CHAPTER 1

The Need for Incremental Software Architecture

Technical books rarely begin with conclusions. But the impetus for this book is so strong that it must be revealed at the onset. So, with no time to spare, here is the bottom line: IT and business organizations, in their current incarnations, must be *eliminated*. Replacing them with regional,[1] nimble, and smaller management and technical groups, called *Micro-Organizations* in this book, will be of immense benefit to the product and software development community.

Mere replacement is not enough, however. Merging a Micro-IT organization with its Micro-Business organization counterpart could diminish the constant battle for alignment efforts and improve firm-wide communication. Moreover, unifying smaller business and IT groups to provide rapid enterprise solutions could reduce the long-running frictions between the two, and create a more productive work environment.

This vision accentuates the need to break down the traditional *enterprise centralized management* into smaller *decentralized organizations* to boost efficiency and speed up decision-making. Consequently, regional, small-scale, and agile Micro-Organizations would seize governance and best practices responsibilities to deliver practical and superior systems.

As a result, joint business and IT teams would operate autonomously to deliver and integrate products on time and on budget. Rather than reporting to enterprise executives, teams would report to *regional management*, which understands the distinct culture and requirements of local business operations.[2]

Such a shift in organizational thinking would eliminate the difficulties of trying to conserve a centralized management structure that is slow to respond to critical business events. A lightweight Micro-Organization would then become proactive, reducing the staggering cost of enterprise policing and governance. *Enterprise-wide technology standardization*, therefore, would be the practice of the past. And *enterprise-wide architecture* best practices and standards would cease to exist.

This does not imply that enterprise-wide architecture groups would vanish, too. The charter of such a design organization would shift to a more tangible one. For that reason, architects should focus on providing certified architecture blueprints guaranteed to work in a production environment.

As you progress through the book, keep the Micro-Organizations idea in mind. And if time allows, imagine a workplace that accepts nothing less than devoting *all* its precious energy to producing high-quality and practical products.

For now, let us focus on chief thrust of this book*: presenting a new approach to enterprise software design, development, and integration—Incremental Software Architecture*.

The new method unveiled in the chapters that follow is suited for all enterprises, regardless of their structure and organization. Pursuing the incremental software architecture approach also may drive organizations to break down their convoluted structures into agile Micro-Organizations, accelerating time to market.

In the meantime, though, there is a compelling reason to understand what incremental software architecture is, and how it can be employed to ward off the deployment of failing systems to production environments. This new approach could also be pursued to save underperforming systems and improve enterprise integration of applications, middleware, and network infrastructure.

Now, we've got our work cut out for us. Let's roll up our sleeves and move on.

End-State Enterprise Architecture: A Risky Proposition

The design phase of enterprise applications and infrastructure integration calls for the delivery of an end-state architecture. Architects, typically senior designers, submit appropriate artifacts to communicate the future state of a production environment to the business, software development, and operations groups. Specifically, the delivery includes diagrams illustrating an ecosystem in which applications and middleware run on networks, exchanging messages to execute business transactions.

Again, *end-state architecture is all about future technological implementation, integration, and distribution of business services to consumers, empowered by enabling infrastructure, to ensure enterprise operation continuity and stability*.

Software architects who deliver an end-state architecture diagram typically claim that the design is solid and unbreakable. In many cases, however, such an artifact merely illustrates intangible and *unproven* deployment that later may fail to operate in production, despite the vast knowledge and experience of its creators.

Why is this architecture unproven? A theoretical enterprise end-state architecture diagram guarantees nothing. Would a depiction of a production environment meet business and technical requirements? Would the illustrated applications operate flawlessly in production? Would service level agreements (SLAs) be fulfilled?

No one really knows.

The consequences of such a theoretical and risky design could be devastating to the business organization that is unable to launch software products on time in harsh market conditions. The skyrocketing cost of the software development efforts that follow an unproven enterprise end-state architecture could be calamitous, and the loss of revenue is typically vast.

Do Not Invest in Unproven Enterprise End-State Architecture

What then would be the consequences of launching a large-scale, or even midsize, software development and integration project enterprise-wide without knowing if in fact the end-state architecture will work in production?

Traditionally, once teams are engaged in actual software development and delivery initiatives, budgets would have been already approved. Allocated funds deplete exponentially as time goes by. Development teams devour resources at the speed

of light, and cost projections are proven false. Consequently, the actual software construction, deployment, and integration phases often commit organizations to overwhelming expenditure, with little chance to reverse the course of projects—resulting in irreparable loss of resources and time.

Business and technological management should not accept an *unproven* end-state architecture, of which no one can predict the pitfalls of ill-designed systems and their corresponding operating environments. Budgets should not be approved and allocated to implement theoretical or academic architecture blueprints.

Simply put, *do not support speculative architecture*.

To prevent such mistakes, a new enterprise design process is therefore required. One that is *proven* and *reliable*. One devised to strengthen the trust between software design practitioners and business organizations. The term "proven" means that the end-state architecture should *not* be a theoretical proposition. It must be a software design and development model based on *tangible and realistic* facts, *pretested*, *verified*, and *certified*. This approach should guide software developers and integrators to deliver smaller chunks of solid code for one purpose only—*verification of enterprise architecture assumptions*.

Consequently, the software construction phase, as we know it now, would transmute into a concrete form of *design proof*, circumventing financial calamity that is hard to recoup.

How can such a method be accomplished?

Focus on Incremental Software Architecture

A new enterprise approach for software product construction, deployment, and integration should be considered. Chartered to deliver proven and solid architecture, design practitioners should lead source code construction and delivery initiatives. They should be accountable for the quality of their design throughout the overall product creation and distribution life cycle.

Developers, on the other hand, should take the back seat, respond to the design pace, and follow successions of software architecture progression. Indeed, they should avoid organically grown environments that are not aligned with an emerging design strategy. Developers should also seek direction from design teams, rather than employing shaky technologies that may fail to perform in production.

Incremental architecture, then, should mark a shift in the phases of enterprise software design, development, and integration.

So What Is Incremental Architecture?

Imagine an end-state architecture diagram that illustrates a production environment, in which a number of systems depend on each other, integrated to enable business transactions. In this diagram, as depicted in Figure 1.1, you may also find a number of architecture components, such as the data access layer, business services layer, repositories, gateways, adapters, and software proxies. In addition, you may note an enterprise service bus (ESB)—a mediating middleware product—that enables message exchange between consumers and services.

Complex? Indeed.

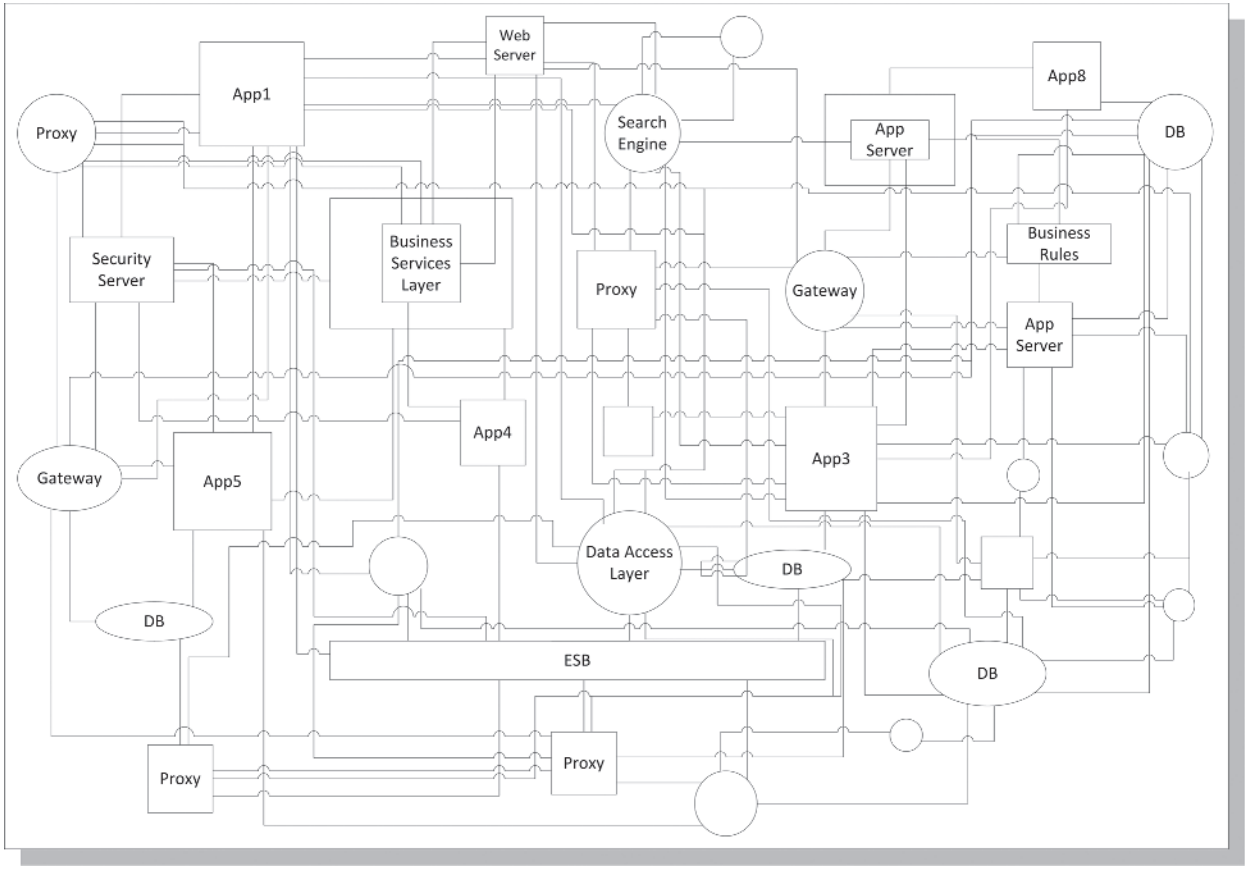

Figure 1.1 Typical End-State Architecture Diagram

Does such an end-state architecture diagram represent a feasible and a proven performing environment? Can anyone assure that such an illustration depicts an error-free software implementation deployed to production? Would the source code meet business requirements? Would performance bottlenecks be nonexistent?

Now, could the end-state architecture blueprint be proven and certified, avoiding a financial burden caused by systems design flaws? This question reveals the motivation for *verifying* the feasibility of a software design and integration throughout all stages of a product's development and deployment life cycle.

Then, how can software architects confirm that an end-state architecture is indeed practical, immaculate, and capable of flawlessly executing business transactions in a production environment that is already strained?

The Art of Architecture Discovery, Analysis, and Decomposition

To accomplish this mission, a meticulous architecture discovery and analysis should begin to study the proposed architecture. Systems, applications, and their supporting middleware and network infrastructure should be ascertained to understand the environment (refer to Chapters 4, 5, and 6 to read about the discovery and analysis of systems, applications, and their supporting production environment).

Next, the end-state software architecture should be sliced into smaller sections, just like cutting a cake into pieces. Subdividing the end-state architecture into smaller segments will enable software architects to drill down into the details of their design. This subdivision process is named *architecture decomposition*, during which an end-state architecture blueprint is *compartmentalized* into distinct areas of interest.

A physical section of an end-state architecture, for example, may include two applications and three databases. Another section may consist of a middleware product, such as an ESB. An additional section may contain a Web server and an application server and so on.

Architecture decomposition does not necessarily have to be based on physical partitions. End-state architecture sections may contain other areas of interest, perhaps contextual: business functions, lines of business, or even business ownership and/or sponsorship.

Decomposition could also be applied to a troubled architecture that has already been deployed to production—an existing implementation that harms the business and creates a great deal of organizational anxiety. In this case, a production environment could be sectioned into smaller segments to simplify the verification process. This effort characteristically helps to isolate underperforming segments of the architecture, narrow down existing issues, and discover root cause problems.

To learn more about architecture decomposition, refer to Chapters 7–11.

Architecture Verification

Next, in lieu of launching a full-fledged and costly enterprise application construction and integration effort, a section-by-section *architecture verification* process should take place. The process of verifying individual sections of an end-state architecture should be easier than attempting to implement and deliver an entire enterprise architecture blueprint. It is apparent now that this mission calls for confirming that each section would work properly in production.

So how can such an architecture verification process be accomplished and what does it entail? Led by design teams, small development groups should be commissioned to construct and deploy only sections of the end-state architecture—not the entire architecture blueprint. Again, the implementation—development, deployment, and integration—must tackle sections of the decomposed architecture. Thus, software construction should be a part of the end-state architecture verification process—pursuing gradual implementation, adjusting the development and integration progress to the *evolution of the design*, rather than leading the product life cycle.

As for that, *the traditional software construction phase in its current manifestation ceases to exist*. Now, *software construction means architecture verification*.

Enterprise architects should then be responsible for proving that vital sections of an end-state architecture meet business and technical requirements. For already deployed, unstable production environments that require repair, the verification process takes place when sections of the architecture are being tested for performance and capacity.

By now, it is obvious that the architecture verification means a gradual approach to proving that each part of the architecture would indeed work, or is working properly in production (if the verification process was performed on an existing failing architecture). Systematic verification will undoubtedly increase the confidence in an enterprise design. This method shifts the focus from development to design, driven by software architecture—not software development.

Not everything is rosy, though. Proving that each section in an architecture works as designed does not mean that the entire end-state architecture and its dependencies will function as they should, once integrated in production. The rule of thumb suggests, therefore, that an entire deployed environment must operate properly as a whole.

An additional verification stage is necessary then to ensure that the proposed end-state architecture is sound and the integration is solid. Enterprise architects, developers, and production engineers perform *architecture stress testing*. This supplemental effort would confirm that the architecture is indeed functioning appropriately under high-volume business transactions.

Finally, as a part of the verification endeavor, an enterprise capacity planning process is launched. Perusing this would ensure proper allocation of computing resources for current and future end-state architecture environment operations.

Chapters 12–15 elaborate on the methods of the ecosystem verification process.

Can Modeling and Simulation Substitute for Incremental Software Architecture?

Traditional approaches to describe software and its environment have been employed for years. One method is software modeling, used to depict a system or an application from different perspectives. Modeling typically illustrates the behavior of software, for example. Another view may identify the various components of an application. Other perspectives focus on physical, logical, and process aspects of a system.[3]

Software modeling is all about expressing a design by using an abstract language, a descriptive syntax and notation, visually depicting a future software implementation

and its corresponding environment. A mere depiction of this architecture would divulge nothing about its ability to meet business requirements. Nor would such a diagram tell us anything about system response time and performance. The software modeling method, nevertheless, is a far cry from the incremental software architecture—an approach described in this book devised to verify if an enterprise architecture will indeed work in production.

Software simulation, however, may shed light upon the capabilities of a system and its environment to meet performance and stability requirements. The simulation of a production environment typically takes place in a virtual space, in which a production landscape is replicated for modeling the behavior of applications, middleware, and network infrastructure. By observing a simulated environment, one may identify a system's failures and ill-designed architecture components that miss the required performance mark. Since software simulation is not pursued in production, the modeling results are difficult to confirm. Nor can simulated models accurately forecast the behavior of a system and its related applications, middleware, and network infrastructure.

With software modeling and/or simulation, no one can accurately predict the solidity and readability of an end-state architecture. No one can ensure a system's stability. No one can pinpoint troubled sections of architecture. And no one can guarantee that a design meets non-functional requirements.

Platform for Change

There is nothing more frustrating to employees than lack of an enterprise platform for change. A platform for change is a powerful stage for those whose mission calls for organizational changes. A platform for change could be an open forum, perhaps gatherings or informal meetings, during which new ideas are voiced to promote technological endeavors or business objectives. A platform for change could also be a laboratory dedicated to technical experiments, during which open-source libraries and components are downloaded from the Internet for evaluation and proof of concepts.

Undoubtedly, there are myriad platforms that enable employees to foster a fresh enterprise direction or vision. The alterations to the way we do business could take many forms. Enterprise cultural changes, for instance, are arduous and slow to fulfill. Cultural aspects pertain to an alteration of a company's core values, communication practices,[4] and even staff attitudes. In contrast, adoption of technological implementations tends to be fast and vigorous. Technological developments occur constantly, influencing the way we run our production environments.

Microservices: A Product of Change

Changes imposed by upper management are named *top-down* initiatives. Executives typically perform reorganizations and issue best practices and policies to drive the direction of the business. These types of changes are slow, and as time passes, they may not be relevant any longer.

Similarly, enterprise architecture standards are not always issued in a timely manner. Governance departments whose charter is to draft best practices are not always

synchronized with the various projects that typically take place simultaneously in the organization. On the other side of the aisle, software developers and integrators, commissioned to deliver source code on time and on budget, cannot afford to wait until enterprise decisions and standards are published.

Not many choices are left. In these cases, the change of architecture direction is propelled from the bottom. Specifically, with the absence of an established organizational platform for change, small development teams tend to ignore enterprise architecture best practices. Often named the *bottom-up evolution*, this movement uses open-source products and mixed and unstandardized technologies to build applications or smaller-scale services.

The outcome of such drive is indeed powerful—at the same time, though, unconventional. The upshot is refreshing since the chief attention is given to product development—not necessarily projects. The formation of such decentralized and self-governed teams allows the management of decentralized databases and the development of organically grown applications. The design method employed here is named *microservices*.[5] The products they deliver are independent, loosely coupled, and focus on smaller-scale problems. The overall emphasis is not on enterprise asset reuse. Here, reusability is applied to components that drive the construction of services—not on enterprise expenditure reduction or asset consolidation.

But even with the focus on small-scale and agile implementations, disregarding organizational standards and enterprise architecture direction, the contribution of the microservices architecture is vast. Turning away from the traditional *monolithic* system architecture is a leap forward in the right direction. This includes breaking off from tightly coupled implementation practices, rejecting a centralized governance approach for software development, and avoiding huge investments in large projects.

Incremental Software Architecture and Microservices Architecture

The incremental software architecture approach is a continuous section-based design, discovery and analysis, decomposition, and verification process. Akin to the microservices architecture, the risk of engaging in perilous and large-scale implementations or producing monolithic application formations is utterly reduced.

As explained in the previous sections, the driving motivation of the incremental software architecture is to conform to an enterprise software design—a high-level view that software developers not always are able to observe. Slicing an architecture blueprint into smaller segments and implementing them minimizes risks to the business as well.

As per the incremental software architecture approach, the gradual verification and certification of an enterprise end-state architecture enforces regional best practices and policies upon smaller development teams. Avoiding employment of unstandardized technologies and, at the same time, decentralizing the software development efforts are other benefits that are hard to ignore.

It is possible to envision, though, that the microservices architecture would be the design verification arm of the incremental software architecture approach. In other words, small development teams would focus only on constructing segments of the end-state architecture, an incremental approach leading to the overall certification of

the overall enterprise design. This would be a combined effort to deliver high-quality software to production environments.

Incremental Software Architecture Process

This book elaborates on the incremental software architecture process and its chief tasks to accomplish in Parts 2, 3, and 4. Part 1 is a guide provided to characterize levels of system failures and assist business and IT professionals to identify and classify the causes of underperforming implementations.

There is nothing intricate about this method of design, implementation, and integration of organizational enterprise assets. There is nothing to fret about, because the approach calls for only three stages, through which an enterprise end-state architecture is discovered and analyzed, decomposed, and certified, as depicted in Figure 1.2:

1. *End-state architecture discovery and analysis*. This stage represents the methods employed to ascertain systems and their related applications in an end-state architecture proposition or in a production environment (Part 2).
2. *End-state architecture decomposition*. Structural, behavioral, and volatile attributes of end-state architecture drive the decomposition process, rendering two distinct perspectives: business and technology (Part 3).
3. *End-state architecture verification*. Proven end-state architecture is one that is certified by three authentication tasks: design substantiation, end-state architecture stress testing, and enterprise capacity planning (Part 4).

Finally, What Is a System?

It would be odd for a book about architecture not to have a definition for the term "system." Unfortunately, as nice as it would be, there is no common industry definition for such an entity. This term means many things to a myriad of organizations. It is so subjective that the various interpretations introduce only confusion to the business and IT communities.

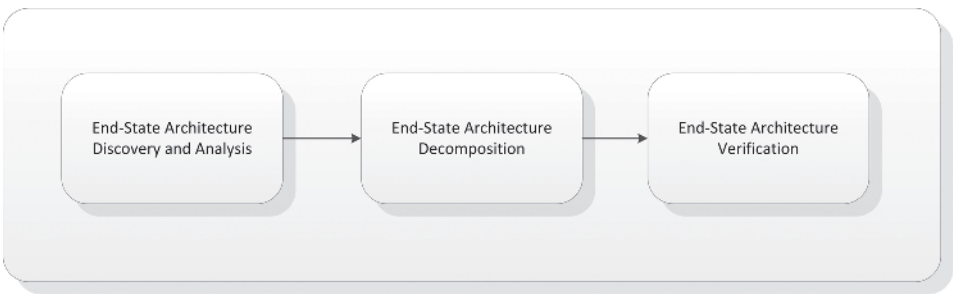

Figure 1.2 Incremental Software Architecture Process

So what is a system? In this book, a system is analogous to an operating technological *environment*. Moreover, a system is the largest entity in production. It encompasses enterprise assets such as applications, components, middleware products, and network infrastructure. When we say "system," we mean an autonomous run-time environment that offers business and technological solutions.

Furthermore, a production environment is typically made up of multiple systems. An end-state architecture, on the other hand, may contain one or multiple systems.

Notes

1. Regional management and technical groups are geographically disbursed business domains, known as lines of business, that specialize in providing services to communities of consumers: Patrick Heinecke, *Success Factors of Regional Strategies for Multinational Corporations: Appropriate Degrees of Management Autonomy and Product Adaptation*, 2011, Springer Science & Business Media, p. 5.
2. Eric Flamholtz, Yvonne Randle, *Corporate Culture: The Ultimate Strategic Asset*, 2011, Stanford University Press, p. 8.
3. The various perspectives of system modeling are elaborated on in Philippe Kruchten's 4+1 well-known research paper, published in 1995.
4. Edgar H. Schein, *Organizational Culture and Leadership*, 2010, John Wiley & Sons, p. 7.
5. Lucas Krause, *Microservices: Patterns and Applications: Designing Fine-Grained Services by Applying Patterns*, 2015, Lucas Krause, p. 44.