Documenting software architecture, Part 1: What software architecture is, and why it's important to document it

Tilak Mitra
Certified Senior IT Architect
IBM Global Services

15 April 2008

Software architecture has increasingly become important for the development of complex real-time systems. In this new series, learn why and how you should document software architecture. You will learn about the five different views, or aspects, that you should document for any medium- to large-scale software development project. This first article in the series introduces software architecture and the importance of documentation. You'll also get an overview of the architecture views that will be covered in upcoming articles.

View more content in this series

Introduction

Software architecture as a discipline began in the 1970s. With the increasing complexity and pressures of developing complex real-time systems, software architecture emerged as a fundamental construct of mainstream systems engineering and software development.

Like any other enduring discipline, software architecture also had its initial challenges. A software architecture represents the structural and behavioral aspects of a system. The textual and diagrammatic expressions that were used during early efforts to document software architectures were often insufficient and imprecise. What was needed was a consistent and well-understood pseudo- or meta-language to unify all the different ways of representing and documenting software architectures. Engineering and computer science communities, fostered by academic research, have made great strides in developing best practices and guidelines for effective documentation of software architecture.

In this series, you will learn about documenting software architecture. Discover the different aspects you can document: system context, architectural overview, functional architecture, operational architecture, and architectural decisions.

© Copyright IBM Corporation 2008

In this first article, learn what software architecture is and the importance of documenting different aspects of the discipline.

Software architecture

Various researchers have interpreted what software architecture is, and they have different viewpoints on how to best represent the architecture of a software system. None of the interpretations is wrong; each has its own merits. The definition by Bass L, *et al* captures the essence of what software architecture should entail:

> "The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships between them."

The definition focuses on architecture made up of coarse-grained constructs (software components) that can be considered building blocks of the architecture. Each software component, or architectural building block, has certain externally visible properties that it announces to the rest of the architectural building blocks. Internal details of the software component design and implementation are not a concern to the rest of the system, which looks at a specific software component as only a black box. The black box has certain properties it exhibits, which the rest of the software components can use to collectively realize the business or IT goals. Software architecture identifies the architectural building blocks, at the right level of granularity. It also identifies and documents how the building blocks are related to one another.

Architecture as it relates to software engineering is about decomposing or partitioning a single system into a set of parts that can be constructed iteratively, incrementally, and independently. The individual parts have explicit relationships with one another. When woven together, the individual parts form the architecture of the system, enterprise, or application.

There is some confusion about the difference between architecture and design. As Clements P, *et al*. point out, all architectures are designs but not all designs are architectures. Designs that need to be bound in order for the system to meet its functional and nonfunctional requirements and goals are architectural in nature. While architecture considers an architectural building block a black box, design deals with the configuration, customization, and internal workings of an architectural building block. The architecture binds a software component to its external properties. Design is usually much more relaxed than architecture, because it allows more ways to adhere to the external properties of the component. Design also considers various ways to implement the internal details of the component.

Software architecture can be used recursively. Consider a software component (C1) that is part of a software architecture of a system. The software architect hands the component to the system designer, along with its properties, functional and nonfunctional capabilities it should exhibit, and its relationship to other software components. The designer, after analyzing software component C1, decides it should be broken down into finer-grained components (C11, C12, and C13), each of which provides a reusable function that would be used to implement the properties mandated for C1. The designer details out C11, C12, C13 and their interfaces.

At this point, to this designer, C11, C12, and C13 are architectural constructs (or components); each of them has explicitly defined external interfaces. To the designer, C11, C12, and C13 are the architecture of software component C1, and the constructs need further elaboration and design to address their internal implementations. Architecture can be used recursively by dividing a large, complex system into small constituent parts and focusing on each part.

Architecture bounds the system using the architectural building blocks that collectively meet the behavioral and quality goals. The stakeholders must be able to understand the architecture. So it is imperative that an architecture is adequately documented, as discussed in the next section.

Importance of documenting an architecture

Stakeholder: A user of the architecture for downstream design and implementation. Someone who funds the definition, maintenance, and enhancement of the architecture.

The key to communicating to stakeholders the blueprint of the system you are building is to *document* the software architecture. Software architectures are represented through different views—functional, operational, decisional, and so on. No single view can represent the entire architecture. Not all views are required to represent the architecture of a system for a specific enterprise or problem domain. The architect determines the set of views that adequately represent the required dimensions of the software architecture.

By documenting the different views and capturing the development of each part, you can communicate information about the evolving system to the development team and to the business and IT stakeholders. A software architecture has a set of business and engineering goals that it is expected to meet. Documentation of the architecture communicates to the stakeholders how these goals will be achieved.

Documenting the various facets of a software architecture helps the architect bridge the proverbial gap between white-boarding the solution (using the box-and-line approach) and representing the solution in a way that's meaningful to downstream design and implementation teams. The box-and-line diagram of the architecture leaves a lot of room for interpretation. Minutiae that need to be flushed out are often left hidden, and confusingly intrinsic, behind those boxes and lines.

Documentation also fosters the creation of architectural artifacts that are tangible and can be developed in a methodological fashion, such as following a standard template. Software architecture, as a discipline, is mature. You can leverage best practices and guidelines to create standard templates for each view to represent a certain part or dimension of the architecture. Templates may educate the architect on what actually needs to be produced. And they help the architect follow a regimen—beyond the box-and-line technique. Templates define the architecture in more concrete terms so it can be traced directly to the business and IT goals that the solution is expected to meet.

Because of their complexity, typical systems-development initiatives can take around 18 months. Attrition is commonplace on design and development teams, causing a frantic struggle to find the right replacements. New team members usually are deterrents to progress, because they have to

navigate a learning curve before they can become productive contributors. A software architecture that has well-documented artifacts provides:

- A perfect platform for educating new team members about what the solution entails.
- An explanation of how the solution meets business and engineering goals.
- Various views of the solution architecture specific to the problem domain.
- Focus on the view that an individual will work on.

Consider a hypothetical artifact called "architecture decisions" (also discussed in a subsequent section). This artifact identifies a problem to be solved and evaluates alternative mechanisms to address the problem. It provides justification for why a particular alternative is chosen over the others.

A problem is identified concerning the mechanism to access mainframe IBM DB2® tables. Two alternatives are evaluated: use IBM MQSeries®, or use a NEON Shadow Direct adapter (a vendor adapter). Though MQSeries has the capabilities and is less expensive, the latter is a much more stable, industry-strength product at the time the decision is made. Now imagine that the original architect has left the project after a year, and a new architect comes on board. The new architect challenges the team as to why it's not using IBM MQSeries to access the mainframe DB2 tables. The team quickly reverts to the architecture decision artifact, and points out the reason for the choice. Because IBM MQSeries has been tested in the past year to be at par with the other solution, and because it has a smaller price tag, the decision is revisited and changed to reflect the updated solution.

This example illustrates why documenting various facets of a system software architecture is imperative to educate new team members and help them get started with minimum down time.

Different views of architecture

You've learned that an architecture can be represented through multiple views, each focusing on a specific aspect and dimension of the architecture. As Bass L, *et al.* point out, a *view* is a coherent set of architectural elements or constructs along with the relationships among them, as written by and read by system stakeholders.

A *functional* view of the architecture enlists the various architectural building blocks, the relationships among them, and how they are allocated to different layers in the architecture. The *operational* view (also called the technology view) enlists the various infrastructure and middleware software components that provide the run time platform for the functional architecture components to be deployed upon. To an application architect, the functional view assumes primary importance. To the infrastructure architect, the operational view is the one to focus on.

Both views take a different approach to solving the same problem, and both of them are required to move from a conceptual architecture to a physical implementation. The view is used to accentuate a specific dimension of the architecture while deliberately suppressing the others.

Since the 1990s there have been many different sets of views. Perry and Wolf suggest there are some very interesting points about building architecture, including software architecture, with

multiple views. Kruchten, who came out with the 4 + 1 view of software architectures, proposes that there are five views, which when combined, represent a software architecture. The first four views are described below.

View	Description
Logical view	Addresses the static design model
Process view	Addresses the design's dynamic view
Physical view	Addresses how the software components are mapped onto the hardware infrastructure
Development view	Represents the static organization of the software components in the development time environment

The fifth view is more of a litmus test view. It takes a set of architecturally significant use cases (business scenarios) and shows how the set of architectural elements in each of the four views, together with the architectural constraints and decisions around those elements, can be used to realize the use cases.

Another view, published by Soni, *et al*. in *Applied Software Architecture*, consists of four main views that comprise a software architecture:

View	Description
Conceptual architecture view	Describes the system in terms of its major design elements and the relationships among them
Module interconnection architecture view	Describes the functional decomposition and how the software modules are arranged in different architectural layers
Execution architecture view	Describes the dynamic structure of the systems
Code architecture view	Describes how the source code, binaries, and libraries are arranged in a development environment

Many other views are described in software architecture publications, but it is outside the scope of this article to cover all of them. A careful analysis of the different views of software architecture shows there is a lot of similarity between different research results. We have an optimal set of views that are most commonly used to represent a system's software architecture.

The next section provides an overview of the artifacts that are recommended as the minimal set of architecture documents you can generate during the architecture phase of a software development life cycle.

What to document

There are many different views, or aspects, of a software architecture that you could document. For any medium- to large-scale software development project, it is recommended you document, at a minimum, the following set of architectural artifacts:

System context

The system context documents how the entire system, represented as a black box, interacts with external entities (systems and end users). It also defines the information and control flows between the system and the external entities.

It is used to clarify, confirm, and document the environment in which the system operates. The nature of the external systems, their interfaces, and the information and control flows help in downstream specification of technical artifacts in the architecture.

Architecture overview

The architecture overview illustrates the main conceptual elements and relationships in an architecture through simple schematic representations. You can produce architecture overview diagrams that include an enterprise view and an IT system view. The overview helps to represent the business and IT capabilities required by the organization.

An architecture overview also provides high-level schematics that are further elaborated and documented in the functional and operational architectures. And it depicts the strategic direction the enterprise is taking as it pertains to IT systems.

Functional architecture

Also known as the component architecture, or model, the functional architecture artifact is used to document how the architecture is decomposed into IT subsystems that provide a logical grouping of the software components.

It describes the structure of an IT system in terms of its software components with their responsibilities, interfaces, static relationships, and the way they collaborate to deliver the required functions from the component. This artifact is developed iteratively through various stages of elaboration.

Operational architecture

The operational architecture artifact represents a network of computer systems that support some of the performance, scalability, and fault tolerance (among others) requirements of the solution. It also runs the middleware, systems software, and application software components. This artifact is developed iteratively through various stages of elaboration.

Architecture decisions

The architecture decisions artifact provides a single place where all the architecturally relevant decisions are documented. Decisions are typically about but not limited to:

- The structure of systems.
- Identification of middleware components to support integration requirements.
- Allocation of functions to each architectural component (architectural building block).
- Allocation of architectural building blocks to the various layers in the architecture.
- Adherence to standards.
- Choice of technology to implement a particular architectural building block or functional component.

Any decision considered architecturally relevant to satisfy the business and engineering goals is documented. Documentation usually involves:

- Identification of the problem.
- Evaluation, including pros and cons, of various solutions.
- Selected solution, including adequate justification and other relevant details that would help downstream design and implementation.

The rest of this series will discuss how to document these five artifacts in a software architecture.

Conclusion

Software architecture has been around for more than 30 years. The past few decades have seen significant work in software engineering. The software architect plays a pivotal role in shaping a solution that addresses the business, engineering, and IT goals of an enterprise. Documenting a software architecture is extremely important. You can use documentation to communicate with stakeholders about an evolving system. Documentation is also useful for enabling new team members to get on board quickly, because they can use the architecture perspectives as a contextual and bounding premise while implementing a solution.

There has been a lot of confusion about what is architectural and what is nonarchitectural in nature and what aspects of a system should be documented. Architectural templates, which define and standardize the contents in each type of artifact, enable a consistent approach to documenting software architectures.

In this article, you learned about software architecture as a discipline and about the importance of documenting the essential elements of the architecture. You also read an overview of the architectural artifacts that are recommended as a minimal set for documentation. Stay tuned for the rest of this series, which will elaborate upon using a set of guidelines and how to document each of the artifacts.

Resources

- Participate in the discussion forum for this content.
- Read a compendium of published software architecture definitions.
- "Foundations for the Study of Software Architecture" by D. Perry and A. Wolf is a classic article on software architecture.
- Read "Architectural Blueprints - The "4+1" View Model of Software Architecture" by P. Kruchten.
- *Applied Software Architecture* provides practical guidelines and techniques for producing quality software designs.
- "Introducing The Open Group Architecture Framework (TOGAF), Part 1: Understand TOGAF and IT architecture in today's world" (developerWorks, Feb 06) explains how TOGAF open standard framework came to be and how it can make you a better IT architect.
- Read "Insight and outlook, Part 6: What are today's most important IT architecture issues?" (developerWorks, May 06) to learn how to stay ahead of the IT architecture curve. .
- Get an RSS feed for the series *Documenting software architecture*.
- Check out developerWorks blogs and get involved in the developerWorks community.

About the author

Tilak Mitra

Tilak Mitra is a Senior Certified Executive IT Architect in IBM. He specializes in SOAs, helping IBM in its business strategy and direction in SOA. He also works as an SOA subject matter expert, helping clients in their SOA-based business transformation, with a focus on complex and large-scale enterprise architectures. His current focus is on building reusable assets around Composite Business Services (CBS) that has the ability to run on multiple platforms like the SOA stacks for IBM, SAP and so on. He lives in sunny South Florida and, while not at work, is engrossed in the games of cricket and table tennis. Tilak did his Bachelors in Physics from Presidency College, Calcutta, India, and has an Integrated Bachelors and Masters in EE from Indian Institute of Science, Bangalore, India. Find out more about SOA at Tilak's blog. View Tilak Mitra's profile on LinkedIn or e-mail him with your suggestions at tmitra@us.ibm.com

© Copyright IBM Corporation 2008
(www.ibm.com/legal/copytrade.shtml)
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)