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

The Pragmatic Architect Evolves

Eoin Woods and George Fairbanks

IN A PREVIOUS article in this department, Eoin Woods explored how the software engineering industry has evolved and what this has meant for the role of the software architect.[1] Eoin demonstrated how the role has changed in response to the changing demands of software engineering practice.

Now, the Pragmatic Architect department is changing as well. So, it seems appropriate to look back over its history to see how it has changed and consider the topics it should cover in the future.

Industry Trends

In that previous article, Eoin observed that major changes in industrial practice seem to happen roughly every 10 years (see Figure 1). The 1980s were the Monolithic Age, characterized by centralized computing (mainframes and minicomputers). The 1990s were the Distributed Monoliths Age, seeing the widespread use of distributed systems (particularly client-server computing). In the 2000s, the Internet-Connected Age arrived, when systems were connected to a worldwide user base through the public Internet. Now, in the 2010s, we're in the Internet Is the System Age, where we've embraced the Internet as a central part of our computing environment, with systems becoming more fluid and dynamic. Next will be the Intelligent Connected Age, featuring context-aware, highly connected, and predictive systems that actively assist their users, rather than just provide useful functions.

In each age, new challenges emerge, and software architects meet them by evolving and extending software architecture practice. The Distributed Monoliths Age saw architecture being recognized as a distinct specialization, as complex system-level decisions needed to be made. The Internet-Connected Age saw a new focus on quality properties. The Internet Is the System Age has needed new, more reactive, less formal approaches that let systems evolve to respond to the demands of Internet-scale use. The Intelligent Connected Age will require us to rethink practice again as big unstructured datasets, analytics, intelligent behavior, and connected devices become common in mainstream systems.

Column Topics

In past articles in this department, three themes have dominated (see Figure 2):

- *Methodology* (software architecture methods and techniques) was the theme of nearly half of the articles.
- *The role of the software architect* (what we do and how we relate to others) filled just over a quarter of the articles.
- *System qualities* (often called nonfunctional requirements or quality attributes) constituted the theme of about a sixth of the articles.

The remaining articles covered general topics related to the department itself.

Popular topics in the articles covering methodology included architectural description (three times), agile working (twice), and dealing with requirements (twice). Topics that appeared once were architecture knowledge, governance, implementation, principles, prioritization, reengineering, refactoring, styles and patterns, technical debt, technology, and testing. This variety reflects the broad range of activities architects are involved in and the wide influence architects can have.

Two of the articles covering the role of the architect discussed how agile development affects the architect's job. The other articles on this theme discussed a range of aspects: DevOps, implementation, innovation, the need to be multiskilled across the system lifecycle, prioritization, the psychology of the role, the importance of understanding

 0740-7459/18/$33.00 © 2018 IEEE

architectural reality versus theory, how the role has changed in response to changing needs, and teamwork.

Somewhat surprisingly, fewer columns explored system qualities. Topics included the qualities of the architecture itself (twice), energy consumption (twice), usability (twice), and value (once). The articles that discussed the qualities of the architecture itself and usability started out as two long articles that each got split into two parts. Also, energy consumption appeared twice because it's a current research interest of Eoin, which probably gave it unusual prominence, compared to most architects' day-to-day concerns.

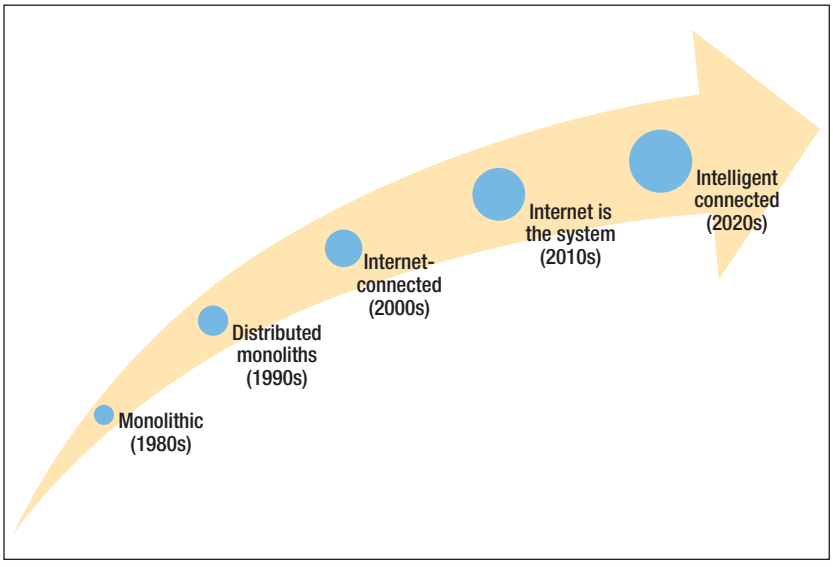

FIGURE 1. The five ages of software systems.[1] In each age, software architects meet the new challenges by evolving and extending how they do software architecture work.

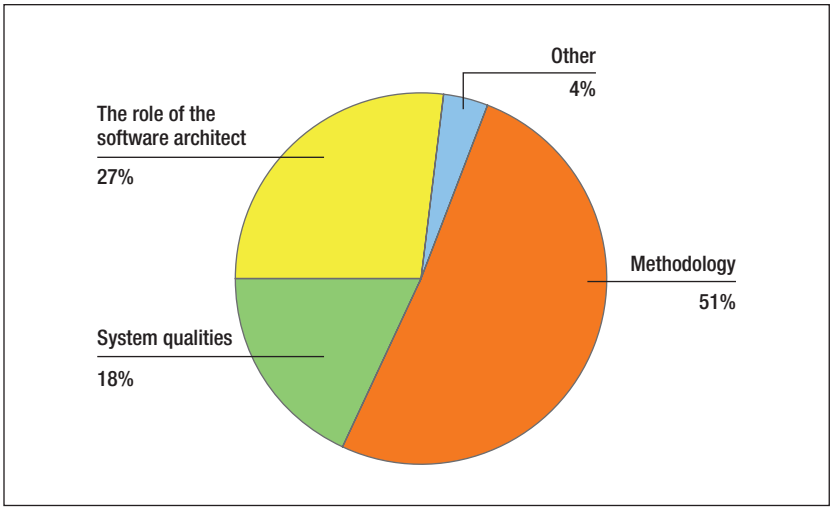

FIGURE 2. Past themes in the Pragmatic Architect.

Toward the Future

So, how should the column evolve to meet software architects' changing needs? When we looked at the current and future ages of software development practice, we found some themes that are driving the evolution of architecture practice.

The Current Age

In the Internet is the System Age, the Internet environment's commercial pressures mean that a primary concern is how to enable a system's rapid, reliable evolution while ensuring that the architecture is sustainable and won't ossify under a mountain of technical debt. This environment will likely result in architects defining their architectures more as a set of patterns and principles than as a static structure that remains stable for a long period. They'll democratize as much of the architecture work as possible by making it a team-wide responsibility. Architects will also work to produce a stream of architectural decisions as they're needed, rather than making most of the big decisions as early as possible, as we used to think was important. A good example of this trend is Eltjo Poort's Risk- and Cost-Driven Architecture.[2]

So, we think that, in the current age of software development, architects have the following concerns.

Architects are still concerned about Internet-era quality properties, particularly (rapid) evolvability, sustainability, scalability, reliability, performance, and security. This department has already talked occasionally about quality properties, and it looks as if the time is right to continue that conversation.

Also, given the need to evolve quickly, it seems inevitable that we'll

incur technical debt in order to meet the challenges of this age. So, the recognition and management of technical debt will likely be of interest to many architects for the next couple of years.

In addition, patterns and principles will be important for architecture definition and communication if we're to deal with constantly evolving, loosely coupled architectures that change rapidly. So, the department should probably explore this topic.

A related topic concerns how architecture work will get done. For development to move at the speed the Internet environment requires, it seems impossible and undesirable for the architecture to still come from the head of one person or a small group of people, as Fred Brookes originally thought best.[3] Instead, the current trend of seeing architecture as a team-wide concern (which Brookes predicted) will likely accelerate. And, we'll need to solve the challenge that Brookes identified of "how to achieve conceptual integrity while doing team design, and at the same time to achieve the very real benefits of collaboration."[3] This suggests a number of topics that the column could explore related to identifying, prioritizing, and executing team-based architecture work.

The Next Age

Regarding the Intelligent Connected Age, the following concerns seem particularly relevant.

Given the growing need for data analysis and intelligent behavior in our systems, we need to move beyond just structural design to focus more on data and algorithm design.

Also, the current trend of providing an up-front defined structure of a system will be replaced by the need to deal with a constantly evolving runtime structure.

That leads to narrow, location-specific decisions being less valuable, because they age quickly, and an increased focus on using principles, patterns, and policies to guide design behavior across the development team.

In addition, as we move to architectures that emerge at runtime and data-driven behavior, the architect's job seems likely to involve less certainty (such as "The system has 20 instances of InboundReqHdlr") and more probability ("We'll have about 2.15 × concurrent_request_volume request handlers implementing the InboundReqHdlr interface running at any point in time").

And, even before DevOps became a trend, many architects recognized the need to be deeply involved in system operation to achieve runtime quality properties. So, they have the experience of working with an operations group to define processes for operating the system. However, future systems will be difficult to operate using a traditional runbook-based approach. Therefore, in the future, this work will focus on operational policies and policy-driven automation, rather than step-by-step manual processes.

Finally, most architects tend to recoil when talk turns to project finance and topics such as budgets, cash flow, and capital and operational expenditure (*capex* and *opex*). However, as the world changes, we might need to be more concerned about these things. We can't just assume that the chief finance officer will be happy that our hardware budget has switched from large-outlay capex, depreciating over three years, to an ever-rising opex bill, paid monthly to our cloud provider. These are often uncharted territories for the finance and technology staff alike, so there's much to learn on both sides.

The Evolution of Architecture Work

This new set of concerns suggests that the Pragmatic Architect might need to cover a range of new topics in the next few years, including these:

- How we adapt our architectural thinking to meet the needs of data-driven systems (such as machine-learning systems).
- How we define, document, communicate, and evolve systems that will evolve faster and change more at runtime than is common today.
- How we make architecture a team responsibility rather than something that's "the architect's problem."
- How the new world of policy-driven automated operations affects application architecture work.
- What a move to consumption-based computing means for people outside a company's technology staff, such as the legal, finance, and risk staff. What will be easier? What will cause difficulties?

Our Changing Environment

If we also consider broader industry trends since the department was founded, we can see other changing factors that will continue to influence the department's content and form.

For example, in 2009, software architecture was a much newer field, so many of the columns introduced fundamental concepts. Today, many books, blogs, and conference talks

cover the fundamentals. So, the column is pushing ahead to emerging topics that are not yet mainstream practice, such as energy consumption as an architectural concern.

Also, the communication channels people use are changing from printed books and long-form blogs to video, audio, tweets, and short pieces on sites such as www.computer.org. In response, *IEEE Software* now republishes content on sites such as InfoQ and sponsors the Software Engineering Radio podcast, to make this content available to a wider audience.

In addition, formal software architecture job titles are becoming less common, in some cases being replaced by "technical lead" or "principal engineer." This reflects the wide acceptance of architecture as a routine part of software development but also its democratization across the software development organization. It also reflects the reduction in up-front design that software architecture work today involves and the greater focus on constant input and evolution, as we mentioned before.

Finally, software architecture practice is changing from one focused on modeling and communicating through formal documentation to being integrated into other activities throughout the software development lifecycle. Thus, it now uses a range of techniques and artifacts (including code, tests, task-specific documents, and oral communication) to communicate the essentials of an architecture to a team.

We can't predict the future, but, as Eoin said, "As software systems have evolved, so has software architecture, with practices evolving to meet each era's new challenges."[1] We're confident that the future is full of new challenges for software architects, and we're equally confident that the field will develop to meet them. The Pragmatic Architect will be trying to cover the topics required to keep architecture practitioners abreast of these developments, and allow them to prepare for this changing world.

Frank Buschmann founded the Pragmatic Architect department in 2009 and edited it until 2013. Eoin took over in 2014 and is retiring with this article. Starting in 2019, George Fairbanks will edit the department.

George is a practicing software developer with academic leanings. He studied software architecture at Carnegie Mellon University and has been involved with the SATURN (Software Engineering Institute Architecture Technology User Network) Conference for many years. Like Eoin, he has written a book on software architecture and is passionate about software design. After years of teaching object-oriented analysis and design and building systems at financial companies, he's a software engineer at Google in New York City. You can find him on the web at http://www.georgefairbanks.com.

We look forward to many more years of the Pragmatic Architect, in whatever form it needs to take to serve you, the practicing software architect. We hope that the department continues to help you in your daily work.

ABOUT THE AUTHORS

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

GEORGE FAIRBANKS is a software engineer at Google. Contact him at gf@georgefairbanks.com.

References

1. E. Woods, "Software Architecture in a Changing World," *IEEE Software*, vol. 33, no. 6, 2016, pp. 94–97.
2. E.R. Poort, "Driving Agile Architecting with Cost and Risk," *IEEE Software*, vol. 31, no. 5, 2014, pp. 20–23.
3. F.P. Brooks, *The Design of Design*, Addison-Wesley, 2010.