Chapter 1: What is Software Architecture?

For more details of the topics covered in this guide, see Contents of the Guide.

Contents

What is Software Architecture?

Software application architecture is the process of defining a structured solution that meets all of the technical and operational requirements, while optimizing common quality attributes such as performance, security, and manageability. It involves a series of decisions based on a wide range of factors, and each of these decisions can have considerable impact on the quality, performance, maintainability, and overall success of the application.

Philippe Kruchten, Grady Booch, Kurt Bittner, and Rich Reitman derived and refined a definition of architecture based on work by Mary Shaw and David Garlan (Shaw and Garlan 1996). Their definition is:

"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns."

In *Patterns of Enterprise Application Architecture*, Martin Fowler outlines some common recurring themes when explaining architecture. He identifies these themes as:

"The highest-level breakdown of a system into its parts; the decisions that are hard to change; there are multiple architectures in a system; what is architecturally significant can change over a system's lifetime; and, in the end, architecture boils down to whatever the important stuff is."

[http://www.pearsonhighered.com/educator/academic/product/0,3110,0321127420,00.html]

In *Software Architecture in Practice (2nd edition)*, Bass, Clements, and Kazman define architecture as follows:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. Architecture is concerned with the public side of interfaces; private details of elements—details having to do solely with internal implementation—are not architectural."

[http://www.pearsonhighered.com/educator/academic/product/0,4096,0321154959,00.html]

Why is Architecture Important?

Like any other complex structure, software must be built on a solid foundation. Failing to consider key scenarios, failing to design for common problems, or failing to appreciate the long term consequences of key decisions can put your application at risk. Modern tools and platforms help to simplify the task of building applications, but they do not replace the need to design your application carefully, based on your specific scenarios and requirements. The risks exposed by poor architecture include software that is unstable, is unable to support existing or future business

requirements, or is difficult to deploy or manage in a production environment.

Systems should be designed with consideration for the user, the system (the IT infrastructure), and the business goals. For each of these areas, you should outline key scenarios and identify important quality attributes (for example, reliability or scalability) and key areas of satisfaction and dissatisfaction. Where possible, develop and consider metrics that measure success in each of these areas.

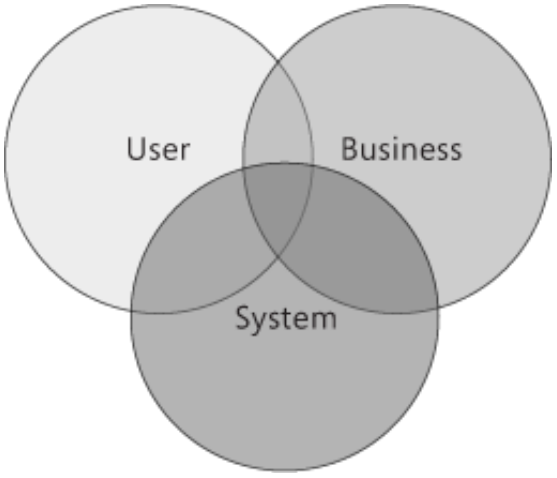

Figure 1

User, business, and system goals

Tradeoffs are likely, and a balance must often be found between competing requirements across these three areas. For example, the overall user experience of the solution is very often a function of the business and the IT infrastructure, and changes in one or the other can significantly affect the resulting user experience. Similarly, changes in the user experience requirements can have significant impact on the business and IT infrastructure requirements. Performance might be a major user and business goal, but the system administrator may not be able to invest in the hardware required to meet that goal 100 percent of the time. A balance point might be to meet the goal only 80 percent of the time.

Architecture focuses on how the major elements and components within an application are used by, or interact with, other major elements and components within the application. The selection of data structures and algorithms or the implementation details of individual components are design concerns. Architecture and design concerns very often overlap. Rather than use hard and fast rules to distinguish between architecture and design, it makes sense to combine these two areas. In some cases, decisions are clearly more architectural in nature. In other cases, the decisions are more about design, and how they help you to realize that architecture.

By following the processes described in this guide, and using the information it contains, you will be able to construct architectural solutions that address all of the relevant concerns, can be deployed on your chosen infrastructure, and provide results that meet the original aims and objectives.

Consider the following high level concerns when thinking about software architecture:

- How will the users be using the application?
- How will the application be deployed into production and managed?
- What are the quality attribute requirements for the application, such as security, performance, concurrency, internationalization, and configuration?
- How can the application be designed to be flexible and maintainable over time?
- What are the architectural trends that might impact your application now or after it has been deployed?

The Goals of Architecture

Application architecture seeks to build a bridge between business requirements and technical requirements by

understanding use cases, and then finding ways to implement those use cases in the software. The goal of architecture is to identify the requirements that affect the structure of the application. Good architecture reduces the business risks associated with building a technical solution. A good design is sufficiently flexible to be able to handle the natural drift that will occur over time in hardware and software technology, as well as in user scenarios and requirements. An architect must consider the overall effect of design decisions, the inherent tradeoffs between quality attributes (such as performance and security), and the tradeoffs required to address user, system, and business requirements.

Keep in mind that the architecture should:

- Expose the structure of the system but hide the implementation details.
- Realize all of the use cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.

The Architectural Landscape

It is important to understand the key forces that are shaping architectural decisions today, and which will change how architectural decisions are made in the future. These key forces are driven by user demand, as well as by business demand for faster results, better support for varying work styles and workflows, and improved adaptability of software design.

Consider the following key trends:

- **User empowerment**. A design that supports user empowerment is flexible, configurable, and focused on the user experience. Design your application with appropriate levels of user personalization and options in mind. Allow the user to define how they interact with your application instead of dictating to them, but do not overload them with unnecessary options and settings that can lead to confusion. Understand the key scenarios and make them as simple as possible; make it easy to find information and use the application.
- **Market maturity**. Take advantage of market maturity by taking advantage of existing platform and technology options. Build on higher level application frameworks where it makes sense, so that you can focus on what is uniquely valuable in your application rather than recreating something that already exists and can be reused. Use patterns that provide rich sources of proven solutions for common problems.
- **Flexible design**. Increasingly, flexible designs take advantage of loose coupling to allow reuse and to improve maintainability. Pluggable designs allow you to provide post-deployment extensibility. You can also take advantage of service orientation techniques such as SOA to provide interoperability with other systems.
- **Future trends**. When building your architecture, understand the future trends that might affect your design after deployment. For example, consider trends in rich UI and media, composition models such as mashups, increasing network bandwidth and availability, increasing use of mobile devices, continued improvement in hardware performance, interest in community and personal publishing models, the rise of cloud-based computing, and remote operation.

The Principles of Architecture Design

Current thinking on architecture assumes that your design will evolve over time and that you cannot know everything you need to know up front in order to fully architect your system. Your design will generally need to evolve during the implementation stages of the application as you learn more, and as you test the design against real world requirements. Create your architecture with this evolution in mind so that it will be able to adapt to requirements that are not fully known at the start of the design process.

Consider the following questions as you create an architectural design

- What are the foundational parts of the architecture that represent the greatest risk if you get them wrong?
- What are the parts of the architecture that are most likely to change, or whose design you can delay until later

with little impact?
- What are your key assumptions, and how will you test them?
- What conditions may require you to refactor the design?

Do not attempt to over engineer the architecture, and do not make assumptions that you cannot verify. Instead, keep your options open for future change. There will be aspects of your design that you must fix early in the process, which may represent significant cost if redesign is required. Identify these areas quickly and invest the time necessary to get them right.

Key Architecture Principles

Consider the following key principles when designing your architecture:

- **Build to change instead of building to last**. Consider how the application may need to change over time to address new requirements and challenges, and build in the flexibility to support this.
- **Model to analyze and reduce risk**. Use design tools, modeling systems such as Unified Modeling Language (UML), and visualizations where appropriate to help you capture requirements and architectural and design decisions, and to analyze their impact. However, do not formalize the model to the extent that it suppresses the capability to iterate and adapt the design easily.
- **Use models and visualizations as a communication and collaboration tool**. Efficient communication of the design, the decisions you make, and ongoing changes to the design, is critical to good architecture. Use models, views, and other visualizations of the architecture to communicate and share your design efficiently with all the stakeholders, and to enable rapid communication of changes to the design.
- **Identify key engineering decisions**. Use the information in this guide to understand the key engineering decisions and the areas where mistakes are most often made. Invest in getting these key decisions right the first time so that the design is more flexible and less likely to be broken by changes.

Consider using an incremental and iterative approach to refining your architecture. Start with a baseline architecture to get the big picture right, and then evolve candidate architectures as you iteratively test and improve your architecture. Do not try to get it all right the first time—design just as much as you can in order to start testing the design against requirements and assumptions. Iteratively add details to the design over multiple passes to make sure that you get the big decisions right first, and then focus on the details. A common pitfall is to dive into the details too quickly and get the big decisions wrong by making incorrect assumptions, or by failing to evaluate your architecture effectively. When testing your architecture, consider the following questions:

- What assumptions have I made in this architecture?
- What explicit or implied requirements is this architecture meeting?
- What are the key risks with this architectural approach?
- What countermeasures are in place to mitigate key risks?
- In what ways is this architecture an improvement over the baseline or the last candidate architecture?

For more information about the key principles of software architecture design, see Chapter 2 "Key Principles of Software Architecture."

For information about the incremental and iterative approach to architecture, baseline and candidate architectures, and representing and communicating the design, see Chapter 4 "A Technique for Architecture and Design."

Additional Resources

Bass, Len, Paul Clements, and Rick Kazman. *Software Architecture in Practice, 2nd ed*. Addison-Wesley Professional, 2003.

Fowler, Martin. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.