Defeasible Argumentation of Software Architectures

José Miguel Cañete-Valdeón*, Antonio Ruiz-Cortés† and Miguel Toro‡
Department of Computer Languages and Systems
University of Sevilla, Spain
*jmcv@us.es †aruiz@us.es ‡mtoro@us.es

***Abstract*—Defeasible argumentation is typical of legal and scientific reasoning. A defeasible argument is one in which the conclusion can be accepted tentatively in relation with the evidence known so far, but may need to be retracted as new evidence comes in. This paper analyses the role of defeasible argumentation in the explanation and evaluation of architectural decisions. We analyse technical explanations offered by engineers at Twitter and eBay about several architectural decisions adopted in those systems. We generalize these examples in four argumentation schemes. We also study the typical case of reasoning with a formal model of an architecture, and we infer a fifth argumentation scheme. Finally, we apply Hastings' method of attaching a set of critical questions to each scheme. We show that the existence of critical questions reveals that the inferred schemes are defeasible: in argumentation theory, if a respondent asks one of the critical questions matching a scheme and the proponent of an argument fails to offer an adequate answer, the argument defaults and the conclusion is retracted. This dialogical structure is the basis of typical architectural evaluations. We conclude that the provided evidence supports the hypothesis that defeasible argumentation is employed in architectural evaluation. In this context, a rich catalogue of argumentation schemes is a useful tool for the architect to organize his or her reasoning; critical questions assist the architect in identifying the weak points of his or her explanations, and provide the evaluation team with a checklist of issues to be raised.**

***Index Terms*—Software architectures, design rationale, architectural evaluation, defeasible argumentation, argumentation schemes.**

I. Introduction

The defense of a software architecture can be framed under the *theory of argumentation*, a rich interdisciplinary area of research spanning philosophy, communication studies, linguistics, artificial intelligence, and psychology [1, p.1]. Software engineers may be more familiar with *design rationale*, which can be regarded as a subfamily of argumentative research focused on design objects. Lee and Lai define it, in its most general sense, as an explanation of why an artifact is designed the way it is [2, p.257].

This paper contributes to design rationale research by applying recent results from the broader field of argumentation theory, with the aim of analysing how architects explain their designs to a specialized audience. With this purpose in mind we analyse some technical explanations offered by engineers at Twitter and eBay in public interviews, blog entries, and articles. We show that the analysed examples are cases of *defeasible argumentation*. In legal and scientific reasoning, a defeasible argument is one in which the conclusion can be accepted tentatively in relation with the evidence known so far, but may need to be retracted as new evidence comes in [1, p.2]. Similarly, we find that the analysed arguments are non-deductive and may be retracted if additional knowledge is provided.

With the aim of supporting this hypothesis, we borrow what Walton et al. regard as "the most useful and widely used tool so far developed in argumentation theory": *argumentation schemes* [1, p.1]. Schemes "represent structures of common types of arguments used in everyday discourse, as well as in special contexts like those of legal argumentation and scientific argumentation" [1, p.1]. A scheme relates one or more *premises* with a *conclusion*. In the context of a dialogue, if an argument put forward by a *proponent* meets the structure of a scheme, and the premises are acceptable to a *respondent*, then the respondent is obliged to tentatively accept the conclusion.

From the case studies we infer four argumentation schemes for software architectures. Figure 1 shows the first one, *argument from scenario*; we will explain it in Section II. We reason that these schemes are defeasible by applying Hastings' method [3] of attaching a set of *critical questions* to each one [1, p.9]. In argumentation theory, if a respondent asks one of the critical questions matching a scheme and the proponent fails to offer an adequate answer, the argument defaults and the conclusion is retracted [1, p.9]. The existence of a set of critical questions which can make an argument default is enough to consider it defeasible. Verheij argues that questions that simply criticize the premises of a scheme are redundant because they merely ask whether the premises are true, which is already done implicitly when an argument is evaluated [1, p.374]. Therefore, in this paper we will not consider that specific category of critical questions.

The concept of argument has long been part of the design rationale body of research in different forms, such as *claims* in DRL [2], *justifications* in RLM [4], and *arguments* in IBIS [5], REMAP [6], and QOC [7]. The meaning closest to our paper is the one used in the Goal Argumentation Method by Jureta et al. [8] in the context of requirements engineering, who distinguish premises and conclusions in arguments. Nevertheless their approach differs from ours in that they do not employ argumentation schemes, and in that they borrow the concept of *defeasible consequence* from artificial intelligence [9, p.129], which does not consider critical questions. Another related research is the recent work by Yuan and Kelly [10] who do identify argumentation schemes and critical questions in the context of safety requirements. As an example from the field of architectures, the AREL approach [11], [12] combines

Argument from scenario
Premise 1: Scenario s is a possible behaviour of architecture A.
Premise 2: Scenario s contributes positively to/satisfies requirement R.
Conclusion: Therefore, architecture A contributes positively to/satisfies R.
Critical questions:
CQ1: Are there any exceptions to scenario s that could invalidate its contribution to R?
CQ2: Are there any other scenarios of A that could contribute negatively to/prevent R?
CQ3: Is there any other requirement R' that is negatively contributed (or even prevented) by scenario s?

Fig. 1. Argument from scenario: scheme and some critical questions

both qualitative and quantitative design rationale. The former considers, for each design issue, the following parameters: assumptions, constraints, strengths, weaknesses, trade-offs, risks, and nonrisks. The qualitative design rationale considers cost, benefit, implementation risk, and outcome certainty risk. However, the AREL approach does not address the structure of the arguments (or "assessments") for explaining why the selected decisions satisfy their associated design concerns.

We base on Lee and Lai's partitions of design rationale to organize our argumentation schemes. The authors distinguish three different (although related) uses of the phrase "design rationale" [2, p.256]: (1) a historical record of the reasons for the choice of an artifact; (2) a set of claims that would have to be true if the artifact is to be successful; and (3) a description of the design space of alternatives. Under the second meaning we categorize those arguments oriented to explain that an architecture satisfies (or contributes to) a given requirement. Section II presents two argumentation schemes in this sense. The first and third meanings can be regarded as forms of practical reasoning: the agent's goal is to evaluate an action through its consequences in order to decide whether or not to carry it out [1, p.94]. We present two schemes related with practical reasoning in Section III.

Section IV presents a discussion on the role of defeasible argumentation in architecture evaluation, based on the identified schemes and on results from other fields, particularly legal reasoning. It also addresses the question of formal reasoning in architectures, introducing a fifth argumentation scheme. Finally, the paper closes with conclusions, acknowledgements, and references.

II. Arguments for satisfaction of requirements

The simplest way of explaining why an artifact satisfies a claimed property is by describing a sample execution that achieves that property. The respondent is expected to accept that the (possibly infinite) remaining executions will similarly achieve the claimed result, or at least will not contradict it. This form of explanation is present in many fields. For example, in their account of mechanisms in Biology, Bechtel and Abrahamsen state that "people, including scientists, understand diagrams of mechanisms by animating them" [13, p.430]. In the case of software architectures, a scenario involving one or several components may be described and then claimed to satisfy a requirement. Figure 1 shows the scheme of the *argument from scenario* and some critical questions.

An example of the scheme is the explanation that Twitter Engineering offers of Blender, a component that replaced the search engine front-end in 2011 [14]. According to the authors, the new component achieved an improvement of the overall user latency by a three-times drop and halved the CPU load of the front-end servers. We can map these two goals to requirement R in Premise 2 of the scheme. How does Twitter Engineering explain that the new search architecture achieves these goals? They offer a description of a typical scenario, since the arrival of a search query to Blender until the response is returned. Figure 2 summarizes the argument as an instance of the scheme (additional details can be found in [14]). The steps correspond to the description of scenario s in Premise 1 of the scheme. After describing the scenario, the authors claim that the requirement is satisfied and generalize this result to the search architecture. The respondent can refute this conclusion by asking the critical questions shown in Figure 1, which makes the *burden of proof* [15] going back to the argument proponent.

Another typical form of explaining why an architecture satisfies a requirement is the use of an architectural tactic, such as those proposed by the Software Engineering Institute[1]. The argument begins by claiming that some architectural decision is an instance of a certain tactic. As tactics are well-known approaches to achieve quality attributes, they are used as intermediate points for arriving at the conclusion that the architecture contributes to some quality attribute. Figure 3 presents the scheme of the *argument from tactic* and two critical questions.

This scheme is employed by Randy Shoup (formerly chief architect at eBay) in his detailed explanation of the architectural decisions taken in the auction system [18], [19]. Shoup begins by introducing a number of "best practices" that contribute to scalability; each one corresponds to a tactic T in Premise 2 of Figure 3. Then he explains that the architecture of eBay includes a realization D for each one of these best practices (Premise 1); this is argued by example. The conclusion is that the whole architecture contributes positively to scalability (quality attribute Q in Premise 2). We will highlight two of

[1]In 2003, the Software Engineering Institute (SEI) published a catalogue of roughly fifty architectural tactics, categorized according to the quality attribute they contribute to [16]. The catalogue was updated in 2012 [17].

Example of argument from scenario: front-end of Twitter's search engine
Background: In Twitter's terminology, a *workflow* is a set of back-end services with dependencies between them, which must be processed to serve an incoming client request to the search engine. Dependencies between services are resolved by pre-computing the execution order of every workflow as a sequence of *batches* (services in the same batch can be called in parallel, while services in different batches must be processed in the order of the sequence). Finally, each workflow with resolved dependencies is mapped to a Netty pipeline (Netty is an asynchronous event-driven network application framework for Java; a Netty pipeline is a sequence of asynchronous handlers of input-output). Each workflow batch is associated to a pipeline handler.
Step 1: When a search query arrives, a proxy layer reads it, figures out which workflow is requested, and routes it to the appropriate pipeline. This process makes use of asynchronous events.
Step 2: At the pipeline, the service handler corresponding to the first batch of the workflow constructs the appropriate back-end requests and issue them to the servers. The input-output (I/O) thread that is processing the client query is freed when all the back-end requests have been dispatched.
Step 3: A timer thread checks every few milliseconds to see if any of the back-end responses has returned from the remote servers and sets a flag indicating if the request succeeded, timed out, or failed.
Step 4: When all the responses from the first batch have successfully arrived, they are aggregated and passed to the next batch in the workflow pipeline.
Step 5: The previous steps are repeated until the workflow is completed or its timeout has elapsed.
Step 6: The response is returned to the client.
Conclusion: "Throughout the execution of a workflow, no thread busy-waits on I/O. This allows us to efficiently use the CPU on our Blender machines and handle a large number of concurrent requests. We also save on latency as we can execute most [client] requests to back-end services in parallel" [14].

Fig. 2. An instantiation of the scheme *argument from scenario*

Argument from tactic
Premise 1: Architecture A includes design decision D which is an instance of architectural tactic T.
Premise 2: Tactic T contributes positively to quality attribute Q.
Conclusion: Therefore, architecture A contributes positively to Q.

Critical questions:
CQ1: Does architecture A contain any other design decision D' which contributes negatively to (or even prevents) quality attribute Q?
CQ2: Does architecture A realize another tactic T' which contributes negatively to (or even prevents) Q?

Fig. 3. Argument from tactic: scheme and some critical questions

these practices together with Shoup's justification by example: (1) *functional segmentation*: "we [...] have the selling systems, distinct from the buying systems, distinct from the search systems, distinct from various back-end systems, and so on" [19]; (2) *horizontal split* (applied to computations): "within the search pool [...], within the selling pool, etc, all the application servers, all the hundreds or thousands of application servers that are in that pool are all entirely equal, and each can serve the load" [19].

Functional segmentation is a specialization of the SEI's tactic "maintain semantic coherence", consisting in ensuring that all the responsibilities of an architectural module work together without excessive reliance on other modules, with the aim of having modifications localized. This tactic positively contributes to modifiability [16, p.106], and, therefore, to scalability. Horizontal split (applied to computations) corresponds to the SEI's tactic "maintain multiple copies of computations" [16, p.114]. Application servers in a pool are replicas of computations whose purpose is to reduce the contention that would occur if all computations took place on a central server. This is a resource management tactic which contributes positively to performance [16, p.114], and, in turn, to scalability. Figure 3

proposes a couple of critical questions for the argument from tactic.

III. Arguments for design decisions

This section deals with practical reasoning: the explanation of the reasons why a design decision was taken, or why a certain choice was made among a set of alternative design decisions, and the consequences of such actions. We have extracted two argumentation schemes in this category from Randy Shoup's explanation of the architecture of eBay. The first one is the *argument from compensation* (Figure 4). It is based on defending that an apparently bad design decision has been taken for achieving a greater benefit than what it could have been initially expected.

According to Shoup, data in eBay is partitioned by functional area (e.g. users, items, etc) and, within each area, data is segmented in databases according to different schemes (modulo of a key, range of identifiers, etc) [18], [19]. The CAP principle states that any networked shared-data system can have at most two of the following properties at the same time: "consistency, equivalent to having a single up-to-date copy of the data; high availability of that data (for updates); and tolerance to network partitions" [20, p.23]. Therefore, the decision of partitioning data (D_1 in Figure 4) carries a negative consequence (N in Figure 4): it prevents either high availability or immediate consistency. This corresponds to Premise 1 in the scheme. However, Shoup defends data partitioning by claiming that it enables to *independently* increase resources (hosts) as necessary (D_2 in Premise 2): "User data, for example, is currently divided over 20 hosts [...] As our numbers of users grow, and as the data we store for each user grows, we add more hosts, and subdivide the users further" [18]. The ability to independently increase the resources as necessary contributes positively to scalability (desirable property P in Premise 3). As Shoup explains it: "if you can't split it, you can't scale it [...] Regardless of the details of the partitioning scheme, though, the general idea is that an infrastructure which supports partitioning and repartitioning of data will be far more scalable than one which does not" [18]. Scalability (P in Figure 4) is "one of [eBay's] primary architectural forces [...] It colors and drives every architectural and design decision" [18]. Contributing to scalability is so important for eBay that it may mean sacrificing either immediate consistency or high availability (N in Figure 4), as Premise 4 states. Some critical questions for this argument are presented in Figure 4.

Another pattern of argumentation, *argument from weakening* (Figure 5), can be inferred by analysing Shoup's explanation to why global, immediate data consistency was given up in eBay. Consider said requirement as R in Premise 1 of the argumentation scheme. According to CAP, meeting R is incompatible with simultaneously satisfying high availability and partition tolerance (the conjunction of both requirements corresponds to P in Premise 2). Shoup proposes renouncing to R and embracing instead a spectrum of degrees, from immediate consistency (local to specific data which are non-partitioned), through eventual consistency for some other partitioned data, and no-consistency for the remaining partitioned data. The new requirement (R' in Premise 3) is a weakened version of the original requirement R. This is how Shoup explains it: "The reality of large-scale systems is that [...] [consistency is] a spectrum [...] There are some operations that need to be very highly consistent and the 100% is the absolutely appropriate number, and nothing less will do. But there are other cases –in fact, again, the majority of cases– where [consistency] doesn't need to be transactional, certainly at that moment [...]. And then there are plenty of situations [...] where consistency is [...] just not all that important. It's okay to lose some information because we got the stuff that was really important, and if somebody has to redo an operation, that's unfortunate, and we wish it didn't happen, but that's the cost and the price that we pay for having a [partitioned,] available system and a scalable system" [19]. As Shoup clearly states, it is preferable that the architecture satisfies this weakened version of consistency, provided that high availability and partitions are maintained (Premise 4). Figure 5 shows some critical questions for this argument.

IV. Argumentation schemes and architectural evaluation

Typical frameworks such as the Architecture Tradeoff Analysis Method (ATAM) [16] organize architectural evaluation as a dialogue among several participants. This is similar to argumentation theory, where argumentation is viewed as a dialogical process for making justified decisions: "the goal of the process is to clarify and decide the issues, and produce a justification of the decision which can withstand a critical evaluation by a particular audience" [15, p.239].

In argumentation theory, the concept of proof is weaker than it is in mathematics and, as in legal reasoning, it is not primarily deductive [15, p.240]. In this context, a proof is a structure which demonstrates to a particular audience that a proposition is *sufficiently* satisfied. To this aim, in the legal domain, four *proof standards* for factual issues exist in common law jurisdictions, each one denoting a different degree of sufficiency [15, p.241].

In this paper we have analysed examples of architectural explanations and showed that they are defeasible arguments. It is reasonable to assume that defeasible argumentation is also employed in typical architectural evaluations. In this context, a rich catalogue of schemes could be useful for the architect to organize his or her reasoning, while the critical questions might make him or her realize weak aspects of the argumentation –or even detect flawed design decisions. The evaluation team could employ the critical questions as a checklist of issues that need to be addressed by the architect; if no acceptable response is provided, the architect's argument would be retracted and the issue would be treated as an architectural risk. Empirical studies suggest that architects tend to focus on the reasons that justify a design over those that explain why the design might have negative issues [21], [22]. The employment of critical questions in argumentation could greatly help to change this attitude.

Argument from compensation
Premise 1: Architectural decision D_1 carries undesirable property N. *Premise 2*: D_1 enables the application of architectural decision D_2. *Premise 3*: D_2 achieves desirable property P, or contributes positively to P. *Premise 4*: It is preferable meeting P and N over not satisfying P. *Conclusion*: Therefore, I ought (practically speaking) to carry out decision D_1.
Critical questions: CQ1: Does decision D_1 carry negative properties other than N? CQ2: Does decision D_2 carry negative properties that have not been considered? CQ3: Is there a design decision D_3 that achieves P without the need of introducing an undesirable property in the architecture? CQ4: Is the contribution of D_2 to P enough to justify the introduction of N?

Fig. 4. Argument from compensation: scheme and some critical questions

Argument from weakening
Premise 1: R is a requirement of architecture A. *Premise 2*: There exists another requirement P of architecture A such that satisfying $R \wedge P$ is impossible (or unattainable with the current resources). *Premise 3*: There exists a weakening R' of R such that $R' \wedge P$ is attainable. *Premise 4*: It is preferable that the architecture satisfies $R' \wedge P$ over meeting $\neg R \wedge P$ and over meeting $R \wedge \neg P$. *Conclusion*: Therefore, I ought (practically speaking) to give up R and adopt $R' \wedge P$.
Critical questions: CQ1: (In case that $R \wedge P$ is unattainable with the current resources) Can the resources be redistributed (or increased) in order to make $R \wedge P$ attainable? CQ2: Have other weakening options for R been considered such that they meet Premises 3 and 4? CQ3: What criterion has been employed to decide to weaken R instead of P? Have alternative criteria, which might suggest a different choice, been considered?

Fig. 5. Argument from weakening: scheme and some critical questions

Argument from formal model
Premise 1: A correspondence has been established between architecture A and some formal model F. *Premise 2*: A correspondence has been established between F and some classes of real-world phenomena (events, stimuli, etc) that interact with A. *Premise 3*: A formal process has proved that F entails some predicate P. *Premise 4*: A correspondence has been established between P and some property R of architecture A. *Conclusion*: Therefore, architecture A satisfies property R.
Critical questions: CQ1: Does architecture A satisfy the constraints for using model F? CQ2: Do the classes of phenomena satisfy the assumptions for using model F? CQ3: Are the inputs to model F speculative? CQ4: Is model F logically and/or mathematically sound? CQ5: Is property R satisfied by systems based on architectures similar to A? CQ6: Has architecture A been simulated to determine whether property R is consistent with the simulation results?

Fig. 6. Argument from formal model: scheme and some critical questions

As in legal reasoning, defeasible arguments about architectures can only aspire to *convince* the evaluation team. In the ATAM, "the goal is for the evaluation team to be *convinced* that the instantiation of the approach is appropriate for meeting the attribute-specific requirements for which it is intended" (our italics) [16, p.282]. Convincement of the audience has also been employed in requirements engineering: Jackson's *correctness arguments* are intended to convince a

customer that a system specification, together with an analysis of the problem context, satisfies its functional requirements [23]; analogously, Haley et al. employ *satisfaction arguments* to convince a reader that a system can satisfy the security requirements laid upon it [24].

Besides convicement, there is also room for formal reasoning in architectural evaluation. A typical case happens when the architect employs a formalism for building a *model* of the architecture or some part of it [25], [26]. A formal procedure is employed to prove some predicate in the model, which is then interpreted in the architecture. Nevertheless, it is important to be cautious at this point: while a formal process can prove that a formal model satisfies some predicate, the broader context in which the model is built and interpreted may constitute a defeasible argument. This consideration is pointed out by Bass et al. [25] in their account on *reasoning frameworks* for architecture evaluation. A reasoning framework is a structure which among other components includes an analytic theory, a model of the architecture (based on the analytic theory), and an evaluation (proof) procedure. The authors clearly refer to the defeasible nature of the employment of a model when they claim that "to have any confidence that appropriate [architectural] decisions are being made, however, engineers must know how much they can trust a reasoning framework's results" [25, p.17]. To this aim they suggest some criteria for certifying the *accuracy degree* of the inputs to a reasoning framework, as well as some guidelines for validating the obtained predictions.

As an example, Rate Monotonic Analysis (RMA) is an analytic theory (in the sense of [25]) that can be used to reason about worst-case latency. This formalism considers that all event arrivals are periodic [26, p.162]. Besides, RMA tasks have a number of constraints; for example, it is assumed that execution times have little or no variability and that fixed priority scheduling is being used [25, p.8]. If RMA is employed for modelling some behavioural view of an architecture, and both the architecture and the real-world domains in the problem context do not completely match these constraints and assumptions, the confidence degree on the model predictions may be reduced. For a more detailed account on the relationship between formal models and software artifacts, the reader may be interested our research on the topic [27].

Figure 6 shows an argumentation scheme, *argument from formal model*, where formal reasoning is put in the broader context of determining whether an architecture satisfies a certain property. For elaborating the critical questions of this scheme we have based on the certification and evaluation criteria of reasoning frameworks proposed by Bass et al. [25, pp.17-18].

Another typical situation of formal reasoning happens when a formal procedure is employed to prove some property of the architecture itself (instead of proving it on a model). This requires the architecture to be defined with a formal language. However, any formal architecture includes assumptions for the real-world phenomena with which it interacts; for example, the Wright architecture description language [28] considers that events are instantaneous. Assumptions on phenomena may be more or less similar to reality, which may make the conclusions of the proof procedure more or less reliable. A possible critical question could be: are the assumptions about real-world phenomena relevant for the obtained conclusion?

V. Conclusions

Argumentation is understood in the context of a dialogue among participants for making justified decisions [15]. This paper has shown that this is also the case with architectural evaluation. In the same way as styles and tactics are useful tools for designing software architectures, we have given reasons in favour of argumentation schemes as instruments of great help during architectural evaluation, both for architects and the evaluation team. Four defeasible argumentation schemes have been inferred from several explanations of engineers at Twitter and eBay, and an additional one has been described for showing that reasoning with formal models is actually one part of a larger argument which is also defeasible. Future works include enriching the catalogue of schemes as well as studying (conflicting) relationships among arguments.

Acknowledgements

This work has been partially funded by "V Plan Propio de Investigación de la Universidad de Sevilla (VPPI-US)."

References

[1] D. Walton, C. Reed, and F. Macagno, *Argumentation Schemes*. Cambridge University Press, 2008.

[2] J. Lee and K.-Y. Lai, "What's in design rationale?" *Human-Computer Interaction – Special Issue on Design Rationale*, vol. 6, pp. 251–280, 1991.

[3] A. C. Hastings, *A Reformulation of the Modes of Reasoning in Argumentation. PhD dissertation*. Northwestern University, 1963.

[4] P. Louridas and P. Loucopoulos, "A generic model for reflective design," *ACM Transactions on Software Engineering and Methodology*, vol. 9, no. 2, pp. 199–237, 2000.

[5] J. Conklin and M. L. Begeman, "gibis: a hypertext tool for exploratory policy discussion," *ACM Transactions on Information Systems*, vol. 6, no. 4, pp. 303–331, 1988.

[6] B. Ramesh and V. Dhar, "Supporting systems development by capturing deliberations during requirements engineering," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 498–510, 1992.

[7] A. Maclean, R. M. Young, V. M. E. Belotti, and T. P. Moran, "Questions, options, and criteria: elements of design space analysis," *Human-Computer Interaction*, vol. 6, no. 3, pp. 201–250, 1991.

[8] I. J. Jureta, S. Faulkner, and P.-Y. Schobbens, "Argument schemes in computer system safety engineering," *Clear justification of modeling decisions for goal-oriented requirements engineering*, vol. 13, pp. 87–115, 2008.

[9] G. R. Simari and R. P. Loui, "A mathematical treatment of defeasible reasoning and its implementation," *Artificial Intelligence*, vol. 53, pp. 125–157, 1992.

[10] T. Yuan and T. Kelly, "Argument schemes in computer system safety engineering," *Informal Logic*, vol. 31, no. 2, pp. 89–109, 2011.

[11] Y. J. A. Tang and J. Han, "A rationale-based architecture model for design traceability and reasoning," *The Journal of Systems and Software*, vol. 80, pp. 918–934, 2007.

[12] A. Tang, J. Han, , and R. Vasa, "Software architecture design reasoning: a case for improved methodology support," *IEEE Software*, vol. 26, no. 2, pp. 43–49, 2009.

[13] W. Bechtel and A. Abrahamsen, "Explanation: a mechanist alternative," *Studies in History and Philosophy of Biological and Biomedical Sciences*, vol. 36, pp. 421–441, 2005.

[14] Twitter-Engineering, "Twitter search is now 3x faster," *Twitter Engineering Blog, entry of April 6, 2011, available at http://engineering.twitter.com*, Apr. 2011.
[15] T. F. Gordon and D. Walton, "Proof burdens and standards," in *Argumentation in Artificial Intelligence*, I. Rahwan and G. R. Simari, Eds. Springer Science + Business Media, LLC, 2009, pp. 239–258.
[16] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice. Second edition.* Addison-Wesley, 2003.
[17] ——, *Software architecture in practice. Third edition.* Addison-Wesley, 2012.
[18] R. Shoup, "Scalability best practices: Lessons from ebay, available at http://www.infoq.com/articles/ebay-scalability-best-practices," *InfoQueue*, 2008.
[19] ——, "Transcript 109: ebay's architecture principles with randy shoup, available at http://www.se-radio.net/2008/09/episode-109-ebays-architecture-principles-with-randy-shoup/," *Software Engineering Radio*, 2008.
[20] E. Brewer, "CAP twelve years later: how the "rules" have changed," *IEEE Computer*, vol. 45, no. 2, pp. 23–29, Feb. 2012.
[21] A. Tang, M. A. Babar, I. Gorton, and J. Han, "A survey of architecture design rationale," *The Journal of Systems and Software*, vol. 79, pp. 1792–1804, 2006.
[22] M. Keil, H. J. Smith, S. Pawlowski, and L. Jin, "'Why didn't somebody tell me?': climate, information asymmetry, and bad news about troubled projects," *SIGMIS Database*, vol. 35, no. 2, pp. 65–84, 2004.
[23] M. A. Jackson, *Problem Frames: Analyzing and Structuring Software Development Problems*. Addison-Wesley, 2001.
[24] C. B. Haley, J. D. Moffett, R. Laney, and B. Nuseibeh, "Arguing security: validating security requirements using structured argumentation," in *Proceedings of the 3rd Symposium on Requirements Engineering for Information Security*, 2005.
[25] L. Bass, J. Ivers, M. Klein, and P. Merson, *Reasoning Frameworks. Technical Report CMU/SEI-2005-TR-007*. Carnegie Mellon Software Engineering Institute, 2005.
[26] F. Bachmann, L. Bass, M. Klein, and C. Shelton, "Designing software architectures to achieve quality attribute requirements," *IEE Proceedings - Software*, vol. 152, no. 4, pp. 153–165, 2005.
[27] J. M. Cañete-Valdeón, "On the interpretation of mathematical entities in the formalisation of programming and modelling languages," *Mathematical Structures in Computer Science*, vol. 18, no. 6, pp. 1017–1030, 2008.
[28] R. Allen, R. Douence, and D. Garlan, "Specifying and analyzing dynamic software architectures," in *Proceedings of the First International Conference on Fundamental Approaches to Software Engineering (FASE'98)*, 1998.