Characterizing Architecturally Significant Requirements

Lianping Chen, University of Limerick and Paddy Power PLC

Muhammad Ali Babar, Lancaster University and IT University of Copenhagen

Bashar Nuseibeh, University of Limerick and The Open University

// A new framework characterizes architecturally significant requirements on the basis of an empirical study of 90 practitioners from organizations of various sizes and domains. //

AT THE HEART of any engineering discipline is the interplay between problem and solution development. In software engineering, we determine a software solution's effectiveness with respect to a problem, yet the nature of the problem and its scope could depend on what solutions already exist or what solutions are plausible and cost-effective. This iterative and concurrent development and trade-off between software problems and solutions characterize the twin peaks model of software development.[1]

For the development of many large-scale, software-intensive systems, the software architecture is a fundamental part of a software solution.[2] It significantly affects software quality and cost, but not all of a system's requirements affect software architecture equally.[3] In many cases, only some requirements actually determine and shape a software architecture;[4] we call these *architecturally significant requirements* (ASRs). If ASRs are wrong, incomplete, inaccurate, or lack details, then a software architecture based on them is also likely to contain errors.

Unfortunately, in practice, stakeholders and requirements engineers frequently fail to express or effectively communicate ASRs to the architects, preventing the architects from making informed design decisions.[4] This is partly due to a lack of an authoritative, evidence-based discourse characterizing ASRs.

To address this, we carried out an empirical study to help characterize ASRs. Based on the study's results, this article presents an evidence-based framework for systematically characterizing ASRs. We hope our findings can help practitioners better understand and deal with ASRs, as well as provide researchers with a framework for discussing and conducting further research on ASRs. The work can also inform researchers' development of technologies for dealing with ASRs. Finally, our findings can enrich understanding of requirements and architecture interactions, allowing the twin peaks to move from aspiration to reality.

Methodology and Research Design

As our research method, we used grounded theory (GT),[5] which provides a systematic process of generating theory from data. GT has gained popularity among software engineering researchers, who have reported its effectiveness for topics that lack empirical research.[6] GT recommends avoiding the formulation of research questions upfront, promoting instead the selection of a general area of interest for research. We chose to explore ASRs as an area of research.

We collected the data in our study by conducting interviews with

0740-7459/13/$31.00 © 2013 IEEE

90 practitioners, who we recruited through personal connections and social networks. We conducted the interviews mostly via email and sometimes via phone. In instances when we used email for data collection, participants didn't have to find a sufficient chunk of free time to fit researchers' schedules, but could answer in their own time; this also provided participants with more time for reflective answers. Figure 1 summarizes the participants' demographics.

The participants came from four countries (58 percent from the US, 28 percent from India, 13 percent from the Netherlands, and 1 percent from the UK), as shown in Figure 1a. Figure 1b shows that the participants represented a diverse set of industries. Each participant had worked for an average of seven organizations during his or her career. Cumulatively, the participants had worked for more than 500 distinct organizations throughout their careers. The size of the organizations they worked for ranged from very small (fewer than 10 people) to very large (more than 10,000 people), as shown in Figure 1c.

Given our research's focus on understanding requirements' effects on architectures, we decided to limit participation to be only from those professionals who had experience working with software architectures in their current or previous roles. Although the majority of participants we interviewed were software architects, some were executives, managers, consultants, or developers. None of our participants specifically identified themselves as requirements engineers, but we observed that consultants often played such roles. As Figure 1d shows, 52 percent of our participants were architects, followed by 20 percent executives, 14 percent managers, 8 percent consultants, and 6 percent developers. Together, the participants had more than 1,448 years of accumulated work experience in software development and 761 years in software architecture, as shown by Figure 1e.

We conducted our interviews via informal conversations. Most of the conversations kicked off with the following question: "From your observations and experiences, what are the characteristics that distinguish ASRs from other requirements? In other words, what, if anything, makes them unique or peculiar?" Follow-up questions sought to clarify or gather more details about the characteristics mentioned by the participants.

For analyzing the data, we used some of GT's qualitative data analysis techniques such as open coding, constant comparison method, selective coding, and memoing.[7] We began data analysis as soon as we collected some data and continued iteratively and in parallel. We used a tool called NVivo to facilitate the analysis.

Only concepts that were mentioned by at least two participants were included in the final findings. Once concepts earned their way into the theory, we did not discriminate between them based on their frequencies; rather, we focused on the logical relationships among concepts, as recommended by GT.

Characteristics of Architecturally Significant Requirements

A framework of characteristics of ASRs emerged from our analysis. Figure 2 shows the framework, which consists of four sets of characteristics: definition, descriptions, indicators, and heuristics.

Definition

ASRs are those requirements that have a measurable impact on a software system's architecture. Essentially, this definition delimits a portion of requirements, the portion that affects the architecture of a system in measurably identifiable ways. Do such requirements really exist? Our empirical data appears to confirm this—for example, participants didn't perceive a requirement stating that "temperature should be displayed in Celsius not Fahrenheit on this webpage" to be architecturally significant, but they did usually perceive a requirement stating that "the system should provide five nines (99.999 percent) availability" as architecturally significant.

"Significant" is a key term in our definition. What does it mean? Ultimately, it is measured by high cost of change. This cost can be monetary or not (for example, time, resources, reputation, and opportunity cost). What cost is considered as high? The answer is invariably project-specific—a US$10,000 cost can be high for a small budget project but low for a large one. Some cost measures can be entirely qualitative but still identifiable in their ability to distinguish ASRs.

> What cost is considered as high? The answer is invariably project-specific.

Descriptions

ASRs are often hard to define and articulate, tend to be expressed vaguely, tend

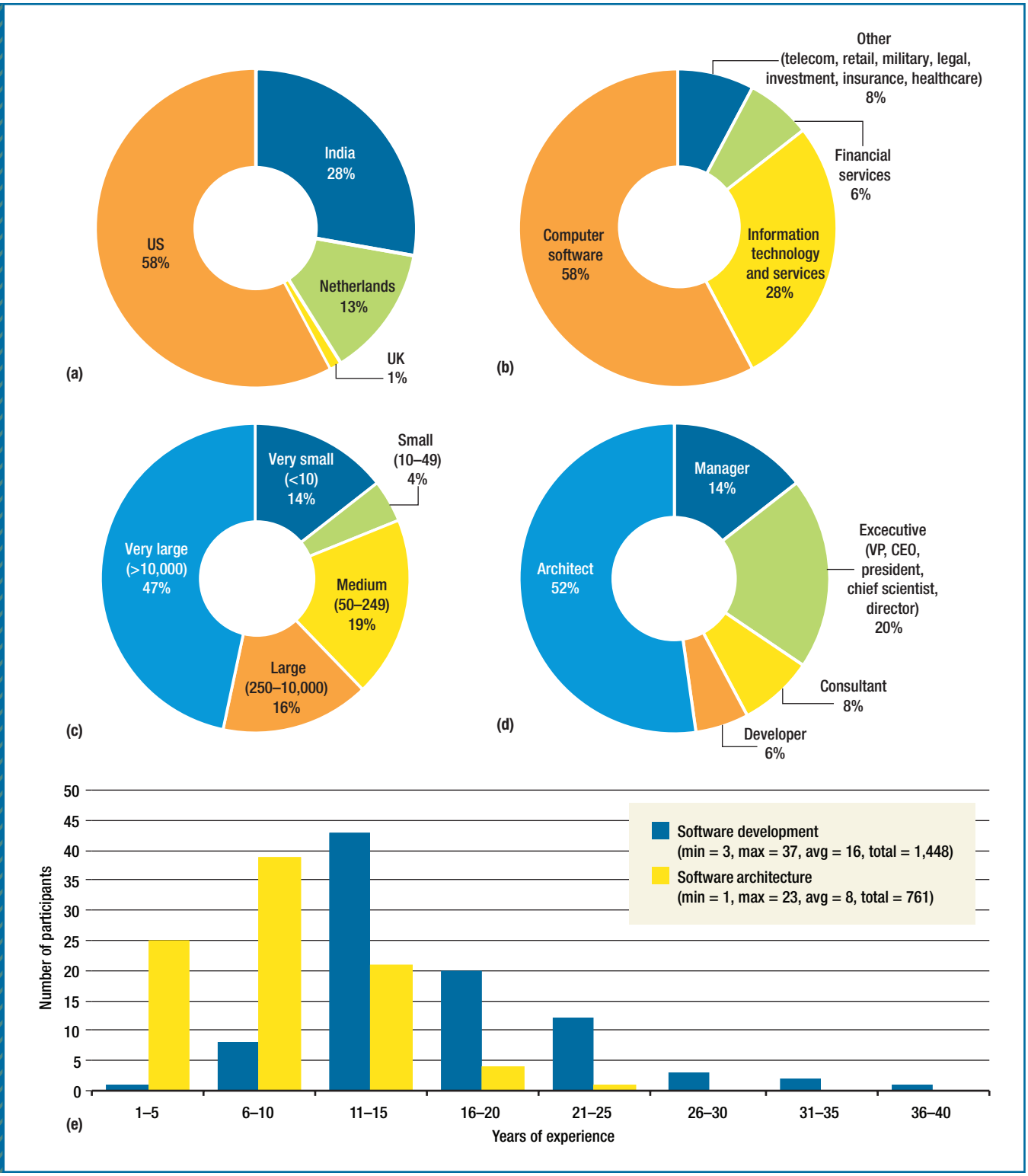

FIGURE 1. Demographics of study participants: (a) distribution over countries, (b) distribution over industries, (c) distribution over organization sizes, (d) job positions, and (e) experience in software development and architecture.

to be initially neglected, tend to be hidden within other requirements, and are subjective, variable, and situational. We could argue that other requirements also demonstrate these *descriptive characteristics*, or *descriptions*. However, ASRs' architectural significance made these characteristics' manifestations unique and challenging for ASRs.

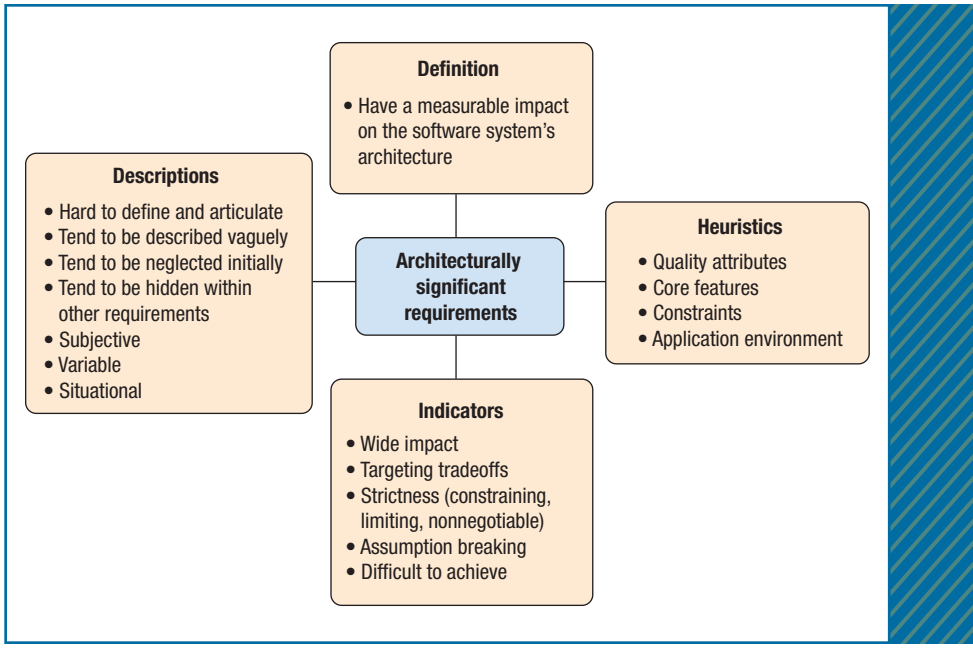

FIGURE 2. A framework of the characteristics of architecturally significant requirements.

Hard to define and articulate. One participant noted, "Users usually find it difficult to articulate [ASRs], as many of them are about abstract and general concepts."

In addition, ASRs are expected to be ready in the early stage of the software development process. At this early stage, customers might not yet be clear about their exact needs, and some requirements decisions (especially about details) might not have been made. This adds to the difficulties of defining and articulating ASRs.

Tend to be described vaguely. Many architects reported that ASRs they received are too vague to be used for making informed architectural decisions. Vaguely described ASRs often lead to bad decisions because architects can make wrong assumptions about the missing details.

Consider an example of users requesting the ability to receive notification about cash flows. Architects assumed that emails would be acceptable for this. During detailed design, users explained that they wanted real-time notification, the ability to subscribe to different account topics, and a user interface that shows all of this. This required publish-subscribe, a different architecture style.

Tend to be neglected initially. One participant said, "Typically [ASRs] are overlooked in the early phase of a project." ASRs are often neglected initially because of a lack of initial awareness of their significant effects on architecture, and thus aren't given sufficient attention. One participant said, "Users do not typically have a good understanding of [ASRs]; people who conduct requirement analysis often do not document them properly. … Users will not ask for them unless you are dealing with a highly tech-savvy group of users." Another participant said that neglecting ASRs in this way often happens with less-experienced teams.

Many times, requirements aren't recognized as architecturally significant until they've incurred a high cost, and at that stage, rectifying mistakes can be costly.

Tend to be hidden within other requirements. Following the way people spontaneously express requirements, ASRs usually aren't emphasized in requirements' oral or written descriptions; rather, they're embedded in other requirements' descriptions. Short phrases such as "highly available system of 99.999 percent uptime" or "fault tolerant" are often only mentioned briefly while describing other requirements, but these phrases can significantly affect architecture.

When architects receive ASRs that are hidden within other requirements, the ASRs usually aren't sufficiently elaborated and often lack the key details required for making informed architectural decisions.

Subjective. ASRs tend to be requested subjectively based on opinion instead of fact-based objective decisions. One participant reported, "[ASRs requested by customers] usually contain subjectivity—for example, 'The system should be available for 24/7'—whether it [needs to] be or not."

This subjectivity can lead to inaccuracy of the requirements conveyed to architects. Small differences in ASRs can lead to big differences in resulting architectures.

Variable. ASRs can change over time. This is usually unavoidable owing to

changes that take place in business and technology. As a participant reported, "Requirements always change over time, during the project and afterward." Another participant reported, "Technology as defined as hardware, software, and the relatively new inclusion of user experience is now changing so quickly that companies are viewing products as having a very limited (and short) lifetime before needing to be redesigned to take advantage of those changes."

ASRs can also vary over space—for example, consider the variability of ASRs for different but similar software systems that are engineered using a software product line paradigm.[8]

Situational. Whether a piece of requirement is architecturally significant depends on situations. As one participant said, "A requirement is architecturally significant in one case, while being 'just a requirement' in the other case."

Requirements can be situational regarding an existing architecture, a project's context or scope, and so on. A requirement might be architecturally significant in a situation where a "bad" architecture is in place, but is not so when the architecture is "good." Another participant reported, "A simple requirement in a smaller project might not be architecturally significant. Take the same requirement and enhance scope, add multiple interactions—then it could become significant."

A requirement's architectural significance can depend on many factors. So, a definitive judgment on a requirement's architectural significance usually can only be made when the requirement really incurs a high cost or if the architects require it to make architectural decisions. During requirements gathering, we can only say that certain requirements are likely to be architecturally significant. Finding a definitive list of ASRs is not feasible. ASRs need to be dealt with individually.

Indicators

Although the cost of change is a measure of significance, getting an accurate cost of a requirement is challenging (despite extensive studies[9]). In addition, an estimate of whether a requirement is architecturally significant can be important for guiding the gathering of ASRs, but cost estimation is usually undertaken after requirements have been gathered.

ASRs do have some characteristics that help us distinguish them from other requirements without needing to undertake a full cost estimation. Indicative characteristics, or indicators, provide such pragmatic hints about the architectural significance of a requirement.

Although the cost of change is a measure of significance, getting an accurate cost of a requirement is challenging.

Wide impact. When a requirement has wide impact on a system, it's usually architecturally significant. This can be in terms of the components, other requirements, the code modules, the stakeholders, or other things that are affected by the requirement. Participants noted, "Yes, there are requirements [that] can be thought of as architecturally significant because those requirements impact the system as a whole immediately"; "[An ASR] has widespread impact across multiple components of the system"; and "The more broadly a requirement and its resolution can be applied, the more significant it is."

Targeting trade-offs. A requirement that targets trade-off points is usually architecturally significant. A trade-off point is one at which there's no solution that satisfies all involved requirements equally well, therefore architects must select a design option that compromises some requirements to meet others. Such requirements can directly affect architecture decisions' outcome, thus they're architecturally significant. One participant reported this as follows: "The trade-offs are the weak points—the raw nerves—of the architecture. If a new requirement happens to (unintentionally) target these trade-offs—these weak points—then the probability of it becoming architecturally significant is higher than if it did not target these trade-offs."

When the requirement targets a trade-off point, the details, accuracy, and precision of the requirement description become important. One participant reported that half-a-second difference in his system's performance requirements can lead to a totally different design.

Strictness (constraining, limiting, non-negotiable). A strict requirement necessitates a particular design option, and the requirement itself can't be negotiated. When making architectural decisions, the requirements that can be satisfied by multiple design options (or can be negotiated to a form that can be satisfied by multiple design options) will allow flexibility in architectural design, whereas a strict requirement will determine architectural decisions because it can't be satisfied by alternative design options (thus being constraining or limiting). One participant reported, "Our

significant requirements were those that would be the limiting, or defining, characteristics of the product."

Assumption breaking. When designing a system's architecture, the architect makes some fundamental assumptions (explicitly or implicitly). For example, architects assume that turning down the server at midnight for maintenance is acceptable. Later on, this might no longer be acceptable if the business expands to countries in different time zones.

When a requirement breaks any of these assumptions, it's architecturally significant. This is because it requires architectural change to accommodate the requirement. One participant reported, "Down the line someday, we meet our assumptions face to face, and a requirement that actually crosses the boundary of that assumption would be the one that changes the architecture."

Judging whether a requirement breaks any such fundamental assumptions requires good knowledge about the system's existing architecture. A well-organized record of the assumptions and related architectural decisions can make the job easier, so there's a need for effective tools that can manage assumptions and design decisions.

Difficult to achieve. If a requirement is difficult to meet or is technologically challenging, it's likely to be architecturally significant. One participant reported, "The uniqueness of [ASRs] to me is … difficulty of achieving. For example, latency is very hard to achieve if not thought [about] early on."

Heuristics

Judging whether a requirement has wide impact, whether a requirement targets trade-offs, whether a requirement strictly requires a particular design option, whether a requirement breaks existing architectural assumptions, or whether a requirement is difficult to achieve can require substantial knowledge of the solution space.

Requirements engineers, however, aren't usually expected to have extensive solution space knowledge. Thus, many of them are unlikely to be able to use such indicators to identify ASRs. Therefore, we need characteristics that are familiar to requirements engineers. We discovered a set of such characteristics, which we call *heuristic characteristics*, or *heuristics*.

Heuristics can guide requirements engineers to ask questions proactively regarding concerns that are likely to be architecturally significant but that users don't mention.

Quality attributes. When a requirement specifies a software system's quality attributes, it's usually architecturally significant. The participants mentioned a variety of such quality attributes from the software systems that they worked on (see Figure 3).

Quality attributes' specific meanings can vary from project to project, so we must also gather details of quality attributes—for example, a general statement that "the system should respond in real time" probably won't be enough to make correct architectural decisions.

Core features. A requirement that refers to a software system's core features is likely architecturally significant. Core features define the problems the software is trying to solve; they usually capture the essence of the software system's behavior and describe the core expectations users have on the software system. They directly serve to achieve the objective of building the system.

These requirements are usually assumed, often implicitly, to be the invariants of the software system. They are part of the fundamental assumptions that the architecture is built upon. Based on a software system's core features, the most relevant quality attributes often choose themselves; as one participant reported, "Based on the unique functional aspects a software system is targeted to address, the nonfunctional ones often align themselves. … A high-frequency quantitative trading engine with millisecond latency will automatically emphasize performance, whereas an airline control system will immediately focus on reliability."

Adaptability	Extensibility
Availability	Modularity
Configurability	Portability
Flexibility	Reusability
Interoperability	Testability
Performance	Auditability
Reliability	Maintainability
Responsiveness	Manageability
Recoverability	Sustainability
Scalability	Supportability
Stability	Usability
Security	

FIGURE 3. Software systems' quality attributes that participants mentioned during the study.

Constraints. Requirements that impose constraints on a software system are usually architecturally significant. These can be nontechnical—such as financial, time, and developer skill constraints—or technical—such as constraints imposed by existing architectural decisions or by a client's technical decisions.

Application environment. Requirements that define the environment in which the software system will run are likely to be architecturally significant. Participants mentioned examples such as application servers, the Internet, corporate networks, embedded hardware, virtual machines, mobile devices, and so on. Systems running in different environments often have vastly different architectures.

Implication on Practice

In this article, we made a distinction between requirements that have significant impact on software architecture and other requirements. This distinction has helped us focus on ASRs when dealing with interplay between requirements and architecture.

ASRs can be challenging to deal with, as revealed by our descriptive characteristics. Descriptive characteristics explain why ASRs are challenging to handle. They can also be useful input for developing evaluation criteria for approaches, tools, and practices to deal with ASRs.

In practice, there is no label for each requirement to tell us whether or not it is architecturally significant. To make the distinction practical, we need pragmatic characteristics to enable us to identify ASRs. Indicative characteristics provide such pragmatic hints about the architectural significance of a requirement. They rely on some knowledge of the solution space.

Heuristic characteristics highlight requirements that tend to be architecturally significant, using terminology familiar to people in the problem space. In contrast to indicators, they can be used by people who do not have sufficient knowledge of the solution space. They can guide requirements engineers to ask questions proactively, on concerns that are likely to be architecturally significant but that users do not mention. These characteristics have the potential to bring substantial improvements to the gathering of ASRs by identifying those ASRs that might otherwise be ignored.

The twin peaks model suggests that requirements and architecture should be treated iteratively and concurrently. Our observations of the situational nature of ASRs suggest that such iteration is inevitable. Some requirements are only recognized as architecturally significant after architects realize they need them to make architectural decisions. At this point, architects usually need to ask requirements engineers to provide them.

Our findings also reveal another interaction between requirements and architectures: providing feedback on an architecture's impact on requirements to inform requirements decisions and to avoid committing to infeasible requirements or requirements that cost more than the value they can generate.

The concrete manifestation of this finding can be in the form of improved requirements negotiation—for example, if a requirement targets a trade-off point but doesn't bring sufficient value to justify the cost incurred by the corresponding design option, one can negotiate the requirement with the users to make the requirement less demanding or drop it altogether.

> In practice, there is no label for each requirement to tell us whether or not it is architecturally significant.

This also poses challenges to what used to be traditional thinking that one should not consider solution space concerns while gathering requirements. This can be true for requirements that are not architecturally significant. However, for ASRs, architects' feedback about requirements' impact can help with better-informed requirements decisions. Thus, we suggest that appropriate use of solution space knowledge can help gather and negotiate ASRs more effectively.

Indicative characteristics also suggest that there should be a closer collaboration between requirements engineers and software architects. Development teams, processes, and practices should be organized and designed in a way that encourages and facilitates such collaboration, with corresponding tools that encourage exchange rather than separation of development activities.

Although it is perhaps controversial, it does appear that there is a case to be made for requirements engineers having better knowledge of software architectural concerns. We suggest that such knowledge could enable them to gather and negotiate ASRs more effectively. They can ask questions of users more authoritatively during requirements elicitation. However, we are not arguing that solutions could determine problem analysis, but rather inform it and ensure that implications of requirements on architecture (and vice versa) are understood and communicated.[10]

Acknowledgments

We thank our study participants for their contributions and this article's reviewers for their prompt and thoughtful comments. This work was funded in part by SFI Lero grant 10/CE/I1855 and ERC Advanced Grant 291652.

References

1. B. Nuseibeh, "Weaving Together Requirements and Architectures," *Computer*, vol. 34, no. 3, 2001, pp. 115–117.
2. R.N. Taylor, N. Medvidovic, and E.M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*, Wiley, 2009.
3. C. Hofmeister et al., "A General Model of Software Architecture Design Derived from Five Industrial Approaches," *J. Systems and Software*, vol. 80, no. 1, 2007, pp. 106–126.

4. P. Clements and L. Bass, *Relating Business Goals to Architecturally Significant Requirements for Software Systems*, tech. note CMU/SEI-2010-TN-018, Software Eng. Inst., Carnegie Mellon Univ., 2010.
5. B. Glaser and A. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine Transaction, 1967.
6. S. Adolph, W. Hall, and P. Kruchten, "Using Grounded Theory to Study the Experience of Software Development," *Empirical Software Eng.*, vol. 16, no. 4, 2011, pp. 487–513.
7. J. Corbin and A. Strauss, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, 3rd ed., Sage, 2007.
8. M.A. Babar, L. Chen, and F. Shull, "Managing Variability in Software Product Lines," *IEEE Software*, vol. 27, no. 3, 2010, pp. 89–91, 94.
9. M. Jorgensen and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies," *IEEE Trans. Software Eng.*, vol. 33, no. 1, 2007, pp. 33–53.
10. L. Rapanotti et al., "Architecture-Driven Problem Decomposition," *Proc. 12th IEEE Int'l Requirements Eng. Conf.*, IEEE, 2003; doi: 10.1109/ICRE.2004.1335666.

ABOUT THE AUTHORS

LIANPING CHEN is a doctoral researcher at Lero—the Irish Software Engineering Research Centre at the University of Limerick and a senior software engineer at Paddy Power PLC. His research interests include software requirements and architecture, and software product lines. Chen received an MS in software engineering from Northwestern Polytechnical University. Contact him at lianping.chen@lero.ie.

MUHAMMAD ALI BABAR is a reader in software engineering at Lancaster University and an associate professor at IT University of Copenhagen, Denmark. His research interests include software architecture, cloud computing, and software development paradigms. Ali Babar received a PhD in computer science and engineering from the University of New South Wales. Contact him at malibaba@itu.dk.

BASHAR NUSEIBEH is a professor of computing at The Open University, UK, and a professor of software engineering at Lero—the Irish Software Engineering Research Centre at the University of Limerick. His research interests include software requirements and design, security and privacy, and technology transfer. Nuseibeh received a PhD in software engineering from Imperial College London. He currently serves as editor in chief of *IEEE Transactions on Software Engineering* and holds a European Research Council Advanced Grant and a Royal Society-Wolfson Merit Award. Contact him at b.nuseibeh@open.ac.uk.