Editor in Chief: **Diomidis Spinellis**
Athens University of Economics and Business, dds@computer.org

The Changing Role of the Software Architect

Diomidis Spinellis

BEING A GOOD software architect has never been easy. Changes in the software industry are making the job even more challenging. The key drivers are the rising role of software in systems of all kinds and their operation; more emphasis on reuse, agility, and testability during software development; and several quality elements increasingly affected by architectural choices. Most of these drivers have always concerned software architects. What's changing is that the requirements they impose appear much more frequently and are more difficult to satisfy.

Systems and Operations

The systems running the software are becoming ever-more intertwined with it. In the past we could often afford to abstract the system on which the code was running and view software architecture in isolation. John von Neumann taught us how to do that, and it has worked wonderfully for more than half a century. Nowadays, however, software increasingly determines the architecture of a complete system (this includes hardware and "wetware"—humans), which can in turn affect the software's architecture. Consider software that coordinates a car's electronic control units, binds together a large organization's departments, or dishes out work among datacenters around the planet. In each case, the system's structure constrains the software's architecture, which in turn can determine the system's quality (more on that later) and financial viability.

For instance, consider the division of work among multiple (cloud) datacenters. Their setup, cost, and communication affordances will guide architectural choices in the software that processes work requests. The resultant system's architecture determines the offering's performance, reliability, security, and efficiency.

Software architects must also take an enterprise's operations into account. As this issue's guest editors point out in their introduction, the DevOps movement has demonstrated the importance of closely collaborating with IT professionals when you're developing software. Architects must play their part by designing software that's easy to build, configure, test, deploy, monitor, scale, and troubleshoot. For each link in this chain there are important domain-specific designs, tools, and techniques that architects must select and apply.

Reuse

The increasing use of open source third-party components is a welcome and partly unexpected development. Open source software libraries, often concocted by hobbyists and maintained by volunteers through pull requests on GitHub, are finding their way into mass-market TVs, smartphones, routers, enterprise applications, and even medical equipment. Software package managers, such as npm, pip, and Maven, enable and promote deep, complex dependencies between hundreds of components, which we take for granted—until they fail.

Given this state of affairs, modern software architects must decide which components to use as open source software and which to purchase or build in-house. In their de-

IEEE Software Mission Statement

To be the best source of reliable, useful, peer-reviewed information for leading software practitioners—the developers and managers who want to keep up with rapid technology change.

 0740-7459/16/$33.00 © 2016 IEEE

cisions, they must balance factors such as quality, technological risk, and licensing restrictions, as well as each component's fit with the rest of the project and its total cost of ownership. True to choices that building architects have faced for centuries, this is a multi-objective optimization with many unknowns. As Philippe Kruchten put it, "The life of a software architect is a long and rapid succession of suboptimal design decisions taken partly in the dark."[1]

Having chosen the parts to reuse, architects must align the interfaces of diverse modules with the project's design. Then they must come up with a plan for maintaining third-party components and integrating their changes back into the project.

Agility

Today's software architects are instrumental in delivering and managing development agility. Early on, they must consider and decide on the change forces that the software will most likely face. Then, they need to devise a design that not only is amenable to the change coming from these forces but also maintains its conceptual integrity when subjected to them.

Some architects' choices will be (or turn out to be) incorrect. In such cases, they must come up with fresh software designs that accommodate novel requirements with gusto. These must result in a new powerful software structure with minimal disruption to existing code. And, just as the software's initial architecture must enable its efficient implementation, modified architectures must allow efficient refactoring.

Testability

System architectures have long been designed for testability: consider the power-on self-test of most electronic equipment or the test pads and modes on complex integrated circuits. Software architects can help developers with designs that are easy to unit-test, components that can be tested in isolation, interfaces that be easily driven, and operations that are amenable to automation. For instance, a microservice with a public RESTful HTTP interface expressed as an Open API specification (formerly called Swagger) might be easier to test than a proprietary, internal Java interface in a monolith. (REST stands for Representational State Transfer.)

Quality

As key quality elements affected by architectural choices in the digital age, consider usability, security, reliability, and efficiency.

Usability quirks and wonders, on devices ranging from phones to desktops, often come about through bad and good architectural decisions. An appropriate abstraction can provide a coherent and pleasant user experience; failing to provide one can result in a hodgepodge of nauseatingly inconsistent interfaces. Adding insult to injury, a first-class software design will deliver orthogonal functionality with negligible implementation cost, whereas the deficient model might require expensively coding thousands of error-prone special cases. For example, think of how some popular OSs and their applications handle the changing of a display's resolution.

Security's importance increases as each new device on the planet connects to the Internet. The software's architecture will determine where data is stored, where it's processed, what becomes encrypted, and how the software guarantees

the maintenance of its security attributes, such as the data's confidentiality and integrity. Architecture also determines the software's attack surface: the places where enemies can target their assaults. The goal here is to minimize the attack surface and bring it close to (physical or logical) places where it can be effectively safeguarded. The differing number of vulnerabilities discovered in competing OSs or Web browsers exemplifies this situation. A large part of the difference is due to architectural choices.

In the past, the reliability of most software depended mainly on its implementation. Now architecture is playing an increasing role, for two reasons. First, service-oriented architectures and cloud-based infrastructures depend on connectivity that might be intermittent or fail entirely, problems that are exacerbated by wireless and mobile data links. Second, the Internet of Things has software work with ephemeral and fickle devices. Reliability in these settings can't be micromanaged at the implementation level; it must be architected. The software architect is responsible for adopting tactics such as caching, redundancy, idempotence, and graceful degradation to handle varied and sometimes unexpected failures. Architectural patterns such as Circuit Breaker or Watchdog can be applied to achieve the desired reliability qualities, following a "design for failure" principle.

Efficiency has always been an important concern in software architecture. What's changing is the breadth of choices and the diversity of outcomes and performance indicators. Will an expensive calculation be performed on a phone's CPU, draining its battery, or on the cloud, increasing the latency? Will some time-critical processing occur on a powerful multicore CPU, a cluster of servers, or a GPU? Will service elasticity be provided by the dynamic leveraging of costly cloud-based resources or by the peer-to-peer distribution of the load among free but unreliable clients?

All the choices and challenges facing the modern software architect are staggeringly intertwined. Each of them both constrains the others and affects multiple key outcomes. Minor decisions often have huge implications. As in the physical world, this is the difference between engineering and architecting.

Reference

1. P. Kruchten, "The Architects—and the Software Architecture Team," *Proc. 1st Working IFIP Conf. Software Architecture* (WICSA 99), 1999, pp. 565–584.

HOW TO REACH US

AUTHORS

For detailed information on submitting articles, write for our editorial guidelines (software@computer.org) or access www.computer.org/software/author.htm.

LETTERS TO THE EDITOR

Send letters to

Editor, *IEEE Software*
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
software@computer.org

Please provide an email address or daytime phone number with your letter.

ON THE WEB

www.computer.org/software

SUBSCRIBE

www.computer.org/software/subscribe

SUBSCRIPTION CHANGE OF ADDRESS

Send change-of-address requests for magazine subscriptions to address.change@ieee.org. Be sure to specify *IEEE Software.*

MEMBERSHIP CHANGE OF ADDRESS

Send change-of-address requests for IEEE and Computer Society membership to member.services@ieee.org.

MISSING OR DAMAGED COPIES

If you are missing an issue or you received a damaged copy, contact help@computer.org.

REPRINTS OF ARTICLES

For price information or to order reprints, email software@computer.org or fax +1 714 821 4010.