The Future of Software Architecture

David Garlan

A critical issue in the design and construction of any complex software system is its architecture: that is, its organization as a collection of interacting elements – modules, components, services, etc. [SG96]. A good architecture can help ensure that a system will satisfy its key functional and quality requirements, including performance, reliability, portability, scalability, and interoperability. A bad architecture can be disastrous.

Over the past two and a half decades software architecture has received considerable attention as an important subfield of software engineering [BCK12, SC06]. Practitioners have come to realize that getting an architecture right is a critical success factor for systems. They have begun to recognize the value of making explicit architectural choices, and leveraging past architectural designs in the development of new systems. Today there are numerous books on architectural design, regular conferences and workshops devoted to software architecture, a growing number of commercial tools to aid in aspects of architectural design, courses in software architecture, major government and industrial research projects centered on software architecture, and an increasing number of formal architectural standards. Codification of architectural principles, vocabulary, methods, and practices has begun to lead to repeatable processes of architectural design, criteria for making principled tradeoffs among architectural decisions, tools for evaluating properties of architectures, and standards for documenting, reviewing, and implementing architectures [B+03, C+11].

However, despite this progress, as engineering disciplines go, the field of software architecture remains relatively immature. While the foundations of an engineering basis for software architecture are now clear, the changing face of technology and its roles in society raise a number of important challenges for the next few decades.

Agility and Architecture: A critical issue for software architecture is how best to integrate architectural practices into the broader software development processes [BT03, Fai10]. At the core of this issue are questions such as "How much architecture is enough?" and "When should architectural design be performed?" Although considerable progress has been made in establishing processes for architectural design and evaluation, an on-going concern is the role of architecture in agile processes. It may seem that architecture would have no place in a rapidly evolving system, where an organization's development practices deemphasize the creation of artifacts that do not manifest themselves directly as user-visible functionality. But further reflection suggests that architecture is, in fact, an *enabler* of agility, not an impediment to it. We can see this in the emergence of extensible platforms (each implying a set of architectural commitments), which can be rapidly customized with new plug-ins and applications [Bos09].

Some would argue that architecture is important, but that it will emerge naturally as a system evolves through extension and refactoring. This idea is reinforced by the fact that most software development efforts today do not require a significant amount of bold new architectural design: the most important design decisions have been made earlier, are fixed by pre-existing conditions, or are a *de facto* architectural standard in the respective industry. Choices of operating system, servers, programming language, database, middleware, and so on, are pre-determined in the vast majority of software development projects, or have a narrow range of possible variations.

Architectural design, when it is really needed because of project novelty, has an uneasy relationship with agile practices, including the recent trend towards "DevOps" and micro-services. Unlike the functionality of the system, it cannot easily be decomposed into small, incremental chunks of work. The difficult aspects of architectural design are driven by systemic quality attributes (security, availability, scalability, etc.), which cannot be easily decomposed. It is clear that architectural design and the incremental building of a system must go hand-in-hand. The important question then becomes: How to address architectural issues over time, in a way that will lead to an architecture enabling incremental development?

Network-Centric Computing: The primary computational model for systems has shifted from localized computation to a network-centric model. This trend has a number of consequences for software engineering and software architecture. Historically, software systems have been created as closed systems, developed and operated under control of individual institutions. Architectures for such systems are largely static – allowing minimal run-time restructuring and variability.

However, within the world of pervasive services and applications available over networks, systems may not have such centralized control. A new set of software architecture challenges emerges. First, is the need for architectures that scale to the size and variability of the Internet. While many of the traditional architectural paradigms apply, the details of their implementation and specification need to change. Second, is the need to support computing with dynamically-formed, task-specific, compositions of distributed autonomous resources. Component composition in this setting is difficult because it is hard to determine what assumptions each component makes about its operating context and whether their combined functionality is what you need [GAO09]. Third, is the need to architect systems that can take advantage of the rich computing base enabled by network-centric computing. Service-oriented architectures are a step in that direction, but are often cumbersome. Moreover, while cloud computing platforms provide almost unlimited access to storage and computation, exploiting these requires architectures that can scale to large volumes of data and a huge array of available services. Fourth, is a need to ensure adequate security and privacy in such systems

Pervasive and Cyber-physical Systems: A related trend is towards pervasive computing, in which the computing universe is populated by a rich variety of software-enhanced devices: toasters, home heating systems, smart cars, etc. Architecture for such systems faces a number of challenges. First, we need architecture design tools and models suited to systems that combine both physical and software elements – cyber-physical systems. Second, architectures for these systems will have to be more flexible than they are today. In particular, devices, components, services and other computational elements may come and go in an unpredictable fashion. Handling reconfiguration dynamically, while guaranteeing uninterrupted service, is a hard problem in its own right (see below). This is complicated by the need to include humans in the computational process. Third, is a need for architectures that bridge the gap between technology and technologically-naïve users [GDRS12]. Currently, our computing environments are largely configured manually and managed explicitly by the user. This may be appropriate for environments in which we have only a few, relatively static, computing devices, but does not scale to environments with hundreds of devices. We must therefore find architectures that provide automated management of computational services. Such architectures need to raise the level of abstraction for configuring pervasive systems, allowing users to focus on their high-level tasks [GSSS02].

Fluid Architectures: Today's systems are highly dynamic: new applications are downloaded onto mobile platforms; new devices are installed in homes; mobile devices enter and leave our computing environment; new web services become available; system upgrades are required to address discovered security vulnerabilities, etc. Such fluidity requires architectures that support ease of change and adaptation, while (in many cases) providing uninterrupted service. This, in turn, leads to a need for architectures that take a stronger role in ensuring their own health and quality. Such systems are often referred to as self-adaptive systems or autonomic systems, and are the subject of considerable on-going research [C+09].

One emerging technique is to adopt a control systems view of the problem: each system is coupled with a control layer responsible for monitoring the state of that system, detecting opportunities for improvement, and effecting change as the system runs. Central to such an approach is the use of system models in the control layer. These models reflect the current state of the system, and are used to detect problems and decide on courses of action. It turns out that architectural models are particularly useful in this regard [GSC09]. As such, architectural models now become essential not only as design-time artifacts, but also as run-time artifacts. This mirrors a larger trend in software engineering in which design-time activities, artifacts, and models now become part of the run-time environment (e.g., continuous testing, requirements change at run-time, etc.).

Socio-technical Ecosystems: The emergence of platforms (e.g., Android and iOS) as a basis for modern system development has brought the challenge that software architects must now consider the socio-technical ecosystems that arise around the platform and are necessary to ensure its sustainability [Bos09]. Such ecosystems include not only the *platform* developers, but also the much larger community of developers who provide platform *extensions* (apps, services, etc.), users who must purchase and install those extensions, governance rules and processes to qualify potential extensions, incentive systems to motivate developers of extensions, legal and economic systems, and so on. Today we have weak understanding of the interrelationship between all of these moving parts, and the way in which architectural design facilitates a sustainable ecosystem. Consider, for example, the architectural issue of the kinds of interaction the platform will support between components provided by third parties. Limited interaction can provide stronger guarantees, but may reduce flexibility, and will affect the extent to which a component developer can depend on components built by others.

Note: This article was derived in part from [Gar14].

REFERENCES

[AG97] Allen, R. and Garlan, D. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, Vol. 6(3):213-249 July 1997.

[Bos09] Bosch, J. From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference (SPLC '09)*, Pittsburgh, PA, USA, 111-119. 2009

[BCK12] Bass, L., Clements, P. and Kazman, R. *Software Architecture in Practice, Third Edition*. Addison Wesley, 2012.

[B+03] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. Pattern Oriented Software Architecture: A System of Patterns, Volume 1. Wiley, 1996.

[Bos00] Bosch, J. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. ACM Press/Addison-Wesley Publ. Co. 2000.

[BT03] Boehm, B. and Turner, R. *Balancing Agility and Discipline: A Guide for the Perplexed.* Addison-Wesley/Pearson Education, 2003.

[C+09] Cheng, B. et al. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems.* LNCS, 5525. Springer, pp. 1-26. 2009.

[C+11] Clements, P. et al. *Documenting Software Architectures: Views and Beyond, Second Edition*. Addison-Wesley, 2011.

[Far10] Fairbanks, G. *Just Enough Software Architecture: A Risk-Driven Approach*. Marshal & Brainerd, 2010.

[Gar14] Garlan, D. Software Architecture: a Travelogue. *The Future of Software Engineering*. Hyderabad, India, May 2014

[GAO09] Garlan, D., Allen, R. and Ockerbloom, J. Architectural Mismatch: Why Reuse is Still So Hard. *IEEE Software*, pp. 66-69, July 2009.

[GDRS12] Garlan, D., Dwivedi, V., Ruchkin, I. and Schmerl, B. Foundations and Tools for End-User Architecting. *Large-Scale Complex IT Systems. Development, Operation and Management,* 17th Monterey Workshop, vol. 7539:157-182 LNCS, Springer, 2012.

[GSC09] Garlan, D., Schmerl, B. and Cheng, S. Software Architecture-Based Self-Adaptation. In M. Denko, L. Yang and Y. Zhang editors, *Autonomic Computing and Networking*. Springer, 2009.

[GSSS02] Garlan, D., Siewiorek, D., Smalagic, A. and Steenkiste, P. Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*,. 1(2):22-31, April 2002.

[KC03] Kephart, J.O., Chess, D.M. The vision of autonomic computing. *IEEE Computer*,36/1, Jan 2003.

[SC06] Shaw, M. and Clements, P. The Golden Age of Software Architecture. *IEEE Software*, 23(2), pp. 31-39. March/April 2006.

[SG96] Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.