

Agility and Architecture: Can They Coexist?

Pekka Abrahamsson, *University of Helsinki*

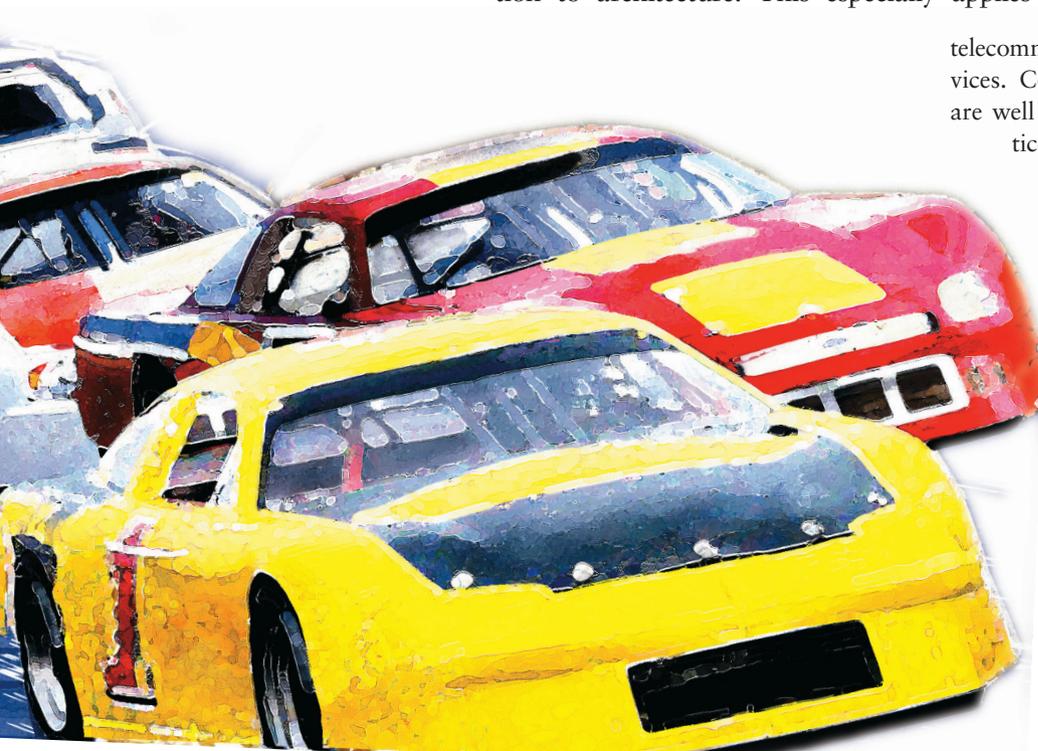
Muhammad Ali Babar, *IT University of Copenhagen*

Philippe Kruchten, *University of British Columbia*

Agile development has significantly impacted industrial software development practices. However, despite its wide popularity, there's an increasing perplexity about software architecture's role and importance in agile approaches. Advocates of architecture's vital role in achieving quality goals for large software-intensive systems doubt the scalability of any development approach that doesn't pay sufficient attention to architecture. This especially applies to domains such as automobiles,

telecommunications, finance, and medical devices. Companies where architectural practices are well developed often tend to see agile practices as amateurish, unproven, and limited to very small Web-based sociotechnical systems.¹

Conversely, proponents of agile approaches usually see little value for a system's customers in the upfront design and evaluation of architecture. They perceive software architecture as something from the past, equating it with big design up-front (BDUF)—a bad thing—leading to massive documentation and implementation of YAGNI (you ain't gonna need it) features. They believe that architectural design has little value, that a metaphor should suffice in most cases,² and that the architecture should emerge gradually sprint after sprint, as a result of successive small refactoring.



Interest is growing in separating the facts from myths about the necessity, importance, advantages, and disadvantages of having agile and architectural approaches coexist, and that's the theme of this special issue.

Any debate, discussion, or effort to assess the necessity of combining agile and architecture should start with questions such as: Are these views contradictory, opposing, or complementary? Do the proclaimed dichotomies between agile and architecture have any truth? What steps will let project teams benefit from the best of both by ignoring unnecessary values or requirements?

Paradox, Oxymoron, Incompatibility?

Jim Highsmith defines agility as “the ability of an organization to both create and respond to change in order to profit in a turbulent business environment.”³ Sanjiv Augustine notes that agile development methods such as Extreme Programming (XP), scrum, feature-driven development, lean, Crystal, and so on have common characteristics, such as

- iterative and incremental life cycles,
- focus on small releases,
- collocated teams, and
- a planning strategy based on a release plan driven by a feature or product backlog and an iteration plan handling a task backlog.⁴

They all also more or less adhere to the values of the Agile Manifesto (www.agilemanifesto.org).

The Rational Unified Process (RUP) defines software architecture as the

*set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements, the composition of these elements into progressively larger subsystems, the architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition. Software architecture is concerned with not only structure and behavior but also usage, functionality, performance, resilience, reuse, comprehensibility, economic and technological constraints and trade-offs, and aesthetics.*⁵

The tension seems to lie on the axis of adaptation versus anticipation. Agile methods want to be resolutely adaptive: deciding at the “last responsible moment” or when changes occur. Agile methods perceive software architecture as pushing too hard on the anticipation side: planning too much in advance. Perhaps we can find a balance between these two extreme approaches and mind-sets.

False Dichotomies?

When discussing the direction of this special issue, Craig Larman asserted that this tension between agility and architecture might be a false dichotomy. Indeed, there are plenty of such artificial splits. Some are inadvertent; some are intentional, to prop up a certain message: agile versus waterfall or agile versus disciplined. Like many others in software development research and practice, we strongly believe that a healthy focus on architecture isn't antithetic to any agile process. The tenors of various agile methods also seem to agree. Along these lines, Satoshi Basaki noted, “It seems that many agile method users misunderstand what agile methods are, just ignore architecture, and jump onto refactoring” as the one and only panacea.

There's also the drive to “deliver value to the stakeholders” right from the first sprint or iteration. But what if the developers, not just the end users, are a key class of stakeholders? Alistair Cockburn developed strategies for starting with a walking skeleton, then evolving it iteratively.⁶ Mary and Tom Poppendieck came up with the notion of “divisible system architecture.”⁷ And finally, Kent Beck's advises that “architecture is just as important in XP projects as it is in any software project. Part of the architecture is captured by the system metaphor [one of the XP practices].”²

So what issues must we address to begin reconciliation?

Discovering the Real Issues

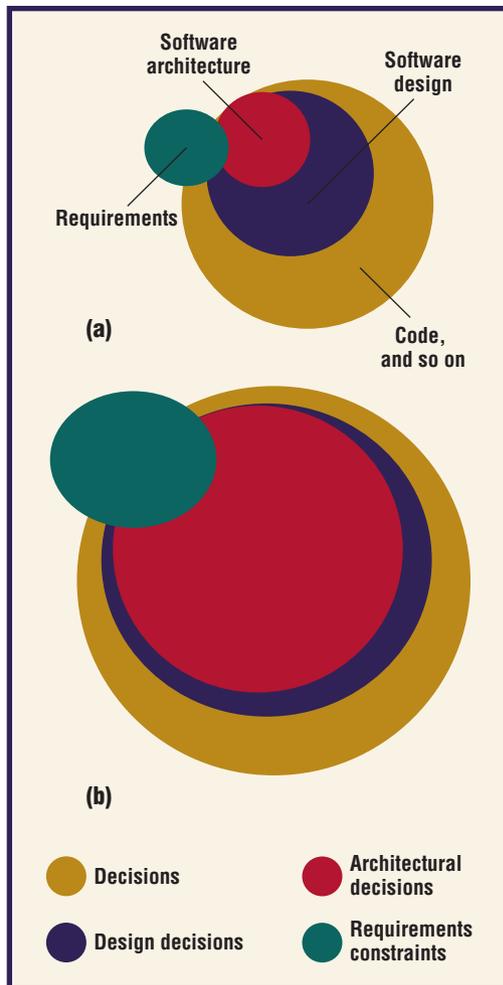
There are multiple levels to understanding the apparent conflict between agile development and architecture: semantics, scope, life cycle, role, documentation, methods, value, and cost.

Clarifying the Semantics

What does a particular project or organization mean by architecture? The concept often has fuzzy boundaries. In particular, not all design is architecture. Agreeing on a definition is a useful exercise and a good starting point.

This tension between agility and architecture might be a false dichotomy.

Figure 1. Reality vs. perception.
(a) While programmers make a great deal of design decisions when developing a software system, only few are architecturally significant. (b) Most design decisions, even minor ones, are perceived as having an impact on the software architecture.



If the yellow circle in Figure 1a represents all decisions made for a software system, design decisions (purple) will be a subset, leaving many decisions at the programming level. In turn, a small subset of these design decisions will be architecturally significant (red). Some decisions are made “upstream” in the form of requirements constraints (green).

Unfortunately, the decision landscape is beginning to look more like Figure 1b, where not much distinction is left between design and architecture (purple equals red). Mary Shaw warned us a long time ago: “Do not dilute the meaning of the term architecture by applying it to everything in sight.”⁸ Not all design decisions are architectural. Few are, actually, and in many projects, they’re already made on day one.

Context Is Key

How much architectural activity will a project need? It usually fluctuates widely depending on the project’s context. By context we mean the project’s environment—the organization, the

domain, and so on—as well as specific factors such as project size, stable architecture, business model, team distribution, rate of change, the system’s age, criticality, and governance (see Figure 2). Other influences can include the market situation, the company’s power and politics, the product’s expected life span, product type, organizational culture, and history.

In an agile project’s “sweet spot,” few architectural activities might be needed. However, many large, complicated projects require significant architectural effort. For these projects, agile methods must suit the specific circumstances inherent in the development’s context.

Architecture: When in the Life Cycle?

When should we focus on architecture? Well, early enough, because “architecture encompasses the set of significant decisions about the structure and behavior of the system.”⁵ These decisions will prove the hardest to undo, change, and refactor, which means to not only focus on architecture but also interleave architectural “stories” and functional “stories” in early iterations.

What’s the Architect’s Role?

Who are the architects? On a large, challenging, novel system, you might need a good mix of experience:

- *architectus reloadus*, the maker and keeper of big decisions, focusing on external coordination, and
- *architectus oryzus*, mentor, prototyper, and troubleshooter, concentrating more on code-facing and focusing on internal coordination,

to follow Martin Fowler’s metaphors.⁸

Not All Documentation Is Bad

How much of an explicit description of the architecture does the project need? In most cases, an architectural prototype will suffice, starting with a walking skeleton along with a small number of solid metaphors to convey the message. But some circumstances require more explicit software architecture documentation—for example, to communicate to a large audience or to comply with external regulations.

There’s a Method to It

How do we identify and resolve architectural issues? Although we’ve shown that some agile methods aren’t opposed to the concept of architecture, they’re all rather silent on how to identify

architecturally significant requirements, perform incremental architectural design, validate architectural features, and so on. Architectural methods exist (see the “Related Work” sidebar for a quick inventory) but aren’t well known.

Architecture Has Value

What’s the cost of architecture anyhow? All agile approaches strive to deliver business value early and often. The problem often seems that whereas the architecture’s cost is somewhat visible, its value is hard to grasp, because it remains invisible. An approach such as incremental funding² might let us find the right compromise between architecture and functionality, without falling into the trap of BDUF.

Bridging the Gap— Leaders and Supporters

On the basis of our observations, experiences, ongoing field studies, and the articles in this special issue, we conclude that emphasis has increased on software architects’ vital role and responsibilities in successfully combining agile and architectural methodologies. Software architects are expected to facilitate software development projects as well as represent the overall system’s quality attributes. How should software architects’ roles change? Or what new responsibilities should they take? We already know that architects can be quite satisfied with agile development. Kati Vilkki surveyed more than 2,400 developers, testers, architects, and managers at Nokia Siemens Networks. She concluded that more than 70 percent of the architects were either satisfied or very satisfied when considering agile development’s impact on their work.⁹ One reason might be the improved feedback cycle and their new role closer to the actual development.

It’s also becoming clear that software developers are equally important in successfully combining agile and architecture approaches because it’s up to the development team how to use various architectural artifacts and documents. So, we need to know agile software developers’ perceptions of software architecture’s relevance and usefulness in their daily activities. It’s also important to understand what agile developers think about combining architectural principles and agile approaches in development projects. One way is to find out how agile teams use software architecture. If agile developers don’t consider software architecture relevant to their day-to-day activities, it would be difficult, even impossible, to convince them to use archi-

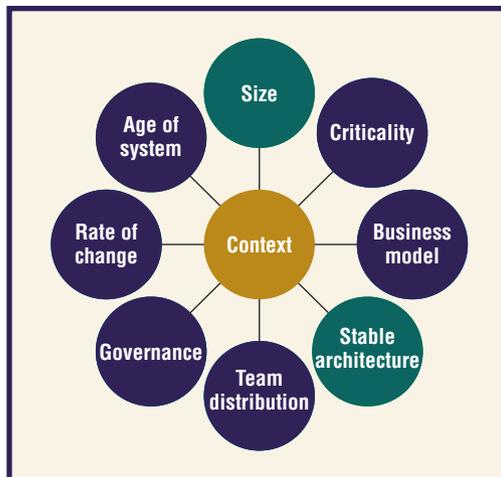


Figure 2. Some factors making up a project’s context. Like other software design and implementation activities, the project’s context, including the customer, needs to drive the project’s architectural activities.

tectural principles and integrate artifacts in agile development.

The Articles in This Issue

Contrary to the perception that architecture is less relevant to agile developers, Davide Falessi, Giovanni Cantone, and Salvatore Alessandro Garcia’ found that agile developers perceive software architecture as relevant on the basis of aspects such as communication among team members, inputs to subsequent design decisions, documenting design assumptions, and evaluating design alternatives. These findings are consistent with other reports that agile teams tend to have some sort of architectural documentation.¹⁰

Falessi’s results also suggest that agile developers usually focus on software architecture while working on complex software systems that can be characterized by geographical distributed development, many stakeholders, or many requirements or LOC. This is in line with the common observation that as software complexities increase, so too does the relevance and importance of software architecture and its related documents.

Each development approach is usually based on some fundamental requirements. From the outset, agile values appear to contradict good architectural practices. Any attempt to combine agile and architecture approaches must consider the potential value clash that development teams might perceive. Falessi and his colleagues concluded that the respondents in their study found that agile values and architectural principles support each other.

Timeliness is vital for architectural decisions because they can prove difficult or costly to change. So, architects working with agile or nonagile teams must know the most appropriate time for key architectural decisions. Making such

Related Work on Software Architecture

Agile followers and critics are familiar with the literature on the Agile Manifesto, its principles, and its approaches. We assume that readers of this special issue might want to know about the sources of literature on software architecture methods, approaches, and tools that they can access or customize to integrate into agile processes and practices. We have space to describe only a few sources of literature on software architecture design, documentation, and review phases, but these sources can direct readers to more books and papers on the topic.

The software architecture community has developed various methods and techniques to design software architecture. For example, Jan Bosch proposes a method that explicitly considers nonfunctional requirements during design.¹ Christine Hofmeister and her colleagues propose a framework and global analysis to identify, accommodate, and describe architecturally significant factors including quality attributes early during design.² Lawrence Chung and his colleagues provide a framework to systematically deal with nonfunctional requirements during design.³ The Software Engineering Institute's software architecture group has developed methods to support architectural design including attribute-driven design⁴ and attribute-based architecture styles.⁵ Some quality attribute communities have developed different methods to support systematic reasoning about their respective quality attributes—for example, real time,⁶ reliability,⁷ and performance⁸—during software architecture design and review.

Architecture is a vehicle for communication among stakeholders, so it should be described unambiguously and in

sufficient detail to provide relevant information to each type of stakeholder.⁹ An important issue is to choose a suitable approach that can serve the main goals of documenting architectures, such as communication, analysis, implementation, and maintenance. One recommended practice is to use various architectural views.^{10–12} A suitable architectural description language (ADL) is also required to describe the architectural decisions. There are many ADLs,¹³ including the Unified Modeling Language,¹⁴ that can describe software architecture. ADLs are usually supported by a proprietary or research tool that practitioners can evaluate for suitability and customization for use with agile approaches.

Architecture reviews are an effective way to identify potential risks and questionable design decisions early in the software development life cycle. Some well-known architecture review methods include the Scenario-based Architecture Analysis Method (SAAM),¹⁵ Architecture-Level Modifiability Analysis (ALMA),¹⁶ Architecture trade-off analysis,¹⁵ and the performance analysis of software architecture.¹⁷ Apart from technical decisions, architecture reviews also involve several kinds of nontechnical decisions such as whom to involve, how to select evaluators, how to fund a review, and other organizational and managerial factors. Some researchers and practitioners have provided guidelines and heuristics for dealing with these aspects of the reviews.^{18–20}

There's an increasing body of knowledge on software architecture research and practice published in books, journal articles, and conference papers. As an additional source, *IEEE Software* published two special issues on software architecture, one in November 1995, which offers traditional

decisions too early can constrain development teams in general and agile development teams in particular. Waiting too long to take care of architecturally significant decisions can put the whole project in chaos. To help stakeholders make timely decisions, Stuart Blair, Tim Cull, and Richard Watt present an approach called responsibility-driven architecture that exploits concepts of the real options theory. A simple spreadsheet-based tool and a responsible, accountable, consulted, and informed (RACI) matrix support their approach to track stakeholders' decision-making and responsibilities.

User stories in agile development relate primarily to functional requirements; this means that nonfunctional requirements can sometimes get completely ignored. Unfulfilled nonfunctional requirements can make an otherwise fully functioning system useless or risky. A main objective of integrating architectural approaches in agile processes is to enable software development

teams to pay attention to both functional and nonfunctional requirements. Roland Faber, in "Architects as Service Providers," proposes that architecture should represent nonfunctional requirements. He presents an architectural process in the context of agile projects that exploits well-known architectural concepts and principles. He also provides advice on tailoring scrum to incorporate the concept of architecture as a service and facilitating communication between architecture and development teams.

This special issue closes with an article by James Madison, who advocates the coexistence of agile and architecture as complementary approaches and principles. He emphasizes the software architect's vital role as a linchpin for combining the two. Madison's approach, called *agile architecture*, advocates using agile to get to a good architecture by appropriately applying suitable combinations of architectural functions (such as communication, quality attributes, and

perspectives on aspects of software architecture, and another in March/April 2006, which includes key references to software architecture literature. Other special issues of *IEEE Software* have included references to more specialized topics in software architecture—for example, the special issue on design patterns (July/August 2007) and the special section on capturing design knowledge (March/April 2009). To bridge the community and build up cumulative learning, we maintain a Wikipedia site for agile- and architecture-related resources at www.acube-community.org/wikis/index.php/Architecture-Centric_Methods_and_Agile_Approaches.

IEEE Software recently cosponsored the Software Architecture Challenges in the 21st Century workshop (<http://computingnow.computer.org/sac21>). Hakan Erdogmus briefed us on its outcomes and proposed that “every software-intensive system has an architecture at its soul, regardless of the process used to develop the system.”²¹ His forecast, based on workshop presentations, was to “give the software architecture a pivotal role in the development process, at least a role that’s much more essential than is ordinarily granted in the mainstream agile [software development approaches].”²¹

References

1. J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*, Addison-Wesley, 2000.
2. C. Hofmeister, R.L. Nord, and D. Soni, *Applied Software Architecture*, Addison-Wesley, 2000.
3. L. Chung et al., *Non-functional Requirements in Software Engineering*, Kluwer Academic Publishers, 1999.
4. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed., Addison-Wesley, 2003.
5. M.H. Klein and R. Kazman, *Attribute-Based Architectural Styles*, tech. report CMU/SEI-99-TR-022, Software Eng. Inst., Carnegie Mellon Univ., 1999.
6. M.H. Klein et al., *A Practitioner’s Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Publishers, 1993.
7. M.R. Lyu, *Handbook of Software Reliability Engineering*, McGraw-Hill and IEEE CS Press, 1996.
8. C.U. Smith and L.G. Williams, “Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives,” *IEEE Trans. Software Eng.*, vol. 19, no. 7, 1993, pp. 720–741.
9. P. Clements et al., *Documenting Software Architectures: Views and Beyond*, Addison-Wesley, 2002.
10. *IEEE Std. 1471-2000, Recommended Practices for Architecture Description of Software-Intensive Systems*, IEEE, 2000.
11. P. Kruchten, “Architectural Blueprints—the ‘4+1’ View Model of Software Architecture,” *IEEE Software*, vol. 12, no. 6, 1995, pp. 42–50.
12. N. Rozanski and E. Woods, *Software Systems Architecture*, Addison-Wesley, 2005.
13. N. Medvidovic and R.N. Taylor, “A Classification and Comparison Framework for Software Architecture Description Languages,” *IEEE Trans. Software Eng.*, vol. 26, no. 1, 2000, pp. 70–93.
14. M. Fowler, *UML Distilled*, 3rd ed., Addison-Wesley, 2004.
15. P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, 2002.
16. P. Bengtsson et al., “Architecture-Level Modifiability Analysis (ALMA),” *J. Systems and Software*, vol. 69, nos. 1–2, 2004, pp. 129–147.
17. L.G. Williams and C.U. Smith, “PASA: A Method for the Performance Assessment of Software Architecture,” *Proc. 3rd Int’l Workshop Software and Performance*, ACM Press, 2002, pp. 179–189.
18. M. Ali-Babar and I. Gorton, “Software Architecture Reviews: The State of the Practice,” *Computer*, vol. 42, no. 7, 2009, pp. 26–32.
19. R. Kazman and L. Bass, “Making Architecture Reviews Work in the Real World,” *IEEE Software*, vol. 19, no. 1, 2002, pp. 67–73.
20. J.F. Maranzano et al., “Architecture Reviews: Practice and Experience,” *IEEE Software*, vol. 22, no. 2, 2005, pp. 34–43.
21. H. Erdogmus, “Architecture Meets Agility,” *IEEE Software*, vol. 26, no. 5, 2009, pp. 2–4.

design patterns) and architectural skills at four points (up-front planning, storyboarding, sprint, and working software) in the development life cycle. Madison provides several examples from successful applications of agile architecture to 14 projects in various domains.

So, what does this special issue offer for those interested in designing and deploying agile processes engrained with sound architectural principles and practices?

- Understand the context. There’s a vast array of software development situations, and although “out of the box” agile practices address many of these, there are outliers that we need to understand. What’s the system’s size, domain, and age? What’s the business model and the degree of novelty and hence of risk? How critical is the system? How many par-

ties will be involved?

- Clearly define the architecture: its scope and the architect’s role and responsibility. Don’t assume a tacit, implicit understanding.¹¹
- Define an architecture owner, just as you define a product owner and project leader. But don’t let the architects lock themselves in an ivory tower, polishing the ultimate architecture for an improbable future system. Architects are part of the development group.
- Exploit architecture to better communicate and coordinate among various parties, particularly multiple distributed teams, if any. Define how to represent the architecture, on the basis of various parties’ need to know.
- Use important, critical, and valuable functionality to identify and assess architectural issues. Understand interdependencies between technical architectural issues and visible user functionality to weave them appropriately over time (the zipper metaphor).

About the Authors



Pekka Abrahamsson is a professor of computer science at the University of Helsinki. He's been an active member of the agile community since 2002. His current responsibilities include managing a Flexi-ITEA2 research project, which involves 35 organizations from seven European countries. The project aims to develop agile approaches in the domain of global, large, and complex embedded-systems development. Abrahamsson has a PhD in software engineering from the University of Oulu. He is a member of the *IEEE Software* Advisory Board. Contact him at pekka.abrahamsson@cs.helsinki.fi.

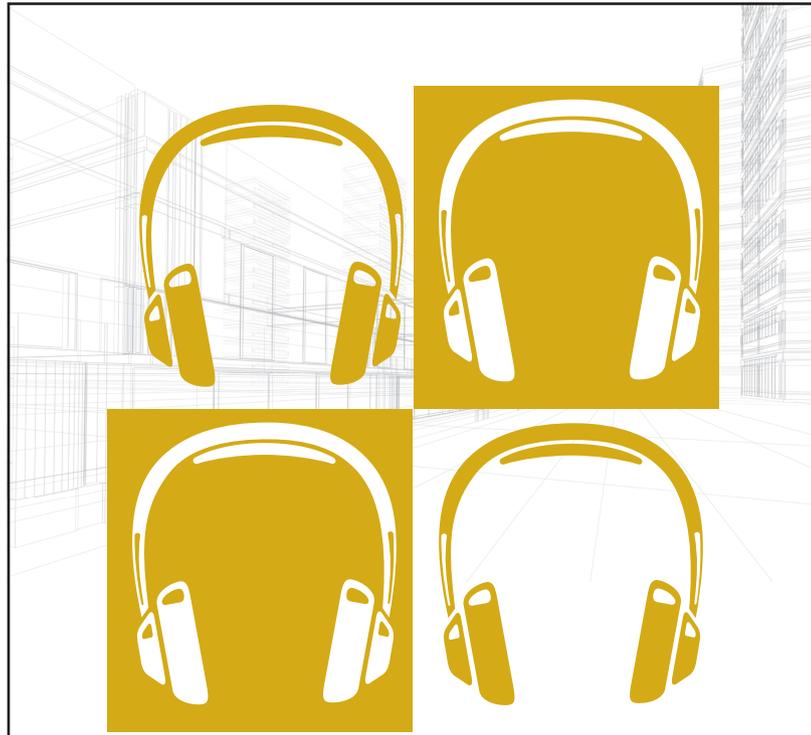
Muhammad Ali Babar is an associate professor at the IT University of Copenhagen. His research interests include software architecture, agile approaches, and global software development. Ali has a PhD in computer science and engineering from the University of New South Wales. Contact him at malibaba@itu.dk.



Philippe Kruchten is a professor of software engineering at the University of British Columbia. He's a founding member of the International Federation for Information Processing Working Group 2.10 on Software Architecture, and the cofounder and chair of Agile Vancouver. In a previous life he led the development of the Rational Unified Process, which was more agile in its intent than practitioners have made of it, and which embodied an architectural method. Kruchten has a PhD in information systems from Ecole Nationale Supérieure des Télécommunications. He is a member of the *IEEE Software* Editorial Board. Contact him at pbk@ece.ubc.ca.

- Understand when it's appropriate to freeze the architecture to provide developers the necessary stability to finish a product release, and what amount of technical debt will then accumulate.
- Keep track of unresolved architectural issues, either in the backlog or in the risks. Deferring decisions to the last responsible moment doesn't mean ignoring them but can add risks that must be managed like other risks in the project.

In a large software organization, implementing agile approaches isn't a straightforward adoption problem. Most likely, it will take several years to shorten the feedback cycles to benefit from the adaptability and earlier value-creation opportunities. Failure is a natural part of process improvement. We believe that stories and studies of failure often shed more light than those of success. We encourage practitioners and researchers to publish these more actively to stimulate discussion and increase learning in the community. 



LISTEN TO GRADY BOOCH
"On Architecture"
podcast available at  <http://computingnow.computer.org>

References

1. P. Kruchten, "Voyage in the Agile Memplex: Agility, Agilese, Agilitis, Agilology," *ACM Queue*, vol. 5, no. 5, 2007, pp. 38–44.
2. K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 2000.
3. J.A. Highsmith, *Agile Software Development Ecosystems*, Addison-Wesley, 2002.
4. S. Augustine, *Managing Agile Projects*, Prentice Hall, 2005.
5. P. Kruchten, *The Rational Unified Process—an Introduction*, 1st ed., Addison-Wesley, 1998.
6. A. Cockburn, *Crystal Clear: A Human-Powered Methodology for Small Teams*, Addison-Wesley, 2004.
7. M. Poppendieck and T. Poppendieck, *Implementing Lean Software Development: From Concept to Cash*, Addison-Wesley, 2007.
8. D. Garlan, "1st Int'l Workshop on Architectures for Software Systems Workshop Summary," *Software Eng. Notes*, vol. 20, no. 3, 1995, pp. 84–89.
9. K. Vilkki, "Impact of Agile Transformation," *Flexi Newsletter*, vol. 2, no. 1, 2008, pp. 5–6.
10. M. Ali-Babar, T. Itheme, and M. Pikkarainen, "An Industrial Case of Exploiting Product Line Architectures in Agile Software Development," *Proc. 13th Int'l Software Product Line Conf. (SPLC 09)*, Carnegie Mellon Univ., 2009, pp. 171–180.
11. P. Kruchten, "The Software Architect, and the Software Architecture Team," *Software Architecture*, P. Donohue, ed., Kluwer Academic Publishers, 1999, pp. 565–583.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.