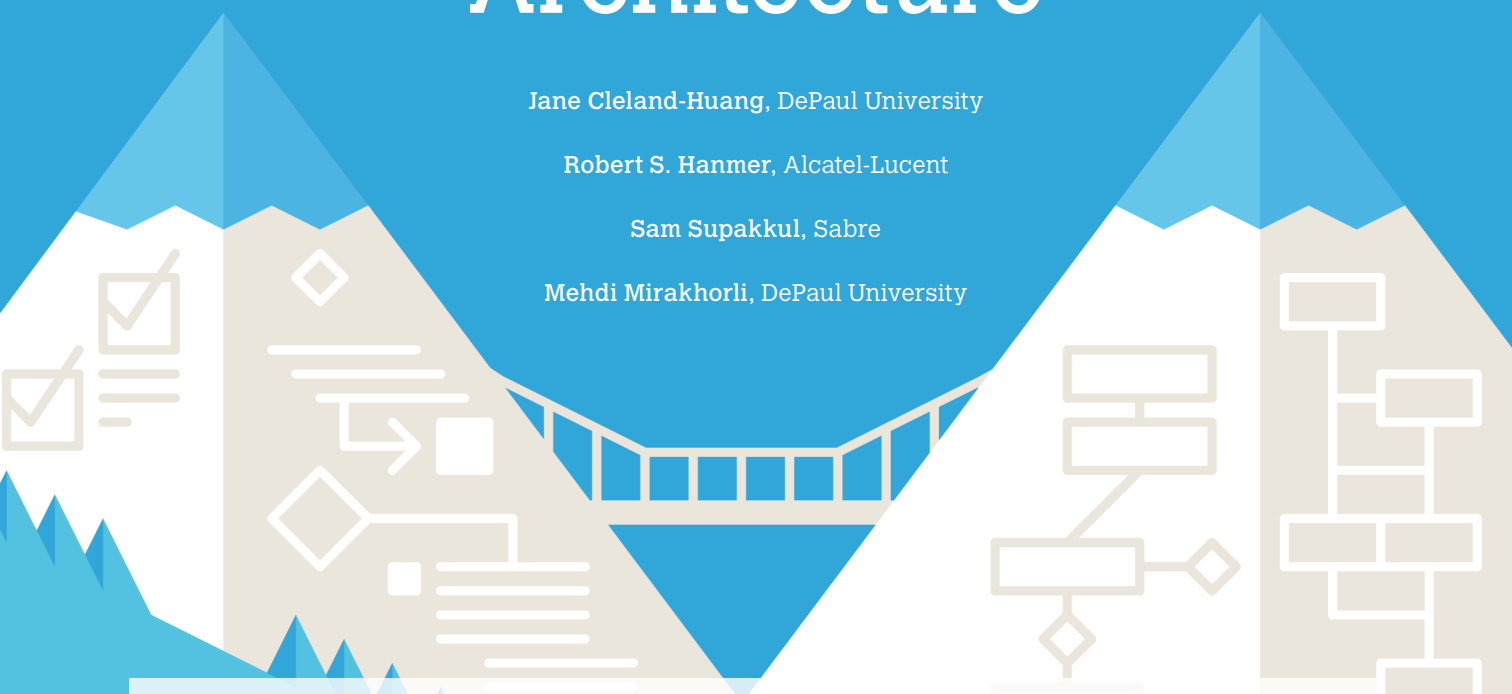
The Twin Peaks of Requirements and Architecture

Jane Cleland-Huang, DePaul University

Robert S. Hanmer, Alcatel-Lucent

Sam Supakkul, Sabre

Mehdi Mirakhorli, DePaul University



QUALITY CONCERNS, often referred to as nonfunctional requirements, service-level agreements, quality attributes, performance constraints, or architecturally significant requirements, describe system-level attributes such as security, performance, reliability, and maintainability. In conjunction with functional requirements, these quality concerns drive and constrain a system's architectural design and often introduce significant trade-offs that must be carefully considered and balanced.[1] The dependencies that exist between requirements and architecture have been referred to as the twin peaks of requirements and architecture.[2]

This model, which is described more fully in the sidebar "The Twin Peaks Model," written by Bashar Nuseibeh, emphasizes the iterative nature in which requirements are elicited and specified and in which the architectural design is established. In their recent book *Relating Software Requirements and Architectures*, Paris Avgeriou and colleagues refer to the two islands of software architecture and requirements engineering, creating an alternate metaphor for the same basic observation.[3]

Dependencies

To build successful and cost-effective software systems, we must understand and leverage the dependencies between requirements and architecture. However, the way in which we accomplish this is clearly influenced by the philosophy and practices of the underlying development process.

In traditional projects, which incorporate more rigorous up-front requirements and design processes, there's a danger of focusing on functionality and ignoring quality concerns. When this happens, developers might deliver solutions

0740-7459/13/$31.00 © 2013 IEEE

that fail to live up to unspoken quality expectations. It's therefore important to proactively elicit quality requirements from project stakeholders during early phases of the project and then design architectural solutions that balance and satisfy those concerns.[4] Early prototypes and architectural evaluations are needed to demonstrate that the delivered system is able to meet its quality goals.

On the other hand, agile and lean projects implicitly rely on short iterations and early delivery of executable code into customer hands. Architectural design emerges incrementally in response to customer needs. Although agile processes bring numerous benefits to a project, the somewhat shorter-term perspective means that developers could be forced into expensive refactoring efforts to deliver new functionality late in the project. Agile processes that elicit architecturally significant user stories in early iterations can balance the way in which functionality is delivered to the customer and enable developers to make informed decisions about the design and construction of the architecture.

It's clear that the fundamental dependencies highlighted by the twin peaks model influence and constrain software development and maintenance efforts, regardless of the development process adopted. These dependencies influence not only the way new systems are built but also the way in which change requests are handled and new functionality is introduced into an existing system. Once the software is constructed and deployed, previous architectural decisions constrain new requirements, and their feasibility and impact must be evaluated in light of the current system. The problem can be exacerbated by ongoing maintenance efforts that erode the quality of previous architectural decisions, making relationships between requirements and architecture less obvious and more difficult to establish and maintain. This is especially true as the software ages after multiple development iterations.[5,6]

Furthermore, at the organization level, the development department in some organizations might have technology standards or an approved technology portfolio in place to control development costs, resource skills, and operation support. However, these standards might constrain the viability of introducing new product features that the marketing department wants. Consequently, this could require negotiation between the two departments to generate new product features or to amend the technology standards to accommodate market pressures, thus leading to the need to manage the resulting architectural impacts.

The Need for Advances

Traditionally, requirements engineering has emphasized early phases of software development, whereas software architecture practices have primarily focused on downstream architectural concerns or architectural analysis techniques. Well-established requirements and architecture conferences, such as the International Requirements Engineering Conference (RE), Requirements Engineering: Foundation for Software Quality (REFSQ), Working Conference on Software Architecture (WICSA), and SEI's Software Architecture Technology User Network (SATURN), have helped advance the state of the art of their respective focuses, but little emphasis has been placed on bridging the gap between these two domains. The primary exception is the From SofTware Requirements to Architectures Workshop (STRAW), which was held at the International Conference on Software Engineering in 2001 and 2002. As implied by its title, STRAW emphasized the evolution from requirements to architecture, as opposed to the interplay between the two.

To encourage advances in bridging the two domains, a workshop entitled Twin Peaks of Requirements and Architecture was held at the IEEE Conference on Requirements Engineering in September 2012 as a forum for people from both the requirements and software architecture communities to collaborate and exchange ideas. The workshop was attended by approximately 15 practitioners from industries including telecommunications, nuclear power, and electronics, as well as 20 academic researchers. The workshop concluded with a brainstorming session during which attendees identified ongoing challenges that should be addressed to bridge the gap between requirements and architecture. The workshop's program chairs, Janet Burge, Mehdi Mirakhorli, and Roshanak Roshandel, present a summary in the sidebar entitled "Climbing the Twin Peaks: Open Challenges."

The fundamental dependencies highlighted by the twin peaks model influence and constrain software development.

In This Issue

This special issue of *IEEE Software* includes a selection of articles that

THE TWIN PEAKS MODEL

The importance of interleaving the tasks of eliciting and specifying requirements with that of designing a software solution has long been recognized.[1,2] The traditional waterfall process produces artificially frozen requirements that often lead to suboptimal architectural solutions, which themselves are often inflexible to future change. Incremental development processes partially address this problem by allowing developers to evaluate repeatedly changing project risks in order to embrace new requirements and to address changing project constraints. The twin peaks model[3] provides an even finer-grained iteration across requirements and architecture that acknowledges the need to develop software architectures that are stable, yet adaptable, in the presence of changing requirements.

As shown in Figure A, the twin peaks model focuses on the co-development of requirements and architecture. Through a series of iterations, the model captures the progression from general to detailed understanding and expression of both requirements and design. Although the schematic of the twin peaks model shows the process initiated on the requirements peak, projects involving modifications to existing systems could be initiated equally well at the architecture peak.

Although the model shown in Figure A has become broadly recognized as the twin peaks model, several alternatives were initially explored including "design alternatives" and "mountain range." Figure B, the design alternatives model, presents a recognized yet frequently neglected activity in software development: that of generating and evaluating candidate architectures with respect to some (fixed) quality requirements. Extending this, the mountain range of Figure C posits that the requirements themselves aren't set in stone, so alternative requirements might need to be explored if particular architectural choices are to be made.

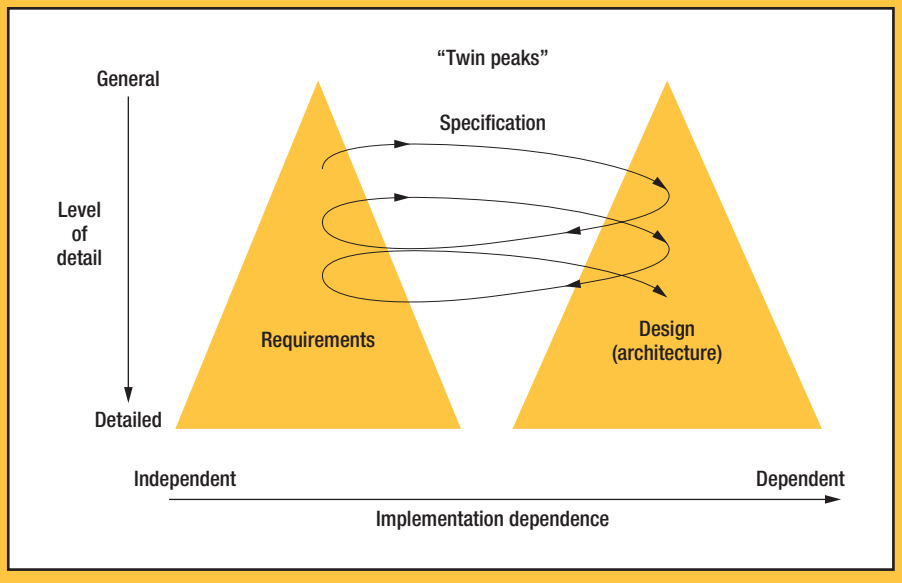


FIGURE A. The twin peaks model. Though a series of iterations, the model captures the progression from general to detailed understanding.

The twin peaks model is arguably little more than a simple process diagram, but it has served as an appealing metaphor for drawing attention to the synergistic relationships between two fundamental software development artifacts: requirements and architecture. For many years, this model remained an aspirational view of software development. However, the recog-

explore issues at the intersection of requirements and architecture.

An interview with Dan Dvorak, principal engineer at the Jet Propulsion Laboratory and lead of NASA's software architecture review board, and Jan Bosch, professor of software engineering at Chalmers University Technology in Gothenburg (and previously vice president of engineering process at Intuit), highlights their perspectives on the interplay of requirements and architecture in two very different kinds of project environments.

In "Characterizing Architecturally Significant Requirements," Lianping Chen, Muhammad Ali Babar, and Bashar Nuseibeh present a classification of characteristics of architecturally significant requirements derived from an analysis of observations and experiences of 90 software professionals using grounded theory. These characteristics describe and expand on the notion of architecturally significant requirements, which could help make it easier for practitioners to recognize and capture these requirements.

To realize architecturally significant requirements in a system or its subcomponents, the architect might follow an architectural design method for carrying out

nition that software processes—agile, lean, or otherwise—are essentially engineering processes means that the twin peaks model captures more than just an iterative development activity but the most fundamental of all engineering relationships, those between software development problems and their solutions.[4]

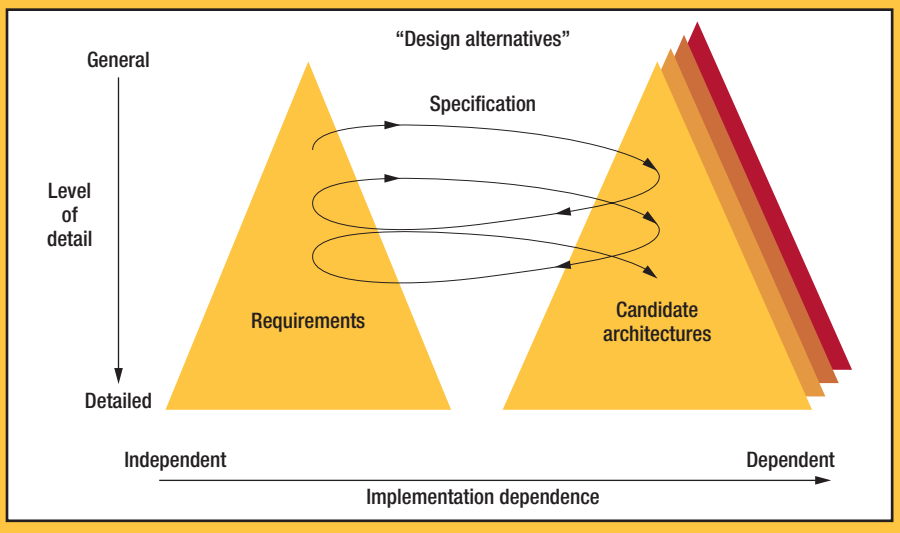


FIGURE B. Design alternatives model. One frequently neglected activity in software development is the generation and evaluation of different candidate architectures.

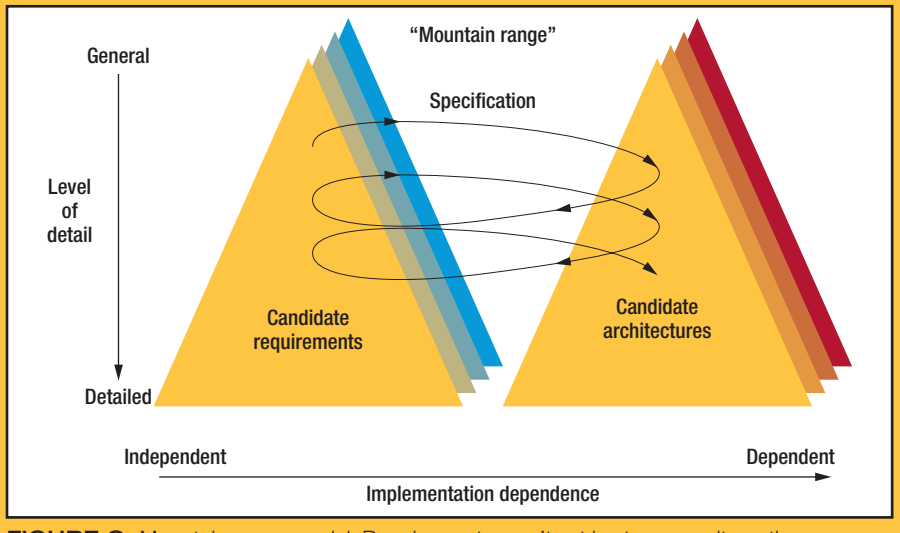


FIGURE C. Mountain range model. Requirements aren't set in stone, so alternative requirements might need to be explored.

References

1. W. Swartout and R. Balzer, "On the Inevitable Intertwining of Specification and Implementation," *Comm. ACM*, vol. 25, no. 7, 1982, pp. 438–440.
2. P. Ward and S. Mellor, *Structured Development for Real-Time Systems: Introduction and Tools*, vol. 1, Prentice Hall, 1985.
3. B. Nuseibeh, "Weaving Together Requirements and Architectures," *Computer*, vol. 34, no. 3, 2001, pp. 115–119.
4. P. Avgeriou et al., *Relating Software Requirements and Architectures*, Springer, 2011.

BASHAR NUSEIBEH is a professor at the Open University UK and Lero—the Irish Software Engineering Center. Contact him at b.nuseibeh@open.ac.uk.

such activity. Several methods have been proposed and used in practice,[3] among which is the attribute-driven design (ADD) method.[4] In "A Principled Way to Use Frameworks in Architecture Design," Humberto Cervantes and colleagues argue that following the ADD method might not be straightforward when dealing with existing frameworks or third-party technologies. To accommodate these architectural constraints, the authors illustrate an extended ADD approach that iteratively considers both functional and nonfunctional requirements as well as third-party frameworks to complement with the traditional tactic- and pattern-based architectural decisions.

Along the twin peaks model, "Your 'What' is My 'How': Iteration and Hierarchy in System Design," by Michael W. Whalen and colleagues proposes to align requirements organization and architectural design based on the architectural decomposition in a hierarchical fashion. Using this approach, requirements help determine subcomponents, verify that subcomponents establish the system, and verify that the requirements themselves are allocated to subcomponents. The article also discusses

CLIMBING THE TWIN PEAKS: OPEN CHALLENGES

While researchers in requirements and architecture communities have independently tackled some of the challenges associated with interdependencies between requirements and architecture, they have rarely gathered to form a unified community that addresses emerging challenges. The First Workshop on the Twin Peaks of Requirements and Architecture in September 2012, hosted by the IEEE International Requirements Engineering Conference, drew more than 35 researchers and practitioners of software architecture and software requirements together to discuss ideas, present state-of-the-art solutions for more effectively interweaving requirements and architecture, and chart a road map for a future research agenda and best practices in this area.

Several clear themes emerged from the workshop that lay the foundation for future research efforts by both academicians and practitioners. The first theme centered on the role that architecturally significant requirements play in shaping the architectural design of a system and constraining the set of viable architectural options. A second complementary concept that emerged was the way in which an existing architecture constrains the financial viability of implementing new feature requests. Dependencies between requirements and architecture therefore come into play in both directions.

Workshop attendees identified the following open challenges:

- *Communication.* Many projects have almost no in-depth communication between requirements analysts and software architects. We need a better understanding of the kinds of information that should be communicated in different types of project environments. Moreover, many current modeling notations are designed either for requirements analysis or for architectural design.
- *Preserving architectural knowledge.* In practice, architectural knowledge is often lost, leading to situations in which the underlying architectural design is eroded during long-term maintenance and evolutionary efforts. Critical architectural knowledge must be seamlessly maintained and preserved not only for development, refactoring, and maintaining systems but also for evolving systems to address new and emerging requirements. To different degrees of granularity and specificity, developers, testers, project managers, business analysts, and requirement engineers need to understand the underlying architectural decisions and rationales to ensure financial and technical feasibility of new requirements without degrading system quality.
- *Reconstructing requirements knowledge.* A large number of currently deployed systems lack documented and up-to-date requirements. This is a critical problem, especially as many of

a virtual integration method to formally verify the properties and contracts between components to detect and prevent integration errors.

Finally, in "Non-functional Requirements in Architectural Decision Making," David Ameller and colleagues explore the real-world pressures faced by software architects in projects. They present results from a series of 13 interviews and discuss the ways in which architects addressed quality concerns from an engineering perspective and then explore the impact of those concerns on decision making. Among other things, the authors' findings confirm the iterative nature in which quality requirements are elicited but also make new observations related to current practices in the validation and measurability of quality requirements.

It's our hope that this special issue will challenge readers to think more about the dependencies between requirements and architecture in their own projects and to consider adopting practices that embrace and even leverage those relationships.

References

1. L. Chung and J. do Prado Leite, "On Non-Functional Requirements in Software Engineering," *Conceptual Modeling: Foundations and Applications*, Springer-Verlag, 2009, pp. 363–379; http://dx.doi.org/10.1007/978-3-642-02463-4_19.
2. B. Nuseibeh, "Weaving Together Requirements and Architectures," *Computer*, vol. 34, no. 3, 2001, pp. 115–119.
3. P. Avgeriou et al., *Relating Software Requirements and Architectures*, Springer, 2011.
4. R. Wojcik et al., "Attribute-Driven Design (ADD)," tech. report CMU/SEI-2006-TR023, Software Eng. Inst., Nov. 2006.
5. D.E. Perry and A.L. Wolf, "Foundations for the Study of Software Architecture," *ACM SIGSOFT Software Eng. Notes*, vol. 17, 1992, pp. 40–52.
6. B. Boehm, "Get Ready for Agile Methods, with Care," *Computer*, vol. 35, no. 1, 2002, pp. 64–69.

See www.computer.org/software-multimedia for multimedia content related to this article.

these projects no longer have access to the original developers. This need is less apparent when systems are continually changed and modified, or when project knowledge is passed verbally from one developer to the next, but this is not always the case.

- *Architectural visualization.* Decomposition of a software system's architecture into viewpoints, perspectives, layers, components, or slices is critical in first understanding and then communicating the architecture to a variety of project stakeholders. Such visualizations should depict the system's friction points and reveal its underlying patterns in order to visualize how changes in one part of the system impact other parts.
- *Evolution in agile environments.* In agile development environments, software architecture emerges as a result of iterative requirements and design processes. The agile philosophy to embrace change means that new requirements are continually introduced, and developers evaluate the feasibility and cost of the requirement with respect to the existing architecture. We need to gain a better understanding of the impact of architectural decisions on future change and specifically explore the idea of architecture breakers: requirements that the current architecture can't cost-effectively accommodate.
- *Tracing requirements to architecture.* Traceability is the ability to establish and understand relationships between different artifacts. In this context, understanding the relationships between requirements and architectural artifacts can better support tasks such as change impact analysis and feature location. This is particularly crucial in certain domains such as safety and/or mission critical systems. Explicit traceability links (matrices) for relating key architectural decisions to rationales, however, often clash with the inclination to avoid introducing additional forms of documentation in the agile methodology.
- *Training*. Software engineering education often doesn't directly address the interdependency between requirements and architectures. Due to the nature of courses, students often work on one activity at a time.

The second twin peaks workshop will be held at the International Conference on Software Engineering on 21 May 2013 in San Francisco.

JANET BURGE, MEHDI MIRAKHORLI, AND ROSHANAK ROSHANDEL were the program chairs for the First Workshop on the Twin Peaks of Requirements and Architecture. Contact them at burgeje@muohio.edu, mirakorli@gmail.com, and roshanak@seattleu.edu.

ABOUT THE AUTHORS

JANE CLELAND-HUANG is an associate professor in the School of Computing at DePaul University, Chicago, where she serves as the director of the Systems and Requirements Engineering Center. Her research interests focus primarily on requirements and architectural traceability. Cleland-Huang received a PhD in computer science from the University of Illinois at Chicago. She serves on the editorial boards for *IEEE Software*, the *Requirements Engineering Journal*, and *IEEE Transactions on Software Engineering*. Cleland-Huang is a member of the IEEE Computer Society and the IEEE Women in Engineering. Contact her at jhuang@cs.depaul.edu.

SAM SUPAKKUL is a Principal Architect in the Travel Network unit at Sabre, where he's leading an effort to improve the agile development process used by all projects in the business unit to better address nonfunctional requirements throughout the development life cycle. His practical and research interests include requirements engineering, nonfunctional requirements, system and software architecture, and pattern-based software development. Supakkul received a PhD in software engineering from the University of Texas at Dallas. He is a senior member of IEEE.

ROBERT S. HANMER is a Consulting Member of Technical Staff with Alcatel-Lucent in Naperville, Illinois. He's active in the software patterns community, including serving as program chair for several pattern conferences, and he has authored multiple articles and books on the subject. Hanmer received an MS in computer science from Northwestern University. He is a member of the IEEE Computer Society, a Senior Member of the ACM, and past-president of the Hillside Group, the organization that sponsors the PLoP conference.

MEHDI MIRAKHORLI is a doctoral student at DePaul University. His research interests include software architecture design, development and maintenance, and also techniques to improve software development, with a focus on requirements engineering and software architecture practices. Previously, he served as an adjunct lecturer at the Iran University of Science and Technology and worked for seven years as a software architect/designer on large, data-intensive, meteorological and healthcare systems.