Editor: **Eoin Woods**
Endava
eoin.woods@endava.com

Software Architecture in a Changing World

Eoin Woods

NEARLY 25 YEARS ago, Dewayne Perry and Alexander Wolf's seminal paper recognized software architecture as a distinct discipline[1]—albeit one that's changed considerably since then. Initially, software architecture focused on the basic problems of designing static software structures. Today, it must deal with global, always-on, Internet-connected systems with ever-changing architectures. The software architect's role has continued to evolve as well, perhaps to the point that anyone might be called an architect. In this column, I trace this evolution and consider what it predicts about the future role of software systems architects.

The Five Ages of Software Systems

We can identify five ages of software system evolution, each roughly aligning with a decade (see Figure 1).

Through the 1980s, software systems were largely monolithic. They tended to run on single computers, whether large central mainframes with terminals or single-user PCs. Software was developed as "programs," and architecture was largely a vendor concern, inherited from the platform the developers were using.

Moving into the 1990s, distributed systems became mainstream, batch processing transitioned to online processing, and the three-tier client-server architecture became standard for enterprise systems. This entailed more architectural decisions than before, but architectural style was still dictated largely by the vendors who supplied the development tools and system software.

With the Internet established as a mainstream technology in the late 1990s, organizations needed Internet-connected systems that were "always on" rather than just "online." These systems initially began as websites but gradually incorporated public UIs for business-to-consumer and business-to-business processing. Thus, architecture now had to support unpredictable and challenging nonfunctional qualities (particularly performance, scalability, and security). Vendors' main concern became supplying products, such as large-scale servers and network load balancers for scalability or firewalls for security, to meet these challenges.

Now in the 2010s, the Internet is a basic service. This has caused our systems to evolve again—from being "always on" to being "used from anywhere for anything"—they're Internet-native systems in which the Internet *is* the system. These systems combine open source components, remote Internet-connected services, and custom code; in turn, their services become part of the Internet via

 0740-7459/16/$33.00 © 2016 IEEE

publicly accessible APIs. These systems need public APIs (rather than just public UIs), provide mobile UIs as their main UI, and use flexible architectural styles (notably microservices) to assemble systems from network-accessible services. With cloud computing's emergence, vendors are concerned with supplying complete cloud-based application platforms, with many Internet systems being deployed to platforms as a service (PaaS) environments rather than traditional IT infrastructure.

On the basis of current trends, the next phase of evolution appears to be intelligent connected systems. AI—particularly machine learning—is becoming mainstream,[2] and fast, reliable networks are becoming ubiquitous, letting us connect "things" to our systems, as well as to traditional computers.[3] These systems will move beyond providing users with access anywhere to providing intelligent assistance. This will require a new architectural focus on data and algorithms as well as a move to "emergent" architecture that forms only at runtime. In this fifth era, "intelligence" will be a primary vendor concern because mainstream systems will use platforms that extend basic PaaS and Internet of Things (IoT) platforms with advanced services such as packaged machine-learning capabilities.

The Five Stages of Software Architecture

Software architecture's evolution has paralleled that of the software industry, with architects' techniques and concerns changing in response to each stage's software engineering challenges (see Figure 2).

During the monolithic systems development stage, software architecture wasn't a recognized discipline.

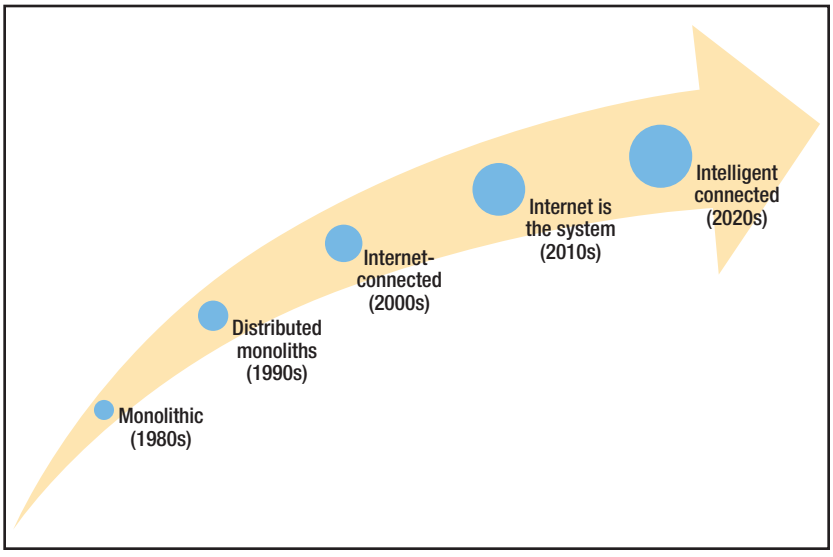

FIGURE 1. The evolution of software systems. Each age roughly aligns with a decade, beginning with the monolithic systems of the 1980s and moving toward intelligent connected systems.

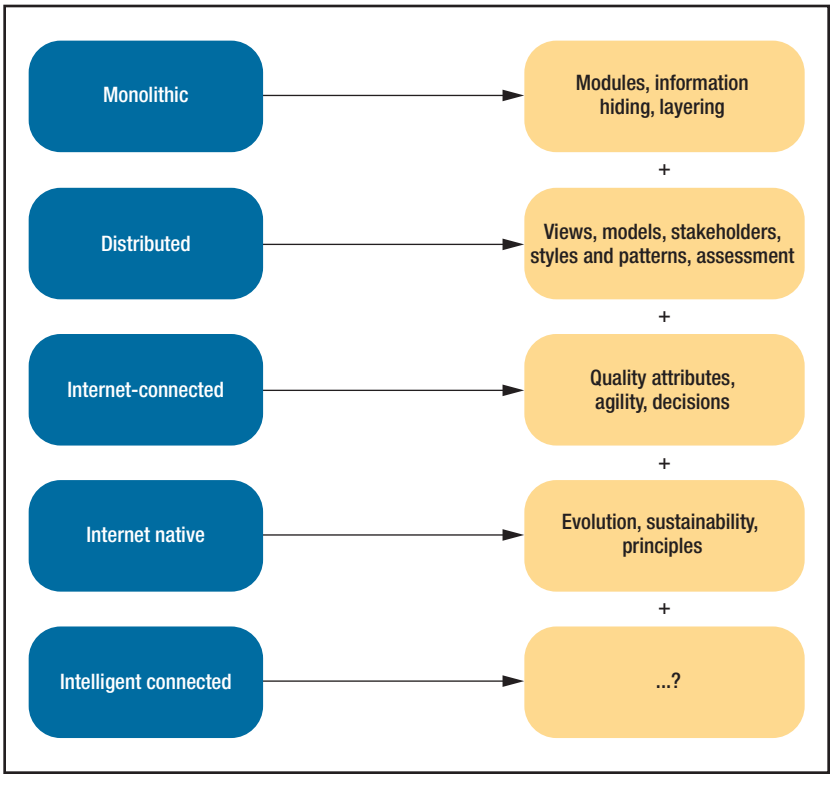

FIGURE 2. The evolution of software architecture. Architects' techniques and concerns continue to change in response to each stage's software-engineering challenges.

Decreasing	Increasing
Structural design	Data and algorithm design
Defined structure	Emergent runtime structure
Decisions	Principles, policies, and algorithms
Certainty	Probability
Operational process	Operational policy and automation
Capex	Opex

FIGURE 3. Software architecture trends as we move into the era of intelligent connected systems. Capex: capital-expenditure biased; Opex: operational-expenditure biased.

However, pioneers were already thinking about large-scale software design. This led to structural design techniques such as using modules to structure large code bases and using information hiding to encapsulate state and isolate change.[4]

As technology moved to distributed systems, the increasingly complex systems and their environments led to complex design decision making that involved long-term tradeoffs. This resulted in software architecture's recognition as a discipline and the introduction of techniques to deal with this more complex environment, such as

- viewpoints and views for architectural description,[5]
- new modeling techniques for architectural design,
- explicit identification and management of stakeholders,
- definition of proven and reusable architectural styles and patterns,[6] and
- architectural assessment techniques[7] for structured comparison of architectural options.

With Internet-connected systems came an architectural focus on

- these systems' challenging non-functional qualities (or "quality properties") and how to achieve them;[8]
- the importance of well-made, clearly communicated architectural decisions;[9] and
- how architectural design could support the agility needed to make quick changes[10] in response to the fast-moving, Internet-driven market.

Today's Internet-native systems are generally more malleable and dynamic, being composed from fine-grained network services (microservices). Systems are often built on PaaS platforms and so can include a mix of platform services, other suppliers' network-connected services, and the system's own unique services. Thus, software architects are concerned with enabling the system's rapid and reliable evolution. They also must ensure that the architecture is sustainable (and won't ossify under a mountain of technical debt) and is defined more as a set of patterns and principles[11] than as a static structure that remains stable for a long period.

As we look toward a future in which intelligent connected systems will be norm, how will software architecture's role change again?

Software Architecture's Future

Figure 3 lists trends likely to contribute to changes in the architect's role as we move into the era of intelligent connected systems. Let's explore each in a little more detail.

Although structural design isn't going away, the need to integrate dynamic architecture and intelligence into systems will make us reconsider the importance of data and algorithms. These topics don't appear prominently in the software architecture literature (for example, the original 4+1 viewpoint set had no data or information view[12]). However, when a system needs to be dynamic and intelligent, data and algorithms become important system-wide architectural concerns.

Another trend is structures moving away from being defined up front through architectural design to being defined at runtime by combining many network services to form a system. This is quite different from the "emergent architecture" that some agile practitioners discuss—they're referring to the conventional, static architecture that develops incrementally as a project progresses and more information becomes available. Instead, this trend focuses on architectural structure that's apparent only at runtime because of the services' dynamic composition.

Another recent development is architects acting less as up-front structure designers and more as overseers of a stream of informed, significant decisions made just in time for the project.[13] Even this practice will likely be affected by the challenges of intelligent connected systems. Although structural decisions will remain important, these dynamic systems will

demand greater focus on principles that guide their structure and evolution, policies that control their runtime behavior (for example, Microsoft Azure's autoscaling guidance; https://azure.microsoft.com/en-us/documentation/articles/best-practices-auto-scaling), and, in some cases, algorithms that govern their dynamic evolution.

Combined, these factors mean that software architects will more often deal with probability than certainty (or an assumption of certainty). Factors such as composing systems from third-party services, using machine learning and analytics in system design, integrating many small (IoT) devices, and using dynamic runtime environments such as PaaS platforms imply that architecture will involve statistical characteristics and trends more than static structures and definite quantities.

Our systems' operational aspects will also evolve, affecting architecture practice. Today, architects might well be involved in defining and advising on operational processes. Tomorrow, large dynamic systems will require policy-driven automation rather than carefully designed step-by-step processes, whether automated or manual.

Finally, architecture is very much the "art of the possible"—so financial constraints can limit or enable architectural decisions. As we move toward cloud-hosted systems, usage-based pricing for platforms and services and policy-driven (rather than statically defined) systems will push our financial models and budgets from capital-expenditure biased (Capex) to operational-expenditure biased (Opex). This implies that architects might be having many more conversations with accountants than they're used to.

As software systems have evolved, so has software architecture, with practices growing to meet each era's new challenges. Architecture is now an implicit part of mainstream practice. It's often seen as an activity rather than a distinct role, focusing more on decisions and principles than on definitions.

Software evolution's next phase looks even more radical. Intelligent dynamic composition, cloud platform deployment, and the connection of "things" to mainstream systems guarantee an exciting and absorbing ride!

References

1. D.E. Perry and A.L. Wolf, "Foundations for the Study of Software Architecture," *ACM SIGSOFT Software Eng. Notes*, vol. 17, no. 4, 1992, pp. 40–52.
2. "Amazon Machine Learning," Amazon Web Services, 2016; https://aws.amazon.com/machine-learning.
3. G. Kortuem et al., "Smart Objects as Building Blocks for the Internet of Things," *IEEE Internet Computing*, vol. 14, no. 1, 2010, pp. 44–51.
4. D.L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," *Comm. ACM*, vol. 15, no. 12, 1972, pp. 1053–1058.
5. "Systems and Software Engineering—Architecture Description," Int'l Org. for Standardization, Int'l Electrotechnical Commission, and IEEE, 2011; www.iso-architecture.org/ieee-1471.
6. D. Garlan and M. Shaw, *An Introduction to Software Architecture*, School of Computer Science, Carnegie Mellon Univ., 1994.
7. P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, 2002.
8. E. Woods and N. Rozanski, "Using Architectural Perspectives," *Proc. 5th Working IEEE/IFIP Conf. Software Architecture* (WICSA 05), 2005, pp. 25–35; doi:10.1109/WICSA.2005.74.
9. J. Tyree and A. Akerman, "Architecture Decisions: Demystifying Architecture," *IEEE Software*, vol. 22, no. 2, 2005, pp. 19–27.
10. P. Abrahamsson, M. Babar, and P. Kruchten, "Agility and Architecture: Can They Coexist?," *IEEE Software*, vol. 27, no. 2, 2010, pp. 16–22.
11. E. Woods, "Harnessing the Power of Architectural Design Principles," *IEEE Software*, vol. 33, no. 4, 2016, pp. 15–17.
12. P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software*, vol. 12, no. 6, 1995, pp. 42–50.
13. E.R. Poort, "Driving Agile Architecting with Cost and Risk," *IEEE Software*, vol. 31, no. 5, 2014, pp. 20–23.

EOIN WOODS is the chief technology officer at Endava. Contact him at eoin.woods@endava.com.