Decision-Centric Architecture Reviews

Uwe van Heesch, Capgemini Germany

Veli-Pekka Eloranta, Tampere University of Technology

Paris Avgeriou, University of Groningen

Kai Koskimies, Tampere University of Technology

Neil Harrison, Utah Valley University

// Architecture evaluation is an important activity in the software engineering life cycle, but unfortunately, it isn't regularly practiced in industry. Decision-centric architecture reviews uncover and evaluate the rationale behind the most important architecture decisions. Experiences in large industrial projects have shown that full-scale DCAR evaluations, including reporting, can be conducted in fewer than five person-days while still producing satisfying results for stakeholders. //

SOFTWARE ARCHITECTURE that's poorly designed or carelessly cobbled together can make an entire software project fail. Therefore, it's important to evaluate software architecture early on in its development. Researchers have proposed various software architecture evaluation methods to systematically uncover architectural problems;[1,2] the most popular are scenario-based—for example, the Architecture Tradeoff Analysis Method (ATAM).[3] In general, architecture evaluations have several benefits, but the most important is to identify problems or risks early enough so that they can be more easily fixed or mitigated than problems found later, such as in the testing or integration phases, or even during maintenance.[4] Furthermore, architecture evaluations encourage communication among the involved stakeholders that wouldn't take place otherwise.

But despite these benefits, architecture evaluation isn't widely adopted in industry today;[1,2] most organizations are aware of its benefits, but very few practice it. A study of software architects uncovered the typical prerequisites that influence an organization's architecture evaluation practices, such as management commitment, company-wide evaluation standards, funding models, and appropriate training, but that aren't often met.[5] Furthermore, the increasingly popular agile development approaches don't encourage the use of architecture evaluation methods because they typically consume a considerable amount of time and resources.

To lower the threshold of industrial adoption, we developed a new evaluation method called *decision-centric architecture review*. We built DCAR from the ground up based on our experiences with performing architecture evaluations in industry and observing what works well in practice. This led to two high-level requirements: first, DCAR had to be lightweight in terms of required time and resources, and second, it had to support a decision-by-decision software architecture evaluation, letting its users systematically analyze and

DCAR: SHORT PROFILE

Evaluation objectives: determine the soundness of architectural decisions that were made
Inputs for evaluation: informal description of requirements, business drivers, and architectural design
Knowledge of evaluators: general knowledge about software architecture
Output: risks, issues, and thorough documentation of the evaluated decisions and their decision forces
Priority setting of decisions: during the review
Project phase: within or after the architectural design is finalized
Reviewers: company-internal or external reviewers
Schedule: half a day preparation and postprocessing and half a day review session
Scope: a set of specific architecture decisions
Social interaction: face-to-face meeting between reviewers, architect, developers, and business representative
Tools or automation: templates, wiki, and UML tool

record the rationale behind architecture decisions. The latter requirement differentiates the method from scenario-based methods, which test software architectures against scenarios that refine a system's major quality requirements.

We've performed multiple DCAR evaluations so far, and our experiences indicate that an average DCAR session takes a half-day, requiring the presence of three to five members of the project team, including the chief architect. Thus, the total amount of company time is less than three person-days plus another two person-days for the review team, which makes DCAR especially suitable for projects that don't have the budget, schedule, or stakeholders available for full-fledged architectural evaluations. DCAR is also pertinent for projects that need an evaluation to justify a set of architecture decisions rather than to ensure that a whole system satisfies its quality requirements.

Architecture Decisions

DCAR is decision-centric in the sense that the evaluation starts when stakeholders (with the review team's assistance) select a set of decisions to analyze in the context of relevant project- and company-specific decision forces. A decision force, or force for short, is any nontrivial influence on an architect seeking a solution to an architectural problem (see the sidebar for more background). DCAR can be used for any set of architectural decisions of any type: it's applicable for all types of software-intensive systems and domains. Understanding architecture decisions and the rationale behind them is crucial for continuously ensuring system integrity.

Architecture decisions are the fundamental choices an architect has to make about a software system's overall structure or externally visible properties.[6] Typical examples include the choice of an architectural pattern or style, the selection of a middleware framework, or the decision not to use open source components for licensing considerations.

Architecture decisions aren't isolated; they can be seen as a web of interrelated decisions that depend on, support, or contradict each other. Some decisions must be combined to achieve a desired property—others are solely made to compensate for a negative impact. As an example, an architect could decide to use an in-memory database to achieve short response times, but this decision has a negative impact on reliability, which, in addition to short response times, is another desired property of the system. To compensate for this negative impact, the architect could decide to use redundant power supplies or to replicate the database and the hardware and use the replica as a hot spare. The decisions to use redundant power supply and a hot spare would then be caused by the decision to use an in-memory database.

In DCAR, the participants identify the architecture decisions and clarify their interrelationships. This is primarily done for two reasons: first, understanding the relationships helps identify influential decisions that have wide-ranging consequences for large parts of the architecture, and second, when a specific decision is evaluated, it's important to consider its related decisions as well.

Several factors must be taken into consideration to evaluate an architecture decision, including constraints, risks, political or organizational considerations, personal preferences, experience, and business goals such as quick time to market and low price. These decision forces[7] each have a direction and a magnitude, pushing an architect either toward or away from a specific solution.

To evaluate an architectural solution, the related decisions also must be contemplated and considered as decision forces. In their totality, these forces reveal the entire context in which a decision is made. Because some of them can be in conflict with—or orthogonal to—each other, an architect must balance all the forces to make the best possible decision. Figure 1 illustrates this concept using the in-memory database decision described earlier. In this particular case, the forces in favor of the in-memory database outweigh the forces against it. DCAR explores the entire rationale behind decisions via the related forces.

After identifying forces, the review participants examine if the rationale behind the evaluated decision is still valid in the current context. This is important, because forces are not immutable; not only do requirements keep changing, but the tactical orientation of the company may evolve, laws and regulations may have changed, or new technologies could exist that would offer a better solution to a design problem at hand. Such changes in the design context may change the magnitude of the forces, or even introduce new forces and make some of the old forces obsolete. In the new design context, if the negative forces outweigh the positive forces, then the reviewers recommend to reconsider the decision.

Introducing DCAR

To achieve best results, DCAR requires the participation of the lead architect and one or two members from the development team with different roles and responsibilities. Additionally, somebody has to represent the management and customer perspectives. This is important because some decisions must be assessed from an enterprise-wide perspective rather than taking only the project-specific forces into account.

The review can be done by external reviewers or an organization's own staff members who aren't directly involved in the project under review. However, the review team's members must have experience in designing software architecture, ideally (but not necessarily) in the same domain as the system under review.

Figure 2 shows DCAR's main steps, as well as the produced artifacts (the boxes on the right). Step 1 happens offline, but all the other steps are performed during an evaluation session in which all participants gather in one room.

Step 1: Preparation

A date for the DCAR session is settled, and the stakeholders are invited to participate. The system's lead architect prepares a presentation that should contain the most important architectural requirements, high-level views of the architecture, the approaches used (such as patterns or styles), and the technologies used (such as database management systems or middleware servers). The representative for the management and customer perspectives prepares a presentation describing the software product and its domain, the business environment, market differentiators, and driving business requirements and constraints. Templates for both presentations can be found at www.dcar-evaluation.com.

The review team receives the presentation slides prior to the evaluation session, so that they can prepare for the meeting. In particular, the reviewers study the material to elicit potential architecture decisions and decision forces. Additional system documentation isn't mandatory, but anything that the reviewers can use to understand the system upfront is helpful.

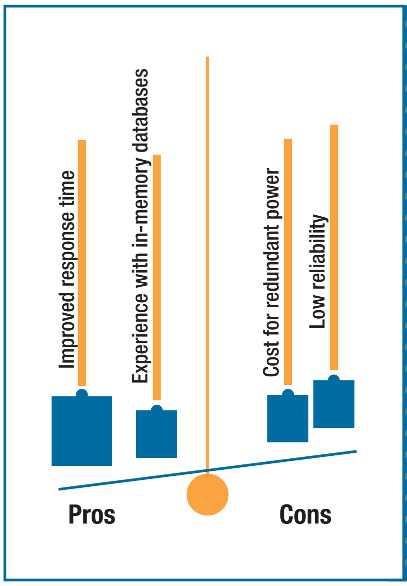


FIGURE 1. Decision forces have different weights and may contradict each other. Here we see some forces for the in-memory database decision.

Step 2: Introduction to DCAR

The evaluation session starts with an introductory presentation of the DCAR method to all participants. This includes the day's schedule, an introduction to the DCAR steps, the evaluation's scope, possible outcomes, and participant roles and responsibilities. The DCAR website provides an example.

Step 3: Management Presentation

The management/customer representative gives a short presentation

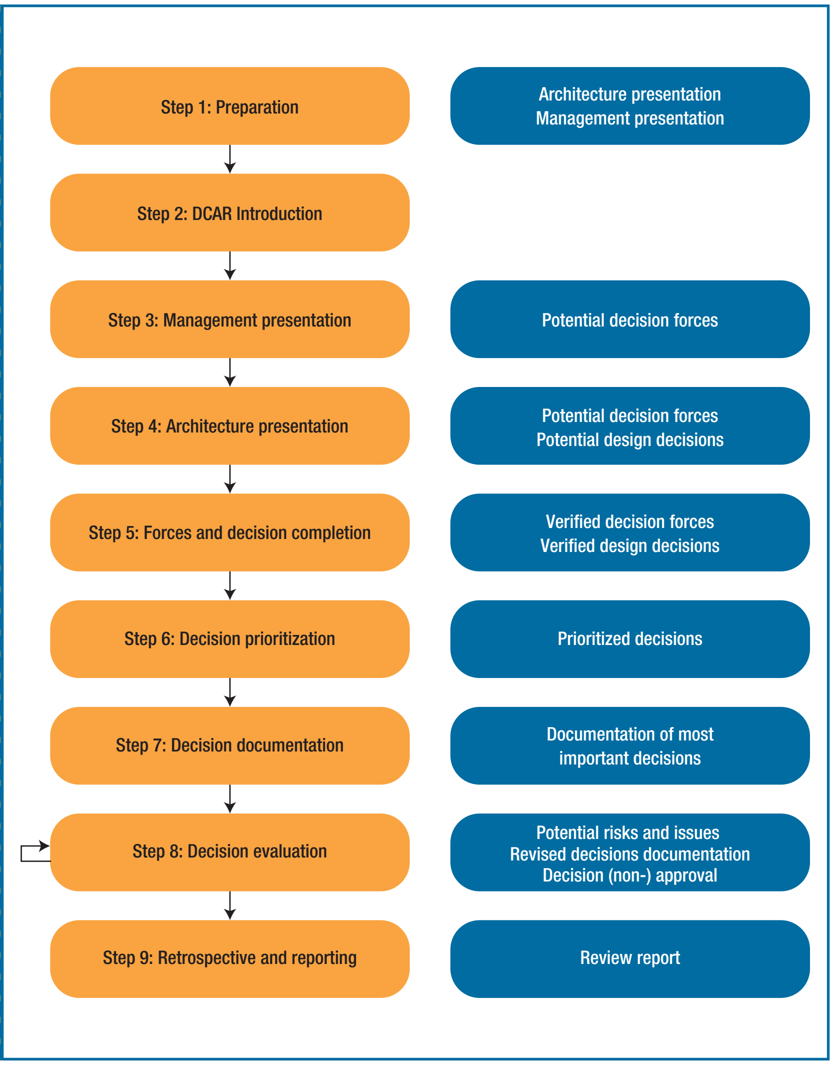

Step 1: Preparation
Architecture presentation
Management presentation
Step 2: DCAR Introduction
Step 3: Management presentation
Potential decision forces
Step 4: Architecture presentation
Potential decision forces
Potential design decisions
Step 5: Forces and decision completion
Verified decision forces
Verified design decisions
Step 6: Decision prioritization
Prioritized decisions
Step 7: Decision documentation
Documentation of most important decisions
Step 8: Decision evaluation
Potential risks and issues
Revised decisions documentation
Decision (non-) approval
Step 9: Retrospective and reporting
Review report

FIGURE 2. The decision-centric architecture review. DCAR has nine sequential steps, each one producing different artifacts.

using the slides prepared in Step 1. In our experience, 15 to 20 minutes should suffice, but more time can be used if the schedule allows it. The main purpose is to let the reviewers elicit business-related decision forces that must be taken into consideration during the evaluation. The review team notes any potential forces during the presentations and asks questions to elicit additional ones. The management/customer representative doesn't need to be present during the rest of the session, but he or she might provide additional insights during the decision analysis.

Step 4: Architecture Presentation

The lead architect uses the slides prepared in Step 1 to introduce the architecture to all DCAR participants. In our own industrial DCAR sessions, we reserved between 45 and 60 minutes for this presentation. The goal is to give all participants a good mental picture of the architecture, and the presentation is supposed to be highly interactive. The review team and the other participants ask questions to complete and verify their understanding of the system. During this step, the reviewers revise and complete the list of architecture decisions they identified as preparation in Step 1. Identifying architecture decisions requires some experience. As a starting point, reviewers can focus on the technologies used, such as servers, frameworks, and third-party libraries. Additionally, it has been a good practice to search for applied patterns in the architecture.[8]

Apart from capturing architecture decisions, the reviewers revise and complete the list of forces they identified in Steps 1 and 2. Forces can be documented as informal statements; both decisions and forces are revisited in the next step.

Step 5: Forces and Decision Completion

At this stage, the reviewers have assembled a preliminary list of architecture decisions and decision forces, so Step 5's goal is twofold: clarify the architecture decisions and their relationships, and complete and verify the forces relevant to these decisions. To support the clarification of decision relationships, one of the reviewers creates a decisions relationship diagram[9] that is constantly revised during the previous steps. Figure 3 shows an excerpt of such a diagram.

Each decision is represented by an ellipse that contains a short descriptive name for the decision. It's important to use the company's own vocabulary for these names, so that reviewers and stakeholders have the same understanding of the applied architectural solution. In the beginning, each decision collected by the reviewers in the previous step is represented in the diagram. After all the participants gain a collective understanding of the decisions, the relationships are established through a directed line in the diagram. Although multiple relationship types exist,[9] the only important relationships in an architecture review are *caused by* and *depends on*. These relationships help both reviewers and stakeholders estimate each decision's importance. Relationships are also helpful for understanding which decisions must be taken into consideration as decision forces for other decisions. Any UML tool can create a relationship diagram; a template is available at www.dcar-evaluation.com.

The forces, presented as a bulleted list, should be formulated unambiguously using domain-specific vocabulary—for example, forces from the machine control domain might be "Firmware-level design and implementation should be sourced out, as this isn't our core business," or "We have a lot of in-house experience with the CANOpen protocol." The review team discusses and completes the list of forces with the company participants.

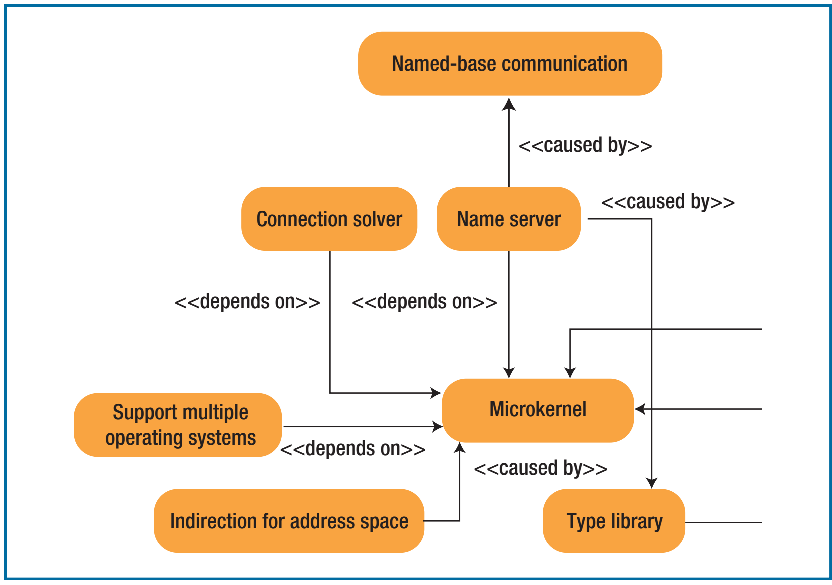


FIGURE 3. The relationship view illustrates decisions and their relationships. In this example, we're looking at an excerpt from a relationship view created in a DCAR session.

Name	Redundancy of controllers			
Problem	The application should run even if the server fails			
Solution or description of decision	The system is deployed to two servers: one is active, the other one is inactive. The active server provides all system services, while the passive one is running in the background. When the active server fails, the inactive server becomes active. During the switch over, the active server tries to update the passive one to make sure that it has the same data and status. Both servers have an identical software configuration. This solution follows the *Redundant Functionality Pattern*.			
Considered alternative solutions	Apply the *Redundancy Switch Pattern*: Both servers are active; external logic is used to decide which output is actually used in the control. In this case, cyclic data copying could be avoided. However, applying this solution would require major modifications to the system. Even though availability would be increased, it would also cause additional costs. The customers are not prepared for paying more for higher availability. Additionally, the external logic component could become a potential single point of failure. Therefore, this alternative was discarded.			
Forces in favor of decision	• Easier to implement than the alternative solution • Scales easily to versions where redundancy is not used • No additional costs			
Forces against the decision	• Slower switch over time than the alternative would have • Hard to offer higher availability than the current 99.99%			
Outcome	Green	Yellow	Yellow	Red
Rationale for outcome	Current solution seems to be ok.	I am concerned about the slow switch over time.	Widely accepted solution. Availability might become a problem in the future.	We should really reconsider this decision, as the next release is likely to have higher availability requirements.

FIGURE 4. During DCAR, decisions are documented and continuously updated. In the end, the descriptions capture the full rationale behind decisions including the outcome of the evaluation.

TABLE 1

Descriptive statistics.

Variable	Value
Average system size	600,000 SLOC
Average number of elicited decisions after step 5	21 decisions
Average number of decisions evaluated in step 7	9 decisions
Average number of decisions evaluated in step 8	7 decisions
Average number or reviewers	4 people
Average number of company stakeholders	4 people
Average effort for reviewer team	50 person-hours
Average effort for company stakeholders	23 person-hours

Step 6: Decision Prioritization

Usually, the number of decisions elicited in the previous steps is too large to discuss during the review itself, so the stakeholders will have to negotiate which decisions to review in the following steps. The criteria for selecting which decisions will be reviewed are context dependent but should include mission-critical decisions, decisions known to bear risks, and decisions causing high costs.

We use the following procedure to prioritize decisions. Each participant gets 100 points to distribute freely over the decisions identified at this point, based on the previously agreed criteria about their importance. Then the points are summed up and the rationale behind each person's rating is discussed. The decisions with the highest ratings (number of points received) go on to the next steps. In our experience, the number of decisions that can be discussed effectively in a half day is seven to ten.

Step 7: Decision Documentation

The architect and the other company participants document the set of decisions that received the highest ratings in the previous step, and each person selects two or three decisions that he or she is knowledgeable about. The decisions are documented by describing the applied architectural solution, the problem or issue it solves, known alternative solutions, and the forces that must be considered to evaluate the decision. The stakeholders use the list of forces assembled in the previous steps to make sure they don't forget important ones, but they can also think of new forces.

Figure 4 shows an example decision documentation template used in DCAR; other established templates appear elsewhere.[6,8]

Step 8: Decision Evaluation

The next step after documenting the decisions is to evaluate them, starting with the highest-priority decision. The participant who documented the current decision presents it briefly, and then the company participants, together with the reviewers, challenge the decision by identifying additional forces against the chosen solution. They use the elicited decision forces and the decision relationship diagram to understand the decision's context—that is, the circumstances in terms that the decision can be fully understood and assessed. The documentation of both decisions and decision relationship diagrams are continuously updated by one of the reviewers during this step. All participants discuss whether the forces in favor of the decision outweigh the forces against it.

Finally, all participants decide by voting whether the decision is good, acceptable, or has to be reconsidered. Figure 4 shows the result of an evaluated decision created during a DCAR session. The traffic light colors indicate the ratings of all participants: green for good, yellow for acceptable, and red for has to be reconsidered. Additionally, it shows justifications for the votes as given by each voter ("rationale for outcome").

During the whole discussion, the reviewers note any potential issues or risks that were mentioned. Each decision is discussed for approximately 15 to 20 minutes. In our experience, the quality of discussion diminishes at some point. If a decision requires more than 20 minutes, it can be flagged as a point for future analysis.

Step 9: Retrospectives and Reporting

After all of the selected decisions are evaluated, the review team collects the notes and artifacts created during the session. These will serve as input for the evaluation report that the review team writes within two weeks of the session. The report is discussed with the architect for verification and eventually refined by the review team. In our own DCAR sessions, the review team prepares the report the following day, the advantage being that the review team and the architect can still vividly remember the discussions held the day before.

Experiences

We developed DCAR in cooperation with industrial partners from the distributed control system domain, but it is by no means restricted to this domain. Since its initial version, DCAR has been applied and refined in five large software projects. In this section, we report our findings from three industrial sessions conducted in different projects at Metso Automation in Tampere, Finland.

Table 1 gives some descriptive statistics for these sessions, which happened over the span of five hours each. The systems under study came from the process automation domain, and the effort that company stakeholders had to expend on the reviews reveals the time spent by the participants for preparation, taking part in the evaluation sessions, and reviewing the evaluation report.

To gather feedback on the participants' perception of DCAR, we interviewed a subset of them, including the chief architect. Apart from the chief architect, who naturally knows the architecture best, all interviewees mentioned that they received a good overview of the system's architecture, something they were missing in their daily work because they were only responsible for smaller subsystems. They also stated that they liked that all important decisions, even if they were considered stable, were brought into question for the purpose of the evaluation. The prioritization procedure in Step 6 made sure that bias on behalf of a decision maker or the responsible architect was reduced. Systematically discussing decisions in a group also helped everyone understand different points of view that need to be considered in the decision's context.

Generally, the participants reported that interactions between the stakeholders and discussions with the review team as external contributors were the most valuable advantages of the evaluation session. The chief architect noted that the evaluation report, produced by the review team, was a valuable supplement to the existing system documentation. The interviewees estimated that the decisions elicited during the evaluation roughly covered the most important 75 percent of all significant architecture decisions; this was regarded as an excellent result given the short amount of time invested in the evaluation.

DCAR's success depends on stakeholders' understanding about architecture decisions and decision forces, so we explicitly addressed these issues in the interviews. Although all interviewees were either already familiar with both terms or grasped the concepts quickly during the DCAR introduction in Step 2, some of them mentioned that the time given for the documentation of decisions in Step 7 was too short. This was particularly the case for stakeholders who had never systematically documented architecture decisions before. They proposed to tackle this problem by providing examples of documented decisions prior to the evaluation.

During the evaluations, we observed that the documentation of reasoning—the forces in favor or against a specific solution—was especially challenging for some of the participants. Therefore, in later evaluations, we provided examples with a list of typical decision forces in the domain at hand and alleviated the problem.

These positive experiences and the continuous interest from other industrial partners to hold more evaluations in the future show that DCAR helps organizations adopt architectural evaluations as part of their best practices. We'll conduct additional empirical studies to provide evidence about how far DCAR does lower the threshold for industrial adoption of architecture evaluations.

All participants decide by voting whether the decision is good, acceptable, or has to be reconsidered.

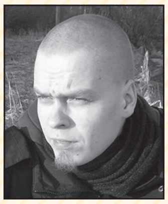
Acknowledgments
We thank the architects and developers at Metso Automation in Tampere, Finland. This work was partly funded by the Finnish Funding Agency for Technology and Innovation (project Sulava).

References

1. L. Dobrica and E. Niemela, “A Survey on Software Architecture Analysis Methods,” *IEEE Trans. Software Eng.*, vol. 28, no. 7, 2002, pp. 638–653.
2. L. Bass and R. Nord, “Understanding the Context of Architecture Evaluation Methods,” *Proc. Joint 10th Working IEEE/IFIP Conf. Software Architecture and 6th European Conf. Software Architecture*, IEEE, 2012, pp. 277–281.
3. R. Kazman, M. Klein, and P. Clements, “ATAM: Method for Architecture Evaluation,” tech. report, Software Eng. Inst., Carnegie Mellon Univ., 2000; www.sei.cmu.edu/publications/documents/00.reports/00tr004.html.
4. J. Maranzano et al., “Architecture Reviews: Practice and Experience,” *IEEE Software*, vol. 22, no. 2, 2005, pp. 34–43.
5. M. Babar, L. Bass, and I. Gorton, “Factors Influencing Industrial Practices of Software Architecture Evaluation: An Empirical Investigation,” *Proc. Quality of Software Architectures 3rd Int’l Conf. Software Architectures, Components, and Applications*, Springer, 2007, pp. 90–107.
6. J. Tyree and A. Akerman, “Architecture Decisions: Demystifying Architecture,” *IEEE Software*, vol. 22, no. 2, 2005, pp. 19–27.
7. U. van Heesch, P. Avgeriou, and R. Hilliard, “Forces on Architecture Decisions: A Viewpoint,” *Proc. Joint 10th Working IEEE/IFIP Conf. Software Architecture and 6th European Conf. Software Architecture*, IEEE, 2012, pp. 101–110.
8. N. Harrison and P. Avgeriou, “Pattern-Based Architecture Reviews,” *IEEE Software*, vol. 28, no. 6, 2010, pp. 66–71.
9. U. van Heesch, P. Avgeriou, and R. Hilliard, “A Documentation Framework for Architecture Decisions” *J. Systems and Software*, vol. 85, no. 4, 2012, pp. 795–820.

ABOUT THE AUTHORS

UWE VAN HEESCH is a software architect and project manager at Capgemini Germany. His research interests include software architecture, particularly architecture decision modeling, software architectural knowledge management, and architecture evaluation. Van Heesch received a PhD in mathematics and natural sciences from the University of Groningen, the Netherlands. He’s an active member of the European pattern community. Contact him at uwe@vanheesch.net.


VELI-PEKKA ELORANTA is a researcher in the Department of Pervasive Computing at Tampere University of Technology. His research focuses on software architectures, architecture work practices, agile methods, architecture evaluations, design patterns, and pattern languages. Eloranta is active in the pattern community and has served on program committees for several Pattern Languages of Programs (PLoP) conferences. Contact him at veli-pekka.elaoranta@tut.fi.

PARIS AVGERIOU is a professor of software engineering in the Department of Mathematics and Computing Science at the University of Groningen, the Netherlands, where he has led the software engineering research group since September 2006. His research interests lie in the area of software architecture, with strong emphasis on architecture modeling, knowledge, evolution, and patterns. Avgeriou serves on the editorial board of *Springer Transactions on Pattern Languages of Programming*, and he has edited special issues of *IEEE Software*, *Elsevier Journal of Systems and Software*, and *Springer Transactions on Pattern Languages of Programming*. Contact him at paris@cs.rug.nl.

KAI KOSKIMIES is a professor of software engineering in the Department of Pervasive Computing at Tampere University of Technology. He has headed the Finnish Graduate School on Software Systems and Engineering and leads ongoing research related to software architectures, with special focus on patterns, evaluation, and automated design. Contact him at kai.koskimies@tut.fi.

NEIL HARRISON is an associate professor of computer science at Utah Valley University in Orem, Utah. He’s the author of numerous publications on software architecture, software patterns, effective organizations, agile software development, and software testing. Harrison received a PhD in mathematics and natural sciences from the University of Groningen, the Netherlands. He’s a member of ACM. Contact him at neil.harrison@uvu.edu.