and maturity (PALM). The process perspective covers operations, risk, financial, resources, estimation, planning, execution, policies, governance, compliance, organizational politics, and so forth. The abstraction perspective deals with what, why, who, where, when, which, and how (6W+1H). The latitude perspective includes principles, functional, logical, physical, interface, integration & interoperability, access & delivery, security, quality of services, patterns, standards, tools, skills, and the like. And the maturity perspective is about performance, metrics, competitive assessment, scorecards, capacity maturity, benchmarks, service management, productivity, gap analysis, transition, and so on.

The abstraction perspective presents a high-level overview of the key artifacts for each architecture layer:

- What—definition of the architecture
- Why—value proposition and benefits
- Who—practitioners
- Where—usage scenarios and patterns
- When—time-based sequence and maturity
- Which—information and data dealt with
- How—approach and tools

Rapid Architecting Process (RAP)

The rapid architecting process (RAP) is designed based on the TIP model discussed in the preceding section. The focus of RAP is meta architecture, conceptual architecture, service architecture, systems architecture, information architecture, and software architecture, as shown in Figure 10. There are four dimensions defined in the process:
Towards a Systematic Method for Solutions Architecting

requirement analysis, specification, validation, and planning (RSVP). The requirement analysis is to identify the architectural requirements. The specification dimension is about designing the architecture based on the outcome of the analysis results and producing the architecture specifications. The validation dimension is for objective/subjective verification and certification of the architecture design completed. Lastly, the focus of the planning dimension is the generalization and optimization of the architectures in a domain and across portfolios.

Further, the process is broken down to eight interrelated steps:

- Step 1—meta-architecture
- Step 2—conceptual architecture
- Step 3—logical architecture
- Step 4—physical architecture
- Step 5—deployment architecture
- Step 6—data architecture
- Step 7—aspect-oriented architecture
- Step 8—component architecture

1. **Architecture requirement analysis:** The requirements of meta-architecture are architecture vision, principles, and trends, as well as the alignment of business and technology models. The conceptual architecture deals with modeling the system functionalities, identifying non-functional requirements, determining the service-level agreements, and analyzing the business process. The scope of the logical architecture is the overall IT system structure,
Towards a Systematic Method for Solutions Architecting

technology options and selection criteria, and application domain modeling. The physical architecture is about platform selection criteria, product mapping approach, capacity planning, and quality attribute analysis. The deployment architecture is with regard to the run-time environment to host the individual applications. The staging requirement of releases such as alpha and beta testing are in conjunction with various QA tests. A pilot may be required for large system rollout in the production environment. A recovery plan is important to provide business continuity in case of disasters. Data architecture handles the data and content requirements such as persistence, reporting, data integration, data quality, analytics, business intelligence, and data extract-transform-load. Aspect architecture treats crosscutting concerns as aspects in software using the architectural patterns. Component architecture is at the level of detailed design and implementation in application software. The analysis of the component needs helps identify reusable components and toolkits to be leveraged, either commercial off-the-shelf (COTS) or open source.

2. **Architecture specification:** The meta-architecture specifies a number of key architecture artifacts across the application portfolio: architecture mission and vision, strategy, roadmap, methods, tools, frameworks, cross application architecture models, architectural patterns, business analysis patterns, and standards. The deliverables of the conceptual architecture design are use case diagram, use case specification, system scope, architecture constraints and assumptions, architecture risks, user interface specification, and service access specification (API and service interface). In the logical architecture, the design is specified via the architecture diagram, subsystem responsibilities and relationships, communication diagram, activity diagram, key technology selection, architecture layering, and interoperability. The physical architecture captures the platform selection, hardware model, system management, system backup and disaster recovery, scalability, high availability, authentication & authorization, security, and

![Figure 9. Contextual spectrum in TIP model](image-url)
Towards a Systematic Method for Solutions Architecting

network requirement. The key elements in the deployment architecture are the network topology, hosting farm or hotel, deployable build models, development/test/production environments, configuration specification, deployment diagram, installation specification, user guide, and data center operations. The data architecture pertains to data model, storage, database engine, database schema, replication, data transfer, data backup, data caching, reporting, meta-data, query, workload priority, and access control. The aspect architecture specifies the application framework, design patterns, sequence diagram, collaboration diagram, performance tuning, session management, and open source products. Finally, the component architecture covers the component specification, CRC cards, class diagram, design patterns, technology, package, and open source utilities.

3. **Architecture validation:** The architectural artifacts as deliverables in the specification stage are verified, typically in a certification process. For meta-architecture, enterprise standards and policies are established, coupled with reference models and solution patterns. Prototype and proof-of-concept evaluations are used to justify the feasibility of new technologies and products. Best practice guidelines and development cookbooks are created as guiding documentation for the project teams. In the conceptual architecture, the business patterns and skeletal architecture are assessed to conduct the impact analysis. The focus of the logical architecture validation is on the architectural patterns, business rules, workflow, integration patterns, and reference models. A key decision justification in the physical architecture is build versus buy. Enterprise standards are checked with regard to the products used in the projects. In the deployment architecture, the deployment patterns should have been considered as the engineering implementation approaches. Data, aspect, and component related artifacts are validated in the respective architecture.

*Figure 10. Rapid architecting process*
Towards a Systematic Method for Solutions Architecting

Last but not the least, the compliance with regulatory requirements and industry standards are investigated across all these architecture models.

4. **Architecture planning**: Architecture planning is for both individual application architecture and portfolio architecture. The granularity can be at the enterprise, CIO, channel, and business domain levels. Benchmarks are used to measure and track the maturity of architecture models. The planning emphasizes the architecture reusability and generalization, which lead to architectural styles and patterns for an enterprise computing environment. The TAP section presented earlier specifies a pragmatic method to conduct effective architecture planning.

Table 1 is a 2-D matrix, which summarizes the key artifacts in the RAP. The matrix serves as a blueprint of key activities defined in the architecting process. It may also be used to verify the deliverables produced in each individual architecting step.

**Architecture Readiness Maturity (ARM)**

An application architecture has a lifecycle, similar to a product lifecycle but at a different level. The relationship of the architecture lifecycle to system lifecycle is analogous to that of an engine to a car. Comprehensive assessment of the application architecture to identify the positioning of the architecture maturity in its lifecycle is decisively important.

A review-refactoring-reengineering-rearchitecting (R4) process is designed to facilitate a comprehensive technical analysis of the application and consequently recommend the next step in the lifecycle. Figure 11 illustrates this process model. The review course provides a complete technical assessment to clearly identify where an existing application stands in its lifecycle. Based on the review results, three methods are leveraged to revamp the application: refactoring, reengineering, and rearchitecting. Refactoring is useful to improve an application without dramatic structural changes. It also helps reverse-engineer an application to recover the technical model, if not previously created in the forward engineering process, which is not uncommon in old applications that lacked disciplines in development. Reengineering is a valuable solution to migrate an application to next-generation technologies with no major architectural restructuring. It is sometimes the most cost-effective way to rebuild legacy systems. Rearchitecting is necessary if an application almost reaches the end of its lifecycle in terms of its technical model and architecture maturity. Replatforming usually falls in between reengineering and rearchitecting.

In addition, other techniques are complementary in assessing the architecture capability maturity, such as CMMi and six sigma. The capability maturity of a solution architecture can be rated to one of the five levels, similar to the CMMi model in principle:

- **Level 1—ad-hoc**: No common format and deliverables are defined. The designers differ from each other drastically in their descriptions produced.
- **Level 2—pattern-based**: Reference architecture is established as architectural models, which serve as templates for solution designs. Domain-specific patterns are identified.
- **Level 3—standardized**: Industry standard methodologies are employed to specify a solution architecture, with an aid of modeling tools.
- **Level 4—portfolio-based**: Connections and relationships of different architectures inside a domain and between the portfolios are identified and controlled. Management tools are leveraged to monitor the architectural assets and demands.
- **Level 5—optimized**: The individual solution architecture and the architectural models within a domain are reconstructable, quantifiable, and consultable at runtime, with iterative and adaptive generalization and optimization in a governable fashion.

The R4 process and the maturity levels are useful to assess the state of a particular application and/or
<table>
<thead>
<tr>
<th>Table 1. RAP matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component Architecture</strong></td>
</tr>
<tr>
<td>Construction transition</td>
</tr>
<tr>
<td>Requirement</td>
</tr>
<tr>
<td>Conceptual</td>
</tr>
<tr>
<td>System level</td>
</tr>
<tr>
<td>Software level</td>
</tr>
<tr>
<td>Data Architecture</td>
</tr>
<tr>
<td>Application platform</td>
</tr>
<tr>
<td>Component architecture</td>
</tr>
<tr>
<td>Aspect Architecture</td>
</tr>
<tr>
<td>Requirement</td>
</tr>
<tr>
<td>Functionality modeling</td>
</tr>
<tr>
<td>Core build</td>
</tr>
<tr>
<td>Business process</td>
</tr>
<tr>
<td>System level</td>
</tr>
<tr>
<td>Software level</td>
</tr>
<tr>
<td>Data Architecture</td>
</tr>
<tr>
<td>Application platform</td>
</tr>
<tr>
<td>Component architecture</td>
</tr>
<tr>
<td>Aspect Architecture</td>
</tr>
<tr>
<td>Requirement</td>
</tr>
<tr>
<td>Functionality modeling</td>
</tr>
<tr>
<td>Core build</td>
</tr>
<tr>
<td>Business process</td>
</tr>
<tr>
<td>System level</td>
</tr>
<tr>
<td>Software level</td>
</tr>
</tbody>
</table>
Towards a Systematic Method for Solutions Architecting

project architecture in its lifecycle. However, the objective maturity assessment of multiple architectural models at a domain level continues to be a challenge. Likewise, conducting apple-to-apple comparisons of architectures in different portfolios for simplification or rationalization is difficult, even though the practice recommendations and tools are emerging to address these issues in the application and project portfolio management space.

Generic Application Platform (GAP)

The generic application platform (GAP) forms a common baseline to implement service-oriented architecture, integration, process, and management in IT systems. It primarily focuses on architecture, hosting environment, application frameworks, components, tools, unit testing, build/deploy management, integration, methodology, system management, security, quality of services, and best practices. GAP consists of several units, as illustrated in Figure 12.

- **System environment:** The *logical architecture* module deals with application partitioning through multiple layers/tiers using architectural patterns. The *physical architecture* module covers system topology, performance optimization through intelligent load balancing, scalability via horizontal and vertical scaling, high availability through different redundancy and fail-over mechanisms, as well as clustering, disaster recovery, and network specifications. The *hardware model* module includes server brands and models, with CPU, memory, and disk space specifications. The *server software* module contains software products installed on Web, application, and database servers.

- **Application technology:** The *application framework* module has application frameworks, which are the foundation of application micro-structures. The *software components* module comprises the packages and toolkits used in software development. The *integration solutions* module handles data resource integration. The *shared services* module is made of common services sharable across channels and lines of business.

- **Development lifecycle:** The *development tools* module is composed of tools used in modeling, development, profiling, defects tracking, configuration management, and code quality analysis. The *unit testing* module consists of standardized unit testing at component, application, and system levels. The *build & deployment* module focuses on the build process for deployment. The *process & methodology* module addresses the design approach and system development lifecycle.

- **Nonfunctional aspects:** The *system management* module concentrates on application and server level management. The *security solutions* module deals with security at the data, application, system, network, and data center levels. The *availability & scalability* module copes with the capacity planning, performance, and high availability. The *best practices* module includes the state-of-the-art solutions and industry-proven patterns.

Tao of IT Development and Engineering (TIDE)

The key components in SAM have been discussed in the foregoing sections, which covers the method, process, platform, and patterns. The Tao of IT development & engineering model introduced in this section is an overarching model of best practices utilizing the components discussed. The model consists of 20 functional blocks, each of which has a name of a word starting with the letter ‘p’—partition, path, patterns, people, performance, perspectives, philosophy, picture, plan, platform, policy, portfolio, power tools, practices, principles, problem solving, process, project management, protection, and prototyping.

The goal of this best-practice approach is to enable competitive and effective development of IT solutions with wisdom, vision, strategy, and thought leadership. The key aspects in each module are listed in the functional blocks in Figure 13.
Towards a Systematic Method for Solutions Architecting

With regard to the principles module, though there are many different approaches to information systems development, there is a core set of design principles that are applied to guide how enterprise applications are designed and constructed. The design principles, in the neighborhood of hundreds in number, can be generally grouped into several categories: general, method, object, component, service, data, and infrastructure. In the object-oriented design, a few well-known principles give useful guidance to the interface and class design:

- **Open-closed principle**: Computing entities should be open for expansion, but closed for change.
- **Liskov substitution principle**: A reference to base types in a module must be replaceable by a reference to derived types without affecting the behaviors of the module. In other words, assuming that there is an instance obj2 of class X, for each instance obj1 of class Y as a subtype of X, the behavior of all programs P defined in terms of Y remains unchanged when obj1 is substituted for obj2.
- **Dependency inversion principle**: High level modules must be independent of low level modules. Abstractions must be independent of details. Conversely, details should rely on abstractions.
- **Interface segregation principle**: Clients should be forced to rely on only the interfaces that they use, no more and no less.
- **Design by contract principle**: Every method is defined with pre-conditions and post-conditions, whereas the pre-conditions specify the minimum requirements that must be satisfied in order for the method to execute successfully, and the post-conditions specify the results of a successful method execution. The contract is then formed between the method and the method caller.

As an example, the design by contract principle is widely applied in application integration design. It prescribes that precise and checkable interface specifications must be defined for the interacting components based on abstract data types, analogous to the conceptual metaphor of a business contract. In the object-oriented implementation, the principle imposes a certain obligation that a client module guarantees on entry to invoke the functionality, and it also guarantees a certain property on exit.
Towards a Systematic Method for Solutions Architecting

The use of this idiom has been extended to other fields beyond the object-oriented paradigm, such as Web services, where the Web services description language (WSDL) defines a standard binding interface between the service consumer and provider. Even in the code-first approach, WSDL provides a message-based XML language standard that is programming language independent. In addition to the input, output, and fault definitions, a WSDL schema specifies the endpoint address and transport protocol as well as custom data types.

FUTURE TRENDS

There are a plethora of architecture methods and frameworks developed in the last two decades. Some of them have evolved and others have diminished to be consolidated or migrated to other forms. However, most of the prominent architecture methods and frameworks serve different purposes and focus on different domains with dissimilar scope and structures based on different principles and approaches. As a result, it may be difficult to directly apply these methods and frameworks into a domain portfolio. Consequently, there are generally three main streams on how to make use of the existing methods and frameworks. One approach is to customize a method or framework to be tailored to the needs in a specific segment. In this case, typically a heavyweight method or framework is selected as the baseline, which is further streamlined and customized to adapt to an individual domain. Another way is to combine several methods and

Figure 12. Generic application platform

GAP Structure

System Environment
- Logical Architecture
  - Layer 7
  - Architectural patterns
- Physical Architecture
  - Failover & redundancy
  - Network bandwidth
- Database & DB2
- Data replication
- Brand & model
- Compute & data
- Server farm
- Web server
- App server
- Database server
- Integrators service
- Cloud
- Cluster

Application Technology
- Application Framework
  - Struts
  - Java Server Faces (JSF)
- Software Components
  - JAX-WS, RMI
- Integration Solutions
  - App connectivity
- Shared Services
  - System monitor
- Data access (DAO)
- CICS Web Services

Development Lifecycle
- Development Tools
  - Design/Modelling
  - Development
  - Testing & tuning
  - Deployment management
  - Code quality assurance
- Unit Testing
  - Core testing
  - Application level
  - System test
  - Regression testing
  - Unit testing
  - End-to-end testing
- Build & Deployment
  - Compile build
  - Package build
  - Automated deployment
- Process & Methodology
  - Iterative process
  - Agile methodology
  - DevOps

Nonfunctional Aspects
- System Management
  - Operation process
  - Backup
  - Auditing
  - Reporting
  - Security
- Security Solutions
  - PKI/CA/ACE
  - ID management
  - Access control
  - Auditing & accountability
- Availability & Scalability
  - 24/7 availability
  - Disaster recovery
  - Performance
  - Virtualization & automation
- Best Practices
  - Security
  - Management
  - Infrastructure
  - Data center
  - Coding style
  - Aspect-oriented programming (AOP)
frameworks to create a best-of-breed solution. The constituents in various methods and frameworks are selectively united to formulate a compound process. This usually results in a hybrid approach in the real-world application. The other alternative is for an organization to create its own framework for the particular requirements in the environment. Nevertheless, a number of artifacts in the well-established methods and frameworks can be leveraged directly and indirectly in this approach, such as principles, techniques, and structures.

On the whole, there is no one-size-fits-all in this space. The existing architecture methods and frameworks will continue to grow and mature, incorporating innovative techniques and adapting to the latest advance in technologies. Brand-new methods and more holistic frameworks, like SAM designed in this work, may emerge to meet the new needs in the architecture paradigm. On the other hand, convergence tends to take place to consolidate the methods and frameworks with similar capabilities and structures. Hybrid methods and frameworks possess more promising potential, exploiting the values of both agile and heavyweight approaches. It can be foreseen that notations will be unified and standardized, with the forthcoming round-trip engineering tools to fully automate the architecting process in solutions design.

**CONCLUSION**

Aiming to effectively organize and manage the architecture complexity and diverse business requirements in a large organization, a comprehensive approach is a necessity to abstract concerns,
Towards a Systematic Method for Solutions Architecting

define responsibilities, and present a holistic view of the architectural aspects in a highly structured way. The solutions architecting method (SAM) is a methodical framework for architecting information systems applications. It defines the key artifacts in the design of solutions architecture.

The core constituent entities of SAM are design methodology, systematic processes, solution patterns, and integrated maturity & performance managements. The key modules related to these constituents are defined and articulated in the chapter: prescriptive artineering procedure, technology architecture planning, architecture stack and perspectives, rapid architecting process, architecture readiness maturity, and generic application platform. Additionally, a Tao of IT development & engineering model is designed as a best practice platform to make good use of these modules in the real-world project work.

Collectively, these modules form a holistic discipline guiding the process of developing architected solutions in an enterprise computing environment. Several unconventional concepts and thinking styles are utilized in this overarching framework.

This method has been extensively utilized directly or indirectly in developing various IT solutions in different industries such as finance/banking, telecommunications, and public sector. A few reference solutions have been presented to illustrate the exemplary implementations of the key elements in SAM.

REFERENCES


**KEY TERMS**

**Architecture:** The fundamental structure of a system and organization of its components, their relationships to each other, and to the environment, and the guiding principles and tenets for the design and evolution.

**E2AF:** Extended enterprise architecture framework, covering business, information, system, and infrastructure in a 2-D matrix.

**Framework:** A well-defined reusable structural and behavioral model in which applications can be organized and developed.

**MDA:** Model-driven architecture, an agile approach. MDA aims to separate business logic or application logic from the underlying platform technology.

**RM-ODP:** Reference model for open distributed processing, a coordinating framework for the standardization of open distributed processing in heterogeneous environments, with five viewpoints and eight transparencies.

**RUP:** Rational unified process, a use-case driven, object-oriented and component approach.

**TOGAF:** The open group architectural framework, a detailed framework with a set of supporting tools for developing an enterprise architecture, composed of architecture development method (ADM), enterprise architecture continuum, and TOGAF resource base.

**Zachman Framework:** A logical structure used to categorize and organize the descriptive representations of an enterprise IT environment, designed by John Zachman.