

Software Architecture: Framing Stakeholders' Concerns

Patricia Lago, *VU University Amsterdam*

Paris Avgeriou, *University of Groningen, the Netherlands*

Rich Hilliard, *Freelance Software Architect*

It has long been recognized that one of the key benefits of architecting our systems is managing their complexity. This complexity arises from many factors: the needs and constraints of the multitude of system stakeholders (including users, owners, future operators, and the current development team); the political, social, and other factors from the environment in which the system is embedded; the realities and constraints of the system's development, implementation,

maintenance, and operation in relation to available resources; and, of course, the intended properties of the system itself. Taken together, these diverse interests are a system's *stakeholder concerns*. But what is a stakeholder concern? ISO/IEC 42010 defines it this way: "A stakeholder concern is any interest in a system relevant to one or more of its stakeholders."

An Important Separation

The ISO/IEC 42010 standard further observes that a concern can "pertain to any influence on a system in its environment including: developmental, technological, business, operational, organizational, political, regulatory, or social influences." Therefore, stakeholder concerns cover any and all the things the architect must care about in envisioning the system, meeting its requirements, overseeing its development, and certifying it for use. The notion of concern derives from the principle of "separation of concerns," which dates back to

the early history of software engineering. (See the sidebar "A Brief History of Concerns" for more background.) The idea is that any system should be organized ("decomposed," in common parlance) in such a manner that each of its elements frames specific concerns. This principle gives us a way to manage system complexity by breaking the overall system challenge into well-defined—ideally, loosely coupled—sub-problems.

Stakeholder concerns articulate the dimensions of what the architect must consider relevant to a software system. These usually result in specific requirements, design constraints, decisions, and priorities. Software architects must identify and manage a multitude of architectural concerns to devise a successful architecture; Figure 1 lists some examples of the range of concerns an architect might confront.

Within software architecture, the principle of separation of concerns has led to the use of multiple architecture views to model and define ar-

A Brief History of Concerns

Edsger Dijkstra, computer scientist and software engineering pioneer, seems to have originated the phrase “separation of concerns,” as it has come to be used in software engineering, in 1974:¹

Let me try to explain to you, what to my taste is characteristic for all intelligent thinking. It is, that one is willing to study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects. We know that a program must be correct and we can study it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day, so to speak. In another mood we may ask ourselves whether, and if so: why, the program is desirable. But nothing is gained—on the contrary!—by tackling these various aspects simultaneously. It is what I sometimes have called “the separation of concerns”, which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by “focusing one's attention upon some aspect”: it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant. It is being one- and multiple-track minded simultaneously.

Programming techniques such as David Parnas' information hiding² can be understood as an application of the principle of separation of concerns to guide the modularization of programs. Information hiding, a foundation of both structured design and object-oriented design, advocates the idea of secrets, by which design and implementation decisions are hidden in one module from the rest of a program. Since then, separation of concerns has been a dominant idea in software engineering, embodied in most, if not all, methods and processes.

In formulating aspect-oriented programming, Gregor

Kiczales and colleagues at Xerox PARC observed that many key properties of programs weren't well served by top-down, structured programming, or object-oriented paradigms.³ This work arose out of studies of open implementation—making explicit key parameters of programs, systems, and application frameworks so that programmers could adjust them in a safe, systematic fashion. Aspect-oriented programming was based on the idea that while functional concerns were well served by existing modularization techniques (whether procedural or object-oriented), other concerns such as memory allocation, synchronization, persistence management, and failure handling “cross-cut” these functional modules. Kiczales and colleagues called these *aspects*, and developed ways to specify and implement concerns by weaving them together with functional modules. “Aspects tend not to be units of the system's functional decomposition, but rather to be properties that affect the performance or semantics of the components in systemic ways.”³ In the present context, we might say an aspect names and then implements a cross-cutting programming concern.

Aspect-oriented programming has become a valuable metaphor outside of programming as well, with work in aspect-oriented design, requirements, and even aspects in software architecture. This work has been useful in getting software engineers, designers, and architects to consider the importance of stakeholder concerns as first-class entities throughout the system life cycle.⁴

References

1. E.W. Dijkstra, “On the Role of Scientific Thought,” Springer-Verlag, 1974; www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html.
2. D. Parnas, “On the Criteria to Be Used in Decomposing Systems into Modules,” *Comm. ACM*, vol. 15, no. 12, 1972, pp. 1053–1058.
3. G. Kiczales et al., “Aspect-Oriented Programming,” *Proc. European Conf. Object-Oriented Programming (ECOOP)*, LNCS 1241, Springer-Verlag, 1997, pp. 220–242.
4. S.M. Sutton Jr. and I. Rouvellou, “Concern Modeling for Aspect-Oriented Software Development,” *Aspect-Oriented Software Development*, R.E. Filman et al., eds. Addison-Wesley, 2004, pp. 479–505.

architectures. Each view is chosen to frame certain stakeholder concerns and depict how the system will address those concerns. The use of multiple views in architecture has been popularized in Philippe Kruchten's 4+1 model and many others, and is codified in IEEE Standard 1471-2000; Table 1 lists a few such approaches. Architecture frameworks such as ISO Reference Model of Open Distributed Processing (RM-ODP), the Open Group Architecture Framework (TOGAF), the US Department of Defense Architecture Framework (DODAF), and the UK Ministry of Defence Architecture Framework (MODAF) also frequently organize their guidance into multiple

viewpoints (although not always using that term) to frame specific stakeholder concerns.

Deconstructing “Nonfunctional Requirements”

Functionality—what a system does—is perhaps the best-understood concern. What capabilities must the system provide? What services does the system perform? Use cases and user stories are about delivering functionality to a system's users. However, architects have realized that functional requirements are often not the hard part; the hard parts are usually everything else and frequently lumped together as “nonfunctional requirements” (NFRs).

accuracy, adaptability, agility, assurance, autonomy, behavior, completeness, complexity, compliance to regulation, comprehensibility, concurrency, configurability, control, cost of ownership, customer experience, data accessibility, deadlock, development cost, development time, distribution, customer loyalty, error handling, evolvability, feasibility, flexibility, functionality, information assurance, inter-process communication, learnability, maintainability, maintenance cost, modifiability, modularity, openness, operating cost, performance, persistence, privacy, project stability, quality of service, reliability, resource utilization, return on investment, reusability, security, state change, structure, subsystem integration, supportability, system features, training time, usage

Figure 1. Architecturally relevant stakeholder concerns. This list is adapted from ISO/IEC FCD 42010.

The term is troublesome in two ways: first, “nonfunctional” bundles together many things that are otherwise unrelated to one another. The term implies, mistakenly, commonality to “everything else”; that the architect simply has to spend some time figuring out the NFRs for a system once (typically after) its functionality is under control. However, there’s no reason to assume that such diverse concerns as design-time modifiability, runtime performance, product time-to-market, and architectural consistency are all amenable to the same treatment. Second, discussions of so-called NFRs often involve much more than stated requirements: stakeholders have preferences, goals, and needs the architect must discover. For example, consider latency—in a real-time audio processor for musicians in live performance situations, reducing latency to milliseconds will be a requirement, whereas latency in an interplanetary communications system is a design constraint likely to be on the order of minutes or longer. In addition, there has recently been a shift from viewing architecture as only structure to a broader view of architectural knowledge that emphasizes the treatment of architectural design decisions as first-class entities.¹ From this perspective, there’s no fundamental distinction between architectural decisions and architecturally significant requirements: they only differ by the moment in time they’re identified, discussed, and taken. Architects are actively engaged in the delineation and negotiation of these issues with the client, leading to possible requirements and possible designs, while still making trade-offs between conflicting desires and goals of diverse stakeholders.

For these reasons, we prefer the term “stakeholder concerns” to NFRs.

By distinguishing and separating concerns, we’re better able to give the architect tools for understanding and managing those critical issues effectively throughout the life cycle. Languages such as UML provide notations for expressing functionality and structure at varying levels of detail, but for other types of concerns, other languages may be needed (see the Point/Counterpoint on page 54). Looking again at Figure 1, how many of these stakeholder concerns can you associate with a standard notation or common modeling approach? Alas, many of these stakeholder concerns present architects with the greatest challenges, uncertainties, and risks, often determining project success or failure. It’s for this reason that NFRs have attracted so much interest.

Stakeholder concerns fall into several cate-

gories: beyond functionality—what the system is to do—architects must determine how it will do it. Functionality constraints are often called *system qualities*, or quality attributes, and can be further categorized, such as by when they come into play, at design-time or runtime. Beyond the “what” and “how” of the system itself, the architect is often concerned with development consequences, such as, Can we build this? Will this development process allow us to deliver the product to market ahead of competitors? Are there adequate resources available to build, own, and manage the system as conceived? These meta-systemic concerns even dominate in many cases, as might how the system will be operated, certified, and even retired.

The architect uses concerns to shape the problems to be solved, gather relevant requirements, and design constraints, but still needs to solve those problems. Rarely is this a single step (for example, from recognizing that reliability is a concern to incorporating automated backup into the solution)—usually some modeling and analysis is involved. Models are devised, considered, shared, and then presented to the client and relevant stakeholders. In line with the principle of separation of concerns, different models help architects tackle complexity by dealing with a subset of concerns at one time and organizing those models into multiple views. Following the terminology of ISO/IEC 42010, the conventions for each view define an *architecture viewpoint*. A viewpoint determines what types of models, notations, and tools can be used for a given set of concerns and stakeholder, together with any associated operations, guidance, and heuristics to aid the architect. Building on the viewpoint idea, an architecture framework is a coordinated set of viewpoints—prescribed modeling resources for a particular community or application domain’s stakeholder concerns. Similarly, an architecture description language (ADL) provides resources for framing some set of concerns via one or more notations and often automated tools.

In This issue

As noted earlier, some stakeholder concerns are well-served today by available architecture viewpoints, frameworks, or ADLs, while others aren’t expressible with current, off-the-shelf approaches. Hence the theme of this special issue: exploring the space of architecting in the face of multiple stakeholder concerns and looking for solutions that help the architect in that space.

The articles in this issue all demonstrate techniques for framing one or more stakeholder con-

Table 1**Viewpoint-based approaches**

| Approach | Viewpoints | Notes |
|-------------------------------|--|--|
| 4+1 View Model | Logical, development, process, and physical, “plus” scenarios | P.B. Kruchten, “The ‘4+1’ View Model of Architecture,” <i>IEEE Software</i> , vol. 28, no. 11, 1995, pp. 42–50 |
| ViewPoints | Stakeholder-centered; proposes a valuable scheme for specifying viewpoints | B. Nuseibeh, J. Kramer, and A. Finkelstein, “A Framework for Expressing the Relationships between Multiple Views in Requirements Specification,” <i>IEEE Trans. Software Eng.</i> , vol. 20, no. 10, 1994, pp. 760–773 |
| Siemens 4 Views | Conceptual, module, execution, and code | C. Hofmeister, R. Nord, and D. Soni, <i>Applied Software Architecture</i> , Addison-Wesley, 1999 |
| SEI Views & Beyond | Component and connector, module, allocation; viewtypes can be used to construct viewpoints | P. Clements et al., <i>Documenting Software Architectures: Views and Beyond</i> , Addison-Wesley, 2010 |
| Software Systems Architecture | Functional, information, concurrency, development, deployment, operational; introduces perspectives that address concerns cross-cutting other viewpoints | N. Rozanski and E. Woods, <i>Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives</i> , Addison-Wesley, 2005 |
| IEEE Std 1471-2000 | No predefined viewpoints; establishes an ontology and generic mechanism for defining particular viewpoints in a uniform manner | Now also ISO/IEC 42010:2010, Systems and Software Engineering—Architecture Description |

cerns. Some have confronted the topic within software architecture; others are included because they offer insights on concerns and viewpoints from other branches of software engineering.

In “Requirements-Driven Design of Service-Oriented Interactions,” Ayman Mahfouz, Leonor Barroca, Robin Laney, and Bashar Nuseibeh address a hard problem in the area of service-oriented architectures—that of architecting interactions with multiple stakeholders (users). This article demonstrates a fundamental premise of this special issue, of using multiple viewpoints to address a particular class of concerns within a specific application domain.

Juha Savolainen and Tomi Männistö in “Conflict-Centric Software Architectural Views: Exposing Trade-Offs in Quality Requirements” argue that many architectural decisions are driven by conflicts—between requirements, between stakeholders, between implementation technologies—and sketch a means for modeling conflicts in a viewpoint-based approach.

In “The Business Goals Viewpoint,” Paul Clements and Len Bass, building on the Software Engineering Institute’s work on quality attribute scenarios, apply the ISO/IEC 42010 viewpoint template to present a viewpoint (notation, association methods) for eliciting and modeling stakeholders’ business-related goals and constraints, which often fall very far outside the familiar categories of functional and quality concerns yet ex-

ert considerable influence over most architectures.

A key tenet of managing diverse stakeholder concerns through multiple viewpoints is matching suitable presentations of results to each stakeholder. Alexandru C. Telea, Lucian Voinea, and Hans Sassenburg, in “Visual Tools for Software Architecture Understanding: A Stakeholder Perspective,” survey the technologies for visualization of a variety of system concerns.

Finally, in this month’s Point/Counterpoint, Eoin Woods, David Emery, and Bran Selic pick up on the theme of the special issue to debate the efficacy of UML for the wide range of stakeholder concerns that the architect must confront.

By paying attention to stakeholders’ concerns and associating modeling techniques with those concerns, the hope is that architects can tackle diverse architectural challenges as systematically as functionality is handled today. The paradigm for functionality is one of “stepwise refinement”: functionality is specified (whether through user stories or formal requirements), elaborated upon (whether on paper or through models and prototypes), designed and implemented, verified, and validated. This approach—which we take for granted in software engineering—is much more effective than waiting until delivery to check whether the system provides all hoped-for services. In con-

About the Authors




Patricia Lago is associate professor at the VU University Amsterdam. Her research interests are in software- and service-oriented architecture, architectural knowledge management, and green IT. Lago has a PhD in control and computer engineering from Politecnico di Torino. She's a member of the IFIP Working Group 2.10 on software architecture, IEEE, and the ACM. Contact her at patricia@cs.vu.nl.

Paris Avgeriou is a professor of software engineering at the University of Groningen, the Netherlands. He has received awards and distinctions for both teaching and research and has published more than 80 articles in peer-reviewed international journals, conference proceedings, and books. His research interests concern the area of software architecture, with a strong emphasis on architecture modeling, knowledge, evolution, and patterns. Contact him at paris@cs.rug.nl.



Rich Hilliard is a freelance software architect and software engineer. He's also editor of ISO/IEC 42010, Systems and Software Engineering—Architecture Description (the internationalization of the widely used IEEE Std 1471:2000). Hilliard is a member of the IFIP Working Group 2.10 on software architecture, the IEEE Computer Society, and the Free Software Foundation, and is an officer of the League for Programming Freedom. Contact him at r.hilliard@computer.org.

trast, less familiar stakeholder concerns, when articulated at all, don't follow this path. More often, what happens is that an architectural solution is devised in terms of familiar views, and the solution is then analyzed for its impact on other concerns after the fact. Although this is preferable to missing the concern completely, can we do better? Can we manage critical stakeholder concerns throughout the process, the way we have come to manage functionality? That is our hope for viewpoints. If so, there's an additional benefit: viewpoints (and frameworks and ADLs) are applicable not just once but have potential as reusable assets applied to many systems. To explore such reuse, we and our colleagues are working to create a viewpoint repository at www.iso-architecture.org/viewpoints, in which we hope the community will participate. 

Reference

1. P. Kruchten, "An Ontology of Architectural Design Decisions in Software Intensive Systems," *2nd Groningen Workshop on Software Variability*, Dec. 2004, pp. 54–61.

CALL FOR ARTICLES

The Software Business

PUBLICATION: JULY/AUGUST 2011

SUBMISSION DEADLINE: 1 DECEMBER 2010

This special issue on the software business will focus on two questions:

- How do software creation and support organizations address an enterprise's existing business model? That is, how does a successful enterprise embrace software products and services to preserve or improve its competitiveness in the marketplace?
- How does an enterprise whose primary focus is producing software (components, applications, services) find a successful business model for competing in the marketplace?

The answers to these questions inform any analysis of how, when, why, and whether software should be incorporated in an enterprise's products, processes, and services. Any engineered product, including software, should be viewed from a business perspective, not just from a technological one.

This special issue will examine established and emerging business models for building, selling, and incorporating software into systems that have been successful in the marketplace. We also welcome articles about lessons learned from failed business models. All segments of the software business (services, products, consulting) are considered in scope. Contributions from commercial, academic, and public entities are welcome.

POSSIBLE TOPICS

- Proven and emerging business models: software as a product, software as a service, mobile applications ecosystems, open source
- The intersection of software engineering with the software business, and business perspectives on technologies
- Outsourcing and offshoring: long-term economic implications, new models, strategic tradeoffs
- Innovation management: approaches and initiatives for fostering creativity and product differentiation

QUESTIONS?

For more information about the focus, contact the guest editors:

- John Favaro, INTECS, jfavar@gmail.com
- Shari Lawrence Pfleeger, Dartmouth College, Shari.L.Pfleeger@dartmouth.edu

For the full call for papers: www.computer.org/software/cfp4

For general author guidelines:
www.computer.org/software/author.htm

For submission details: <https://mc.manuscriptcentral.com/sw-cs>

**IEEE
Software**