The Software Architect's Role in the Digital Age

Gregor Hohpe, Allianz SE

Ipek Ozkaya, Carnegie Mellon Software Engineering Institute

Uwe Zdun, University of Vienna

Olaf Zimmermann, University of Applied Sciences of Eastern Switzerland, Rapperswil

TRADITIONALLY, SOFTWARE architects were entrusted with making "decisions that are costly to change."[1] Because these decisions often had to be made early in the project, architects drew on their experience and ability to abstract to get them right. Repeated project cost and timeline overruns have demonstrated, though, that trying to plan all features and decide the system structure early in a project is difficult at best. This insight, coupled with the increasing demand for delivering high-quality software more quickly, has changed how development teams approach architectural decision making.

Reversing Irreversible Decision Making

Software teams are increasingly embracing tools and practices that help them avoid, decouple, or break down big, up-front decisions. For example, agile practices have reduced the need to make irreversible decisions at a project's very beginning, by starting development based on a simple architecture that focuses on delivering customer value quickly.[2] As teams learn more about customer needs and the system's behavior, they focus development on smaller local decisions, restructuring the system through refactoring to retain development velocity. Lean methods have taken this approach one step further by collecting user feedback not only during development but also from a productive system in a continuous build–measure–learn cycle.[3]

At the same time, rapid technology evolution has made it more difficult for architects to make decisions based solely on experience. Evidence from the field suggests that most successful architectures and their decisions are created through a collaborative team effort, rather than relying only on architects.[4] Architectural decision making has become a collective, continuous discovery-and-learning process, as opposed to being one person's responsibility.

The Code Is the Architecture

With fewer big decisions to be made, are fewer architects needed? Looking at job roles in digital companies such as Google or Spotify, we indeed find hardly any jobs titled architect. Still, these companies boast some of the planet's most innovative software application and infrastructure architectures. So, they're doing architecture, but apparently without architects.

Many architects are tasked with depicting a system's structure and behavior and conveying them to a broad audience to assure conceptual integrity.[5] However, instead of architectural decisions being documented in stacks of binders, Internet-scale companies' architectures live in the code. Discovering, discussing, and evolving the architecture are aided by structuring the code in a single, searchable repository and documenting decisions in version control systems or code review tools. Where pictures are helpful, they can be generated from code in real time using visualization techniques. Wherever a textual explanation is needed that can't be expressed in source code comments, a community wiki explains architectural decisions (for example, see the Chromium Design Documents page, www.chromium.org/developers/design-documents). Even companies with safety- and mission-critical products, which rely heavily on architecture documentation and locked-in decisions, are increasingly moving to architecting approaches that can be integrated in tools and assisted by simple decision-making processes.[6]

Architecture as a Service

Most Internet-scale companies' products and services are available in the cloud as Web APIs in a platform-as-a-service (PaaS) or software-as-a-service (SaaS) model. Such offerings are ready to be used without much consumer-side architectural considerations or the need to build up a complex application runtime environment. For example, PaaS middleware frameworks let architects and developers focus on the application domain while the platform manages most aspects of software deployment, scaling, and resilience. *Serverless architectures*, such as the one implemented by Amazon Lambda, provide a complete execution environment for application functionality. Such environments are sometimes called *functions as a service* (FaaS).[7] Thus, today's software frameworks and middleware platforms further reduce the need for architectural decision making by encapsulating many architectural decisions.

Architecture without Architects?

So, you could argue that ample resources assist teams in making and documenting architectural decisions, and recovering more quickly from bad ones, relieving architects of some of their traditional tasks. As collaborative development environments' capabilities increase, software tools appear to have further reduced the need for architects. Martin Fowler and Erik Dörnenburg underlined this perception with their recent observation that "most of what architects have done traditionally should be done by developers, or by tools, or not at all."[8] However, architects won't become redundant anytime soon—many new, even more challenging aspects of software development await architects in the digital age.

The Impact of Internet Architectures

Internet-scale systems have made software systems' architecture more important than ever. Ten years ago, developers were excited to build a distributed system; today there's hardly a system that isn't distributed or interconnected. Modern systems are expected to be horizontally scalable to thousands of machines, automatically deployable just about anywhere, observable, upgradable with zero downtime, resilient against failure, self-adapting, and antifragile. Chaos monkeys organized into simian armies put these systems to the test by randomly disabling components in production.[9] Systems without well-thought-out architecture surely won't withstand such torture.

Simple architectures that deploy a single application onto a large server no longer meet today's demand for rapid deployment and instant scalability. "Vertical scaling" and "monolith" have even become dirty words replaced by microservices architectures, which feature a much more dynamic, but also more complex, runtime behavior. So, modern software architecture not only concerns itself with structuring system functionality into objects and components but also emphasizes design for automated deployment, dynamic scaling, automated failover, predictive monitoring, and many other advanced runtime considerations.

Further adding to the complexity, Internet architectures blur system context boundaries, replacing single software applications with interconnected ecosystems of services in an API economy. No longer having control over all system components makes it difficult for teams to freeze designs, rendering design flexibility a top architectural quality.

Finally, deploying software into a connected world and onto a variety of devices elevates architectural concerns previously confined to specialized domains. For example, any system that's exposed to the Internet or an internal network becomes a target for cyberattacks, requiring today's architects to be well versed in security architecture. Likewise, applications running on mobile devices must minimize power consumption, a concern once limited to battery-operated embedded systems. Finally, the virtualization of network, computing, and storage components into software-defined infrastructure has provided development teams new flexibilities for runtime configuration and automation but also has expanded the average software architect's purview to include hardware infrastructure. "You build it, you run it" approaches such as DevOps[10] have augmented an architect's job to not only design systems but also monitor and continuously update them.

The Architect Elevator

Whereas the rapidly evolving technology landscape challenges developers, new digital business models challenge company leadership. Hardware companies must reinvent themselves as software companies, and product companies must turn into service providers. Product innovation cycles accelerate as customer expectations for speed and scale rise.

New architectures and approaches, such as the cloud and DevOps, which help traditional companies compete with digital disruptors, necessitate changes to organizational structures and working cultures. Architecture has therefore evolved from a mostly technical discipline to include even more business, social, and cultural aspects.

As technical capability becomes a critical success factor for almost any business, company strategy and technology strategy must be aligned much more closely. So, someone needs to "ride the elevator from the penthouse to the engine room of the organization" to forge this connection.[11]

Architects must also transport and combine knowledge from what used to be isolated domains, such as embedded systems, analytics, or datacenter infrastructure design, into mainstream software development teams, playing a horizontal connector role. Architects are best equipped to play this role because they typically combine a technical foundation with business acumen and communication skills.

Times of rapid change require architects who can act as mentors and bridge builders among project teams, across domains, and between different layers of the organization. Although the digital age has diminished some aspects of a traditional architect's role, such as up-front decision making and system documentation, it has placed a new critical importance on the architect's role as a linking element.

From the Golden Age to the Platinum Age?

In 2006, Mary Shaw and Paul Clements envisioned that software architecture would soon attain the status of all truly successful technologies: people would take it for granted.[12] The increasing availability of techniques, processes, and tools assisting with representing architectures, decision making, and other architecting tasks has enabled significant progress toward this vision.[13] On the other hand, rapid technology evolution and new business models have shifted the target further away.

Having been involved with the evolution of software architecture for about two decades in varying roles, we've witnessed a consolidation and, in some areas, simplification of the architect's toolbox in recent years, but also the desire to add items to it. These trends can be illustrated along three dimensions: *notation*, *process and practices* (including decision making), and *knowledge*. (For a general retrospective, see the sidebar "The Evolution of Software Architecture.")

Notation

The IEEE 1471 standard for architecture descriptions has evolved into ISO/IEC/IEEE 42010:2011, now covering additional aspects such as viewpoints, frameworks, and decision rationale.[14] However, the notation landscape has become more fragmented. It includes not only informal rich pictures and UML profiles but also novel architecture description languages such as ArchiMate (www.archimatetool.com) and AADL (Architecture Analysis & Design Language; www.aadl.info/aadl/currentsite), motivating practitioners to mix notations in a best-of-breed strategy. Nevertheless, tool support for reasoning about runtime behavior still is weak. See Grady Booch's article "Draw Me a Picture" for a general tooling vision and user wants and needs.[15] Recently, researchers and practitioners have articulated a desire for pragmatic modeling, and the first building blocks have been proposed—for example, by Simon Brown and George Fairbanks (see www.wicsa.deib.polimi.it/invited.html).

Process and Practices

Architecture design processes are now aligned better with each other and with other practices such as technical-debt management, continuous delivery, and refactoring. Quality attributes continue to play a central role in architectural analysis, with context as an important complement.[16] For instance, the Software Engineering Institute's Attribute-Driven Design method has been recently updated to include platform-specific design.[17] Practitioners have also proposed lightweight methods for architectural evaluations (reviews).[18]

Architectural decisions have evolved into a major research topic. The architecture-knowledge-management community has proposed a decision-centric view of software architecture. For instance, it has come up with metamodels, methods, and tools for decision capturing and sharing that are increasingly used in practice.[19,20] The community's current focus includes group decision making and its cognitive, behavioral, and social aspects.

A movement toward lightweight, flexible, and aligned approaches became apparent.[21] This trend continues: for example, the theme of the 2017 IEEE International Conference on Software Architecture (http://icsa-conferences.org/2017) is "Continuous architecting—exploring the role, importance, and characteristics of architecture in continuous software engineering development processes."

Knowledge

Capturing architectural knowledge in a single, definite software architecture handbook, which codifies knowledge to make it widely available, has remained an unrealized ideal.[13] In its absence, architectural tactics and patterns have been published for many application genres and technical domains, such as mobile and cloud computing. Architectural styles such as SOA (service-oriented architecture) and REST (representational state transfer), as well as supporting implementation approaches such as the microservices variant of service-based development and deployment, have become popular.

Despite this progress and the field's increased maturity, many architecting challenges lie ahead owing to the need for speed in the digital age. This need is being driven by new business models, virtual products and services, more staffing options, and increased automation and integration; it calls for additional skills.

For instance, increasing system complexity will require architects to be engaged in not just development but also operations and maintenance. This will challenge architects even more to manage systems' structure, behavior, rationale, technical debt, and quality concerns. To successfully ride the architect elevator, architects will need to strengthen their business, financial, communication, and educator skills.

In recent years, we've also seen an inversion of specialization and division of labor—a trend that contrasts with many other fields' evolution. For instance, responsibility for a particular microservice or feature requires a full-stack developer, who combines database, integration, domain logic, and user interface design skills. Architectural analysis, synthesis, and evaluation are no longer performed by individual architects but have become shared team responsibilities; architect is a virtual role on many teams now. Underlying technologies' increasing complexity challenges this trend, so time will tell whether the pendulum will swing back toward specialization.

General problem-solving and complexity management strategies such as decomposition, abstraction–refinement cycles, and asset reuse continue to be essential architect competencies. They also remain the most difficult ones to teach owing to their "experience factor." (For a discussion of software architect training, see the related sidebar. For a discussion of other information resources, see the "Trendspotting (Staying Current)" sidebar.)

In This Theme Issue

The five articles in this theme issue include two surveys and three case studies from diverse domains such as vehicle software, telecommunications, and embedded systems. These articles are by practitioners, joint teams of practitioners and academics, and academics studying the practice's state of the art. Their articles provide evidence for our claims and illustrate our observations.

THE EVOLUTION OF SOFTWARE ARCHITECTURE

IEEE Software devoted theme issues to the state of the art of software architecture in November 1995 and March/April 2006.[1] In Mary Shaw's keynote at the 2015 Software Engineering Institute Architecture Technology User Network Conf. (SATURN), she emphasized that progress has been made through the process of basic research, concept formation, development and extension, internal exploration, external exploration, and popularization.[2] However, she also observed that software engineering still doesn't have all the characteristics of an engineering discipline. For example, it lacks reference material carrying codified knowledge.

To put the software architecture field's evolution into context, Table A lists major additions to the architect's knowledge set and toolbox, starting from an early definition.[3] (Many other definitions of software architecture exist; for a collection, see www.sei.cmu.edu/architecture/start/glossary/community.cfm.)

In summary, software architecture today spans five aspects: context, elements, form, rationale, and realization. The architect in the digital age must be well versed in all of them. (See the sidebar "Software Architect Training.")

TABLE A Architectural dimensions and the evolution of the software architecture field.

	The state of the art		
Aspect	At the field's inception (1990s)[3]	After a decade (mid 2000s)[1]	Today
Context and requirements	Not an explicit part of the early definition[3]	Quality attributes (QAs) and constraints	QAs plus explicit representation of context;[4] more emphasis on business speed and value, cost and risk, architectural principles, and technical-debt management for strategic architecting
Structure	Elements • Processing • Data • Connectors Form • Properties (of elements) • Relationships (between elements)	4+1 views, components and connectors in UML and architecture description languages, informal box-and-line diagrams created by following processes and guidance in architecture design methods, and general architectural patterns; and first domain-specific architectural tactics and patterns (for example, for enterprise application architectures)[5]	More notations, such as domain-specific languages (for example, context maps in domain-driven design); more emphasis on data (for example, information viewpoints) and on architecting runtime relationships (for example, in cloud deployments); design by composition through frameworks; and many more domain-specific architectural tactics and patterns
Design decisions (reasoning behind chosen structures)	Rationale	Architectural decisions recognized as a key architectural concept in many articles and books, but no detailed coverage in most methods and tools[6]	Architecture knowledge management and decision making as a major research field and early adoption in practice (for example, inclusion in ISO/IEC/IEEE 42010:2011)
Realization	Not an explicit part of the early definition	Architecture design often embedded into end-to-end software engineering methods, International Federation for Information Processing (IFIP) subarea "realization," and model-driven software engineering and code generation attempts	Agile practices, continuous delivery, and DevOps; increased emphasis on the time dimension; better enactment and enforcement of architectural decisions (for example, architecturally evident coding styles); and continuous feedback cycles[7]

In "How Software Architects Drive Connected Vehicles," Sören Frey, Lambros Charissis, and Jens Nahm observe that the software architect's role as a single expert is often challenged in the literature and in practice. This is especially true in the context of agile methods, which were used by 80 percent of the professionals in Rainer Weinreich and Iris Groher's study, which we discuss later. Frey and his colleagues' observations from projects in the connected-vehicle domain show the importance of bundling responsibilities in the software architect role, particularly to efficiently manage complexity and spread knowledge.

In "Software Architects in Large-Scale Distributed Projects: An Ericsson Case Study," Ricardo Britto, Darja Šmite, and Lars-Ola Damm report on their experience in a traditional setup. To deal with the challenges of scale, team distribution, and monolithic legacy applications, Ericsson defers decisions to a centralized team of architects. The authors observe that the less mature a distributed team is, the more effort architects spend guarding the system's integrity and evolvability. Here, the architect's connecting or coordinating role is interpreted as a centralized role that coaches the various teams.

In "Embedded-Software Architects: It's Not Only about the Software," Pablo Antonino, Andreas Morgenstern, and Thomas Kuhn observe that many embedded-system architects have an engineering background but limited experience in software design, leading to serious deficiencies in architectural designs. In turn, experienced software architects often have little knowledge about embedded-system architectures. This article reflects well our observations that architects must keep pace with broader and more complex software architecture concerns, including aspects formerly limited to specialized domains.

In "The Architect's Role in Practice: From Decision Maker to Knowledge Manager?," Weinreich and Groher discuss results from a survey of 25 software architects, software team leads, and senior developers from 10 countries. They provide insights into how, when, and by whom architectural decisions are made. They also investigate the factors influencing decision making and the roles and responsibilities for different types of decisions. They observe that the architect's role is changing from being primarily a decision maker to being a coordinator, advisor, and knowledge manager. They view the architect as a central knowledge hub in the future.

Much in line with that thread, Damian Tamburri, Rick Kazman, and Hamed Fahimi examine "The Architect's Role in Community Shepherding." They observe that architects involved in technical or organizational changes, such as a move to DevOps or agile methods or a corporate merger, need to guide and harmonize a community of project stakeholders. The authors summarize issues that can surface as "community smells" and negatively influence system development. They position the architect's role as detecting and resolving these smells.

Several of these articles reflect the connecting or coordinating role for the architect that we've observed. All the articles discuss the architect's shifting role and responsibilities, citing reasons of increasing development speed, higher flexibility in requirements, organizational change, or the

References

1. P. Kruchten, H. Obbink, and J. Stafford, "The Past, Present, and Future for Software Architecture," *IEEE Software*, vol. 23, no. 2, 2006, pp. 22–30.
2. M. Shaw, "Progress toward an Engineering Discipline of Software," keynote at the 2015 Software Eng. Inst. Architecture Technology User Network Conf. (SATURN 15), 2015; http://resources.sei.cmu.edu/asset_files/Presentation/2015_017_101_438724.PDF.
3. 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.
4. P. Kruchten, "Contextualizing Agile Software Development," *J. Software Evolution and Process*, vol. 25, no. 4, 2013, pp. 351–361.
5. C. Hofmeister et al., "A General Model of Software Architecture Design Derived from Five Industrial Approaches," *J. Systems and Software*, vol. 80, no. 1, 2007, pp. 106–126.
6. M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley Professional, 2002.
7. M. Erder and P. Pureur, "What's the Architect's Role in an Agile, Cloud-Centric World?," *IEEE Software*, vol. 33, no. 5, 2016, pp. 30–33.

SOFTWARE ARCHITECT TRAINING

Software architect typically isn't an entry-level position or responsibility. So, software architecture curricula can be found both in academia and as part of continuing education delivered through classroom and distance-learning programs. Such education might also come as on-the-job training based on mentor–protégé (or master–apprentice) relationships.

BOOKS

Many introductory and more advanced software architecture books exist that can help structure a curriculum to teach (parts of) the architect's skill sets. A number of book recommendations are online. For instance, George Fairbanks reviewed a comprehensive set of essential books in a June 2015 blog post and video (http://georgefairbanks.com/blog/software-architecture/book-recommendations).

ACADEMIC CURRICULA

In *Computer Science Curricula 2013*, the use of components in design and basic software architecture concepts and standard architectures (for example, client-server, *n*-layer, transformation centered, and pipes and filters) are Core Tier-2 topics.[1] (Core Tier-1 topics should be in all computer science programs; individual programs choose which Core Tier-2 topics to cover.) Furthermore, architecture patterns and specialized architectures such as parallel architectures are considered elective topics.

Computer Science Curricula 2013 covers only undergraduate education; graduate programs can and should cover architecture topics more deeply. Furthermore, graduate courses on other topics should pay specific attention to architectural concerns. For instance, requirements classes should cover architecturally significant requirements, and courses on software evolution and maintenance should cover topics such as DevOps, monitoring and improving systems at runtime, and architectural refactoring.

For instance, the University of Vienna has developed a curriculum that follows many suggestions from *Computer Science Curricula 2013*. The required undergraduate course Software Engineering 2 features an introduction to software architecture, including topics such as architecture disciplines and basic component-and-connector modeling. The required master's course Advanced Software Engineering includes architecture topics such as domain-driven design, advanced component-and-connector modeling, architectural views, architectural styles and patterns, design decisions, model-driven development, domain-specific languages, architecture analysis, architecture in the organization, and architecture in the development process. Additionally, the elective course Distributed Systems Engineering covers distributed-system

introduction of agile methods. Another common theme is that architecting modern systems requires broader and deeper knowledge of various types, including software areas other than design and architecture, hardware, and domain-related knowledge. In addition, several articles feature sidebars envisioning how the software architect's role might evolve.

Two of the departments in this issue also tie in with our theme. In the Insights department, consulting IT architect Eltjo Poort makes the case for an explicit representation of architectural evolution along the time axis. He also reports on his IT architect community's experiences with architecture roadmapping. His approach supports the modern architect's connector role, assisting both project and product managers with strategic planning and risk management.

Finally, in the Pragmatic Architect department, Eoin Woods presents his view on the evolution of software architecture's past, present, and future, which complements the analysis in this article. He identifies five ages of software systems and five corresponding stages of software architectures. He also calls out six future trends for architecting practices, including more focus on data and algorithm design, emergent runtime structure, and operational policy and automation.

Software architecture has become broader and more complex, presenting students and practitioners with the challenge to stay up to date and hands-on, with not only an ever-faster stream of new technologies and open source projects but also new concepts and concerns. Whereas being conversant in object-oriented design was once largely sufficient to design systems, it's now but one of many aspects. Thought leadership, mentoring, and conveying complex concepts in patterns, including architectural patterns. Other elective courses cover specific kinds of architectures, such as cloud, parallel, process-driven, cooperative-systems, or game architectures. All these courses dedicate 40 to 50 percent of their time to delivering conceptual knowledge, supplemented with practical hands-on exercises, including programming, designing, studying architectures, and reviewing case studies.

The University of Applied Sciences of Eastern Switzerland, Rapperswil balances theory and practice in a similar way. For instance, the advanced undergraduate course Application Architecture offers 14 lessons accompanied by exercises and self-study assignments on quality attributes, architectural principles and patterns (such as loose coupling and layers), context and component modeling, architectural decisions, dependency injection containers, enterprise integration patterns, service orientation, and domain-driven design. Case studies and examples illustrate the abstract concepts and refine them for domains and application genres such as information systems (also known as enterprise applications), distributed control systems, and cloud services. Other courses that cover architectural topics include Distributed Software Systems, Advanced Patterns and Frameworks, and Cloud Solutions.

INDUSTRY TRAINING

A number of education and certification programs targetting professionals feature topics similar to those in academic curricula. Examples include the Software Engineering Institute's Professional Software Architecture Certificate (www.sei.cmu.edu/training/certificates/architecture/professional.cfm), IASA Software Architect Certification (http://iasaglobal.org/certifications), and the Open Group Certified Architect (Open CA) Program (www.opengroup.org/openca/cert). A variety of corporate programs also provide architect training and certification—for example, at ABB, GE, IBM, Raytheon, and Siemens.[2]

In addition, online courses are available through massive open online course platforms, such as Doug Schmid's Pattern-Oriented Software Architectures courses (www.dre.vanderbilt.edu/~schmidt/Coursera).

Continuing-education programs face the same challenges as academic programs, having to balance current, easily applicable content and timeless, universal problem-solving competencies. Just as in software architecture design, tradeoffs are inevitable.

References

1. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, ACM and IEEE, 2013; www.acm.org/education/CS2013-final-report.pdf.
2. F. Buschmann, "What an Architect Needs to Know: Experiences from the Siemens Curriculum for Software Engineers," 2016; http://gotocon.com/dl/jaoo-aarhus-2010/slides/FrankBuschmann_WhatArchitectsNeedToKnowExperiencesFromTheSiemensCurriculumForSoftwareEngineers.pdf.

approachable terms has therefore become more important than ever. Architects are uniquely qualified to play this role.

Being relieved of some traditional tasks, such as centralized decision making and documenting architectures, is a welcome break for architects who navigate not only a more complex technical environment but also visit the corporate penthouse more frequently to align business and technical strategy. Isn't it a great time to be an architect?

Acknowledgments

Ipek Ozkaya's contribution to this article is based on work funded and supported by the US Department of Defense under contract FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution.

References

1. G. Booch, "Architecting the Unknown," keynote at the 2016 Software Eng. Inst. Architecture Technology User Network Conf. (SATURN 16), 2016; www.youtube.com/watch?v=RJ3v5cSNcB8.
2. P. Abrahamsson, M.A. Babar, and P. Kruchten, "Agility and Architecture: Can They Coexist?," *IEEE Software*, vol. 27, no. 2, 2010, pp. 16–22.
3. E. Ries, *The Lean Start-Up*, Crown Business, 2011.
4. M.P. Robillard and N. Medvidović, "Disseminating Architectural Knowledge on Open-Source Projects," *Proc. 38th ACM/IEEE Int'l Conf. Software Eng.* (ICSE 16), 2016, pp. 476–487.
5. J. Klein, "What Makes an Architect Successful?," *IEEE Software*, vol. 3, no. 1, pp. 20–22.
6. R.L. Nord et al., "Missed Architectural Dependencies: The Elephant in the Room," *Proc. 13th Working IEEE/IFIP Conf. Software Architecture* (WICSA 16), 2016, pp. 41–50.
7. M. Roberts, "Serverless Architectures," 4 Aug. 2016; http://martin

TRENDSPOTTING (STAYING CURRENT)

The following sources can help keep you up to date on software architecture and related subjects.

ONLINE RESOURCES

Many online resources and communities can help developers and architects educate themselves about basic concepts and emerging trends. Popular community websites include these:

- arc42, http://arc42.org. This site offers templates and checklists for architectural descriptions.
- DZone, https://dzone.com. This site offers refcards for many concepts and technologies.
- InfoQ, www.infoq.com. This site offers e-magazines compiling articles on popular topics. In particular, see www.infoq.com/architecture.
- Microsoft's Patterns & Practices, https://msdn.microsoft.com/en-us/library/ff921345.aspx.
- The SATURN (Software Engineering Institute Architecture Technology User Network) Blog, http://insights.sei.cmu.edu/saturn.

A number of personal websites and blogs also cover software architecture topics regularly—for example, Martin Fowler's, http://martinfowler.com/design.html, and Philippe Kruchten's, https://philippe.kruchten.com/category/architecture. In addition, many episodes of Software Engineering Radio investigate architectural concerns, either explicitly (see www.se-radio.net/tag/architecture) or under related topics such as the cloud, distributed systems, and DevOps.

CONFERENCES

Practitioner conferences come in many forms. For instance, the SATURN conference is in its 13th year (www.sei.cmu.edu/saturn/2017), and O'Reilly launched a software architecture conference series in 2015 (http://conferences.oreilly.com/software-architecture). Developer conferences such as QCon typically include software architecture topics in dedicated tracks. Similarly, the hybrid industry–academia PLoP (Pattern Languages of Programming) worldwide conferences and their regional counterparts (for example, EuroPLoP) also cover software architecture, although they have a broader scope.

Major academic conferences are ICSA (International Conference on Software Architecture) and ECSA (European Conference on Software Architecture).

PERIODICALS

Architecture articles appear regularly in major journals, such as the *Journal of Systems and Software*, *IEEE Transactions on Software Engineering*, or *Information and Software Technology*. Also, magazines such as *IEEE Software* often publish architecture-related articles and theme issues.

STANDARDS BODIES

Relevant de jure or de facto standards bodies include

- the ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission), which, for example, jointly provide SQuaRE (Systems and Software Quality Requirements and Evaluation) and *Committee Draft (CD) 42020. Systems and Software Engineering—Architecture Processes*;
- the Open Group, which, for example, provides TOGAF (The Open Group Architecture Framework) and ArchiMate; and
- the OMG (Object Management Group), which provides UML, SPEM (Software & Systems Process Engineering Metamodel), and RAS (Reusable Asset).

Such standards and specifications typically define terms, metamodels, and notations and sometimes methods, techniques, and tool interfaces for architectural content creation and processing. Occasionally, supporting material such as primers and templates are available—for example, a template exists for the ISO/IEC/IEEE 42010:2011 standard for architecture descriptions (see www.iso-architecture.org/ieee-1471/templates/42010-vp-template.pdf).

fowler.com/articles/serverless.html.
8. M. Fowler and E. Dörnenburg, "Architecture without Architects," presentation at 2016 Craft Conf., 2016; www.ustream.tv/recorded/86152575.
9. Y. Izrailevsky and A. Tseitlin, "The Netflix Simian Army," blog, 19 July 2011; http://techblog.netflix.com/2011/07/netflix-simian-army.html.
10. L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*, Addison-Wesley, 2015.

11. G. Hohpe, *37 Things One Architect Knows about IT Transformation: A Chief Architect's Journey*, 2016; https://leanpub.com/37things.
12. M. Shaw and P.C. Clements, "The Golden Age of Software Architecture," *IEEE Software*, vol. 23, no. 2, 2006, pp. 31–39.
13. M. Shaw, "Progress toward an Engineering Discipline of Software," keynote at the 2015 Software Eng. Inst. Architecture Technology User Network Conf. (SATURN 15), 2015; http://resources.sei.cmu.edu/asset_files/Presentation/2015_017_101_438724.PDF.
14. *42010-2011—ISO/IEC/IEEE Systems and Software Engineering—Architecture Description*, Int'l Org. for Standardization, Int'l Electrotechnical Commission, and IEEE, 1 Dec. 2011; http://ieeexplore.ieee.org/document/6129467.
15. G. Booch, "Draw Me a Picture," *IEEE Software*, vol. 28, no. 1, 2011, pp. 6–7.
16. F. Torres, "Context Is King: What's Your Software's Operating Range?," *IEEE Software*, vol. 32, no. 5, 2015, pp. 9–12; http://doi.ieeecomputersociety.org/10.1109/MS.2015.121.
17. H. Cervantes and R. Kazman, *Designing Software Architectures: A Practical Approach*, Pearson, 2016.
18. E. Woods, "Industrial Architectural Assessment Using TARA," *Proc. 9th Working IEEE/IFIP Conf. Software Architecture* (WICSA 11), 2011, pp. 56–65.
19. M.A. Babar et al., eds., *Software Architecture Knowledge Management: Theory and Practice*, Springer, 2009.
20. Z. Li, P. Liang, and P. Avgeriou, "Application of Knowledge-Based Approaches in Software Architecture: A Systematic Mapping Study," *Information and Software Technology*, vol. 55, no. 5, 2013, pp. 777–794.
21. M. Keeling, "Lightweight and Flexible: Emerging Trends in Software Architecture from the SATURN Conferences," *IEEE Software*, vol. 32, no. 3, pp. 7–11.

ABOUT THE AUTHORS

GREGOR HOHPE is the chief IT architect at Allianz SE. His interests include large-scale IT transformation and the role of architects and architecture in large enterprises. Gregor received a master's in computer science and a master's in engineering management, both from Stanford University. He's the author of *37 Things One Architect Knows about IT Transformation: A Chief Architect's Journey* and is on the *IEEE Software* advisory board. Contact him at info@enterpriseintegrationpatterns.com or follow him on Twitter @ghohpe.

IPEK OZKAYA is a principal research scientist at the Carnegie Mellon Software Engineering Institute. Her interests include development and application of techniques for improving software architecture practices and practices to manage technical debt in large-scale, software-intensive systems. Ozkaya received a PhD in computational design from Carnegie Mellon University. She's on the *IEEE Software* advisory board. Contact her at ozkaya@sei.cmu.edu or follow her on Twitter @ipekozkaya.

UWE ZDUN is a full professor of software architecture at the University of Vienna. His research interests include software patterns, software architecture, language engineering, service-oriented architecture, distributed systems, and object orientation. Zdun received a doctorate in computer science from the University of Essen. He's on the *IEEE Software* editorial board. Contact him at uwe.zdun@univie.ac.at.

OLAF ZIMMERMANN is a professor of software architecture and an institute partner at the Institute for Software at the University of Applied Sciences of Eastern Switzerland, Rapperswil. His research interests include service-oriented computing and architectural knowledge management. Zimmermann received a doctorate in computer science from the University of Stuttgart. The Open Group has awarded him a Distinguished IT Architect (Chief/Lead) Certification. And he's on the *IEEE Software* editorial board. Contact him at olaf.zimmermann@hsr.ch.