Apache Web Server
Applying Lessons from Physical Architecture to Enable Systems to Thrive

WILLIAM T. SCHIANO, DAVID J. YATES, AND LESLIE J. WAGUESPACK
Apache Web Server: Applying Lessons from Physical Architecture to Enable Systems to Thrive

William T. Schiano, Bentley University, USA
David J. Yates, Bentley University, USA
Leslie J. Waguespack, Bentley University, USA

Abstract: The Apache web server is one of the most successful software products in history. The architecture of Apache has been a primary driver of this success. To understand what has made Apache thrive and to learn from its success, we turn to lessons gleaned from physical architecture. Christopher Alexander distilled architectural quality into fifteen properties, the combination of which determines whether the architecture will exhibit wholeness, or life. Waguespack has applied Alexander’s theory to articulate fifteen corresponding properties of modeling and information systems that determine whether an information system will have sufficient functionality and strength to thrive. We first explain Apache’s success over time through the application of these properties. We then explore how organizations can emulate this success in their own information systems by managing the portfolio of properties. Linking information systems properties to desirable business outcomes to create a synergy between design goals and business requirements is critical to success. For a system to serve the needs of a business, its design and implementation should not only accommodate but facilitate change. We argue that facilitating change amplifies, not obviates, the need for great architecture. Applying the lessons from physical architecture to enable systems to thrive thus improves both system agility and business agility.

Keywords: Alignment, Architecture, Design, Functionality, Strength, Business Outcomes, Information Systems, Software Engineering, Thriving Systems

Introduction

The Apache web server is one of the most successful software products in history. The architecture of Apache has been a primary driver of this success. To understand what has made Apache thrive and to learn from its success, we turn to lessons gleaned from millennia of physical architecture.

In the first century BC, Vitruvius described two crucial facets of architecture, functionality and strength, which combine to make architecture beautiful. Early in the 21st century, noted architectural theorist and philosopher Christopher Alexander distilled architectural quality into fifteen properties, the combination of which determines whether the architecture will exhibit such beauty, which he refers to as wholeness, or life. He based his findings on forty years as an award-winning architect and contractor while researching the history, culture, and evolution of architecture.


Waguespack (2010) drew upon the same fundamental underlying nature of order to articulate fifteen corresponding properties of modeling and information systems that determine whether an information system will exhibit such beauty and, as a result, thrive. Figure 1 below depicts both Alexander’s mapping of the nature of order into the underlying properties of great architecture and Waguespack’s mapping of the same fundamental underlying nature of order into properties of modeling and information systems.

Waguespack drew his fifteen information systems properties from an array of computing domains, including requirements engineering, system analysis & design, and software
engineering. We know of no other source that addresses this entire portfolio of properties. It is this breath of coverage that gives the portfolio the power to explain the success of Apache’s architecture and enable the development of better systems and the realization of business value.

![Diagram of Alexander’s Properties](image)

Figure 1: Alexander’s Properties in The Nature of Order Expressed in Modeling and Information Systems

This is not the first time Alexander’s work has had a major impact on information systems and software development. Alexander’s earlier work, *A Pattern Language: Towns, Buildings, Construction* (Alexander, Ishikawa & Silverstein 1977), was so influential that he was invited to give the keynote address at ACM’s OOPSLA, a leading object oriented development conference (Coplien 1996). Lea (1994) outlines the relevance of this work for information systems designers.

Every branch of information systems and computer science research and theory has referred to architecture as an impetus for better systems. The opportunity to learn from thousands of years of accumulated knowledge in physical architecture and emulate the success of physical architects for the construction of great, thriving systems has long been coveted by information systems professionals. Just as building owners work with architects to get the design of a building right, so too can the business work with information systems (IS) architects to get systems right.

The fifteen properties are a powerful vehicle for facilitating the communication necessary between the business and information system architects to build beautiful, thriving systems that are both functional and strong. It has been well established that improvements early in the design not only dramatically improve the quality of the final product but also lower the cost and time to market. Linking the properties to business outcomes strengthens the design by keeping it focused on system needs. By maintaining focus on the portfolio of properties, systems can thrive, improving business outcomes and enhancing both the agility of the systems themselves and the business processes that rely on them.
We first explain Apache’s success through the application of Waguespack’s fifteen properties. We then explore how organizations can emulate this success in their own systems by managing the portfolio of the properties.

**Apache: Great Architecture in Vivo**

Because of its many benefits, the Apache server has been widely used in business contexts as different as the high-profile companies that use it, e.g., Google, Amazon, and IBM. Apache’s versatility, one of its key benefits, is based on the functionality and strength it delivers. This versatility is partially supported by a programmable system with composable functions that can implement a variety of web-based applications. This versatility also goes hand-in-hand with Apache’s manageability, helped by an elegant design with clearly identifiable entities that are constructed and organized using intuitive patterns.

Apache’s versatility, manageability, and other benefits are enabled by a great architecture that has made it the most popular web server for more than sixteen years. For the Apache server, initial acceptance and subsequent adoption and loyalty have been promoted by a system that has a strong track record encompassing the fifteen properties of great architecture described here. Table 1 illustrates examples of design decisions in Apache that exemplify the fifteen information system properties. The text that follows defines each property with a description of the corresponding Apache design decision.

**Functionality in Apache**

**Correctness**

A system is correct if it computes the desired function, i.e., if, for each of the desired range of input data sets, the system produces the desired output (Pollack 1982). The shared experience of the Apache architects and the HTTP/1.1 authors resulted in faithful and concise rendering of HTTP/1.1 in Apache 1.x. Initially, the correctness of the HTTP implementation was maintained by the original eight Apache architects, most of whom ran production web servers. More recently, the open source community validates the system’s external functionality and inspects the source code for consistency and accuracy.

**User Friendliness**

User friendliness is a combination of the following characteristics: ease of learning; high speed of user task performance; low user error rate; subjective user satisfaction; and, user retention over time (Shneiderman 1992). The user friendliness of Apache is driven by the varied wishes and needs of the system administrators of the servers on which it runs. Such diverse requirements demand a multi-purpose and highly configurable server, and require the flexible configuration and management that has evolved as part of Apache. This configuration and management is accessed via a series of hierarchical and layered interfaces, mirroring the modular structure of the server (Engelschall 2000). Although the Apache user interface is complex, this accommodates its users, i.e., server administrators or web masters, who support an increasing number of web-based applications.

**Extensibility**

Extensibility is a system design principle where the implementation takes into consideration future growth. It is a systemic measure of the ability to extend a system and the level of effort required to implement the extension (van Vliet 2008). Apache’s extensibility has played a key role in the addition, adaptation, and evolution of server functionality, while retaining HTTP
compliance. Extensibility is achieved via flexible Apache APIs (Thau 1996). The APIs’ “call out” and “call in” interfaces expose several layers within the server to which users of Apache can add functionality. Apache’s multiple interface mechanisms support a variety of extension approaches and a myriad of programming languages.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Information System Property</th>
<th>Apache Server Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>HTTP implementation</td>
<td></td>
</tr>
<tr>
<td>User friendliness</td>
<td>Flexible configuration &amp; management</td>
<td></td>
</tr>
<tr>
<td>Extensibility</td>
<td>Public application program interfaces (APIs)</td>
<td></td>
</tr>
<tr>
<td>Patterns</td>
<td>Module design patterns</td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td>Process lifecycle and resource management</td>
<td></td>
</tr>
<tr>
<td>Transparency</td>
<td>Apache portable runtime</td>
<td></td>
</tr>
<tr>
<td>Programmability</td>
<td>Hierarchical and layered configuration</td>
<td></td>
</tr>
<tr>
<td>Identity</td>
<td>Root classes and their interrelationship</td>
<td></td>
</tr>
<tr>
<td>Elegance</td>
<td>Configuration change management</td>
<td></td>
</tr>
<tr>
<td>Stepwise refinement</td>
<td>HTTP request handling</td>
<td></td>
</tr>
<tr>
<td>Modularization</td>
<td>Private application program interfaces (APIs)</td>
<td></td>
</tr>
<tr>
<td>Cohesion</td>
<td>Resource pools</td>
<td></td>
</tr>
<tr>
<td>Encapsulation</td>
<td>Platform independence</td>
<td></td>
</tr>
<tr>
<td>Composition of function</td>
<td>Filter chains</td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td>Management interfaces</td>
<td></td>
</tr>
</tbody>
</table>

**Patterns**

Design patterns are general reusable solutions to commonly occurring problems in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations (Gamma et al. 1995). Apache module interfaces follow a handful of consistent design patterns. These patterns map access to existing Apache functions, and prescribe the form for access to future extensions. Interfaces connect modules in familiar and often reconfigurable ways. Modules can function independently or in cooperation, characterized by another set of patterns.
Reliability

Reliability is a design engineering discipline which applies scientific knowledge to assure a product will perform its intended function for the required duration within a given environment (Pham 2000). Two cornerstones of Apache’s reliability are its process lifecycle management and resource management. For example, by configuring the server to use one process per connection, Apache localizes any impact of crashes to a single client. An Apache administrator may also limit the number of requests serviced per process to minimize the impact of cumulative errors, e.g., resource leaks, fragmentation, etc.

Transparency

In software-based systems, transparency requires consideration of heterogeneity of supporting software and hardware, the location and mobility of components, and the mechanisms required to achieve a given level of performance in the event of failures (Kaisler 2005). As open source software, Apache renders the complete architecture transparent; every aspect may be scrutinized (Mockus, Fielding & Herbsleb 2002). From a developer’s perspective, the Apache architecture achieves transparency as the Apache Portable Runtime (APR) reveals a consistent, virtualized host interface despite the numerous idiosyncrasies of the host platforms and operating systems that must be handled.

Programmability

Programmability is the capability within hardware and software to change; to accept a new set of instructions that alter its behavior (Birrell & Ould 1988). The programmability of Apache is realized through its hierarchical and layered configuration, which can be used to control and monitor the server at different granularities and layers. Apache can be configured to run multiple virtual servers on the same system, and these servers can share all, some, or no configuration parameters. Apache’s configuration supports programming at several layers within the server environment, e.g., the entire server, a directory, a location, a file, etc. A master configuration for an Apache server may be read and applied as the server is started. This startup configuration information can be overridden or augmented at the granularity desired for the application. For example, a secure server deployment might have a “startup” configuration that cannot be modified, limiting the programmability of all or part of the server configuration information to the master configuration.

Identity

Identity is that property of an object which distinguishes each object from all others (Khoshafian & Copeland 1986). The four core elements of the Apache server vocabulary are connection, request, server, and process. These elements identify specific concepts and their interrelationships form the core definition of Apache’s behavior. The terms are intrinsic to learning, explaining, or discussing Apache structure and behavior. For example, one connection can service one or more requests and one or more processes run in a context specified by a single server.

Elegance

Elegance combines simplicity, power, and a certain ineffable grace of design. Antoine de Saint-Exupéry, the French aviator and aircraft designer, gave us perhaps the best definition of engineering elegance when he said “A designer knows he has achieved perfection not when there is nothing left to add, but when there is nothing left to take away” (Raymond 1996). A signature
aspect of the functionality achieved in Apache's design is the elegance of its configuration change management. The server administrator changes the configuration by editing the configuration files. Once finalized, a graceful restart of the server increments the process generation number, signaling the new generation to all processes. Every request handling process verifies that it is a contemporary of the current configuration and if not, self-terminates. A new generation of request handling processes is spawned to replace the processes from the previous generation according to specifications of the new configuration. This elegant, coordinated protocol harmonizes the servers’ responsibilities with a minimum impact on efficiency while maximizing the server administrator’s ability to satisfy server user requirements.

**Strength in Apache**

**Stepwise Refinement**

Stepwise refinement is the art and science of producing software by making a series of small steps, each step increasing the level of detail and representing some design decision; it is a technique of design that is carried out in a form of high-level language and which can therefore lead you finally straight into code (Birrell & Ould 1988). Apache’s evolution from serving static web pages to serving complex, dynamically generated pages illustrates stepwise refinement, a coherent progression of expanding capability. Apache 1.3 mapped incoming HTTP requests to handlers monolithically on a first-accepted, first-assigned basis, an implementation sufficient for static web pages (Engelschall 2000). Apache 2.0 elaborated request handling, passing requests through a filter chain, i.e., a collection of service providers. Each service provider determines if it is relevant to a request and chooses to contribute to request processing in a sequence determined for each request at run-time (Gröne et al. 2008). Such stepwise addition of features and advances in design is apparent throughout Apache’s software and documentation.

**Modularization**

Modularization is a particular design structure, in which parameters and tasks are interdependent within units (i.e., modules) and independent across them (Baldwin & Clark 2000). Apache’s use of standard private application program interfaces facilitates the effective compartmentalization of its functionality within the system for Apache developers. Every feature and function in the server, outside of a small core, is implemented as a module with a standard interface (Kew 2007). These modules provide the first line of extensibility for specializing or extending features of the server. Modularization in this form has been leveraged in at least two important dimensions: security and flexibility (e.g., providing a framework to accommodate different server environments such as those based on Java and .Net).

**Cohesion**

Cohesion measures the strength of functional relatedness of elements within a module (Zuse 1997). Apache’s resource management library demonstrates the cohesion achieved structurally and functionally by associating resource pools with different components. Each component class, i.e., requests, connections, server processes, and configurations, supports instances with distinct lifetimes, i.e., an HTTP request, a TCP connection, a server process lifetime, and a configuration lifetime. This integrates the management of request, request service, response, and associated resources, e.g., file descriptors and memory. This integration is leveraged for several purposes, including enforcing a limit on resource retention. Resource pools reflect a fundamental and more cohesive factoring of resource management in Apache’s design.
Encapsulation

Encapsulation enables the software engineer to group data and the routines that operate on them together in one place, and to hide irrelevant details from the users of an abstraction (Scott 2006). Platform independence is a critical feature of Apache 2.0, specifically achieved by the Apache Portable Runtime (APR) project. Through encapsulation, the APR insulates the remaining server architecture, rendering it largely immune to host dependencies. For instance, the network service framework virtualizes the host operating system, enabling Apache to thrive on diverse host operating systems.

Composition of Function

Function composition (not to be confused with object composition) is an act or mechanism to combine simple functions to build more complicated ones (Meyer 1988). Apache uses composition of function to cascade server requests through an input filter chain and server responses through an output filter chain. These filter chains selectively apply features to requests for which the server is configured (Gröne et al. 2008). Composition of function is evident in the structure of input and output filter chains and in their dynamic execution.

Scale

Scale results from the imposition of a telescoping sense of focus that may be directed to an observer’s purpose and renders in clarity the system features relevant to that purpose. By achieving scale, a system designer provides differing granularities of comprehensibility to suit the requirements of a variety of observers (Waguespack 2010). Apache’s management interfaces are designed to scale to the user’s needs. The interfaces provide system control by mirroring the server’s telescoping modular structure. Users may tune the behavior of individual server functions at multiple levels depending on the scope and degree of control desired.

Improving Business Outcomes with Thriving Systems

Inadequately architected systems haunt businesses and information systems organizations, posing cost and continuity risks. Systems often don’t work as intended. Systems sometimes collapse. Many systems are unattractive. Often enhancements to software are prohibitively expensive. In other words, systems often suffer from poor architecture. Perhaps this is because we often hire the equivalent of interior decorators to design our systems and they focus on only a subset of the properties of great architecture. The interior decorator knows a great deal about what may appeal to clients. Too often the outcomes are driven more by the decorator’s preferences than those of the client. In information systems, we often design software through mock ups of interfaces or other slightly more robust forms of prototypes. While this may lead to an interface that is appealing to the client, it does little to ensure that the system will be correct, reliable, and built to last; that is, it does little to ensure the system will be beautiful, and thrive.

Poor architecture is often the result of an ill-conceived design. Historically, much attention has been paid to requirements engineering, but less to the design stages following. While many organizations share functional specifications with business clients, fewer have formal mechanisms for interacting around design.

Apache’s architecture and design was summarized in the previous section entitled Apache: Great Architecture in Vivo. The framework for this summary was based on Waguespack’s fifteen properties (Waguespack 2010), drawn from the nature of order (Alexander 2002a). The remainder of this section connects modeling and information systems properties to business outcomes via these properties.
Tying IS Properties to Business Outcomes Strengthens System Design

Information systems thrive when the goals of the enterprise fuse with the capabilities of their information system(s). Businesses thrive and improve their return on investment by carefully considering maintainability and usability in the information system design. For a system to serve the current and future needs of a business, its design and implementation should not only accommodate but facilitate change both in the system and the business. In physical architecture, blueprints formalize the shared understanding of stakeholders, facilitating the construction of more durable and usable systems, and make subsequent changes easier. The documents that serve as blueprints for the Apache server (Engelschall 2000; Gröne et al. 2008; Kew 2007) reveal a system that is very maintainable, enabled by a modular implementation of cohesive subsystems that are well encapsulated.

Fortunately, there are standards-based tools that help businesses in casting an initial system design and changing or refining a design in order to improve business processes. For example, Unified Modeling Language (UML) is a standard way of representing information system architecture. UML provides the ability to create blueprints readable by all professionals involved in the design and construction of systems. They also offer an artifact to facilitate subsequent maintenance.

Many of the properties of great architecture are promoted through the use of standards. The common benefit of doing so is reducing cost. For the Apache server and business applications that use it, cost efficiencies result from correct compliance with standards that promote interoperability such as HTTP, PHP, JSP, Java EE, etc. An additional benefit of leveraging standards is avoiding vendor lock-in. Apache users enjoy this benefit because of its standards compliance and because of a design that encapsulates platform dependencies.

Employing the Fifteen Properties Aligns Design with Business Needs

Alignment of design and requirements can occur seamlessly when the properties frame an ongoing dialog about design and quality. Managing information system architecture via the properties works especially well for organizations that build their own software or work closely on design decisions with their outside vendors who build it for them. The properties are an excellent starting point for establishing quality targets for a design or a completed system. Establishing the right balance of properties is idiosyncratic to a company’s strategy, context, processes, and systems.

The properties are also important throughout the system lifecycle. Even though the properties help to strengthen architecture and evolve a design, this is not without challenges. The properties are interconnected, and emphasizing any given property may counteract the influence of another. Thus, the properties need to be managed as a portfolio.

Thriving Systems Improve Development Agility

The return on architecture investment increases dramatically the more robust the design, and the longer the product is kept in service. Conservative estimates place the cost of maintenance of software at over 50% of total cost. Good architecture can dramatically lower this cost while also improving the user experience.

While the properties add value at all stages of the traditional waterfall lifecycle, they are equally valuable in increasingly common agile methods. In agile methods, contact with the customer becomes more important, and the properties facilitate this interaction. Time pressure is increased and the consequences of choices made early on are greater. Understanding the properties and tradeoffs enables the better and faster decisions that are necessary for agile development.
The recent push for agile methods is often seen antithetical to architecture. When we think of architecture, we think of a complete plan drawn up in advance of construction. Buildings, however, are not expected to turn out different from what was intended at the outset. The point of agile methods is to facilitate change in response to evolving or poorly understood user requirements. This does not obviate, but amplifies, the need for architecture. Building systems more quickly requires a strong foundation upon which to build, and it is crucial to avoid duplicating existing functionality. Such duplication takes extra time, costs more to maintain, and misses opportunities to enhance modules that have economies of scale.

The use of agile methods has created a series of software product lines with the Apache web server at their core. The Axis toolkit can be added to a web server run-time configuration to support web services. More than 500 modules in the open source ecosystem that surrounds the core Apache server extend Apache and provide a variety of Software-as-a-Service (SaaS) implementations. These product lines have evolved as a result of implementing useful applications efficiently via an extensible and programmable infrastructure. Although these applications are built using a common core, the variety of deployment scenarios required by a multitude of business needs is quite remarkable. The flexibility and scalability of Apache, in terms of its deployment, use, and operation, is supported by the encapsulation of platform dependencies that are transparent within different run-time environments.

**Thriving Systems Enhance Business Agility**

Much has been made of the need for businesses to be more responsive to the marketplace due to increased competition and an escalating rate of change. Architecture can improve alignment between business and information systems in several ways. Architecture creates a shared understanding of what the system will do and how, in a format that all stakeholders can understand. The architecture can be referred to throughout the development process and throughout the life of the product. The process of creating an architecture should encourage intense communication between business stakeholders and information technologists, which facilitates alignment.

The Apache server enables agility in business applications because it is extensible and continues to grow and evolve. This agility can be achieved efficiently because of Apache’s correctness, reliability, and user friendliness. These properties collectively help in managing the cost of system configuration and change management while improving system operation.

**Summary and Implications**

Alexander’s properties of physical architecture derive from a fundamental underlying nature of order in the universe. Waguespack (2010) draws on the same nature of order to articulate fifteen corresponding properties of modeling and information systems that determine whether an information system exhibits great architecture. These properties dissect the cumulative effect of design decisions providing a taxonomy with which to inspect the varied elements of design that evoke the observer’s sense of quality. The Apache web server has grown, evolved, and thrived over time by strengthening the properties described in this study. Like the Apache server, business information systems that achieve functionality and strength through these properties will thrive.

Organizations need to aspire to beauty in their information systems, based on functionality and strength, rather than just meeting business requirements. By managing the fifteen properties as a portfolio, information systems can provide greater value to the business by meeting current and future needs. As businesses become increasingly dependent on information systems, developing thriving systems will become a necessary part of building a thriving business. For a system to serve the needs of a business, its design and implementation should accommodate and facilitate change, both in the system and the business. We believe that facilitating change
amplifies, not obviates, the need for architecture. Building more flexible systems requires a strong foundation upon which to build and functionality that is aligned (and continuously realigned) with changing business requirements.

Conclusion and Future Directions

Great architecture can maintain and improve alignment between business and information systems in several ways. The process of creating an architecture encourages communication between business stakeholders and information technologists, which facilitates alignment. The fifteen properties are an excellent starting point for establishing the desired qualities of a design or a completed system. The properties create a shared understanding of what a system does or will do and how, in a format that all stakeholders can understand, that can be referred to throughout the life of the product.

Although this research focuses on Waguespack’s fifteen properties of modelling and information systems in the Apache web server; they promise broader application. Waguespack (2010) demonstrates exploiting the properties in system models using the object-oriented paradigm and the relational paradigm. Waguespack and Schiano (2012), an analysis of SCRUM, examines the properties as expressed in the agile project management paradigm. The properties also form the foundation of Thriving Systems Theory, an information systems design theory, described in Waguespack and Schiano (2013). Future research explores managing the properties’ impact on both waterfall and agile methodology effectiveness. When using such methods, project management choices include partitioning resources, assigning responsibilities, task sequencing and precedence, communication patterns, milestones, and quality control measures. Traditionally these choices are made at different times, reflect different priorities and organize various activities to yield a project plan. The architecture of a project guides the sequence and substance of such choices. Through the fifteen properties, the degree to which these choices contribute to the project outcome determines to what degree the project has what Alexander describes as wholeness, or life.

Finally, we hope that other scholars and practitioners will study Christopher Alexander’s more recent work (Alexander 2002a, 2002b, 2004, 2005) with the goal of understanding the nature of order and applying this rich theory in their own field or discipline.
REFERENCES


**ABOUT THE AUTHORS**

**Dr. William T. Schiano:** Bill Schiano is Professor of Computer Information Systems at Bentley University. In 2000, Bill served as President of thoughtbubble productions, a New York based new media company founded in 1995, helping secure an investment from The Formula Group in November 2000. Before joining thoughtbubble, Bill was a Research Affiliate at CSC Index Research and Advisory Services. Bill has conducted extensive research and published several articles and case studies on the role of technology in retail and service industries. His current research focuses on the role and management of information systems within organizations faced with outsourcing and financial challenges. Bill’s consulting centers around the strategic use of information technology and the management of information systems departments and personnel. Dr. Schiano has an AB in Economics from Williams College and Doctorate in Information Systems from Harvard Business School.

**Dr. David J. Yates:** David Yates is Associate Professor of Computer Information Systems at Bentley University. David’s research areas include computer networking, data communications, sensor networks, embedded systems, and operating systems. Before joining Bentley, David held research and academic positions at the University of Massachusetts and Boston University. His work has been published and presented at international symposiums and conferences, and appeared in refereed journals. In the corporate arena, he was a co-founder and vice president of software development at InfoLibria—a startup that grew to become a leading provider of hardware and software for building content distribution networks before it was acquired. With various colleagues, he holds several patents for processes and systems related to computer networking, content management, and mobile computing. Dr. Yates holds a PhD and MSc from the University of Massachusetts and a BSc from Tufts University.

**Dr. Leslie J. Waguespack:** Dr. Waguespack is Professor of Computer Information Systems at Bentley University, where he has been a member of the faculty since 1984. Les joined Bentley after four years on the computer science faculty at Louisiana State University. He holds BSc, MSc and PhD degrees in computer science from the University of Louisiana at Lafayette with research in information systems and computer architecture. As a United States Air Force officer, he served five years as a systems architect at the Air Force Manpower and Personnel Center designing and implementing decision support and worldwide information retrieval applications. His work has been presented at international symposiums and conferences, and published in refereed journals. *Thriving Systems Theory and Metaphor-Driven Modeling* appears as a volume published by Springer-Verlag. He has also consulted for IT, medical, financial, and not-for-profit corporations. He teaches systems analysis, data management, software project management and object-oriented systems engineering. His research interests include object oriented modeling and systems engineering and the nature of design quality in physical architecture and systems.
The International Journal of Design Management and Professional Practice is one of six thematically focused journals in the collection of journals that support the Design Principles and Practices knowledge community—its journals, book series, conference and online community.

The journal explores the organization of design, design work and design as a professional practice.

As well as papers of a traditional scholarly type, this journal invites presentations of practice—including case studies documenting professional practice, accompanied by exegeses analyzing organizational purposes, processes and effects.

The International Journal of Design Management and Professional Practice is a peer-reviewed scholarly journal.