Bridging the Twin Peaks: the Case of the Software Industry

Garm Lucassen, Fabiano Dalpiaz, Jan Martijn van der Werf and Sjaak Brinkkemper
Department of Information and Computing Sciences
Utrecht University
Email: {g.lucassen, f.dalpiaz, j.m.e.m.vanderwerf, s.brinkkemper}@uu.nl

Abstract—**We review the relationship between software architecture and requirements in the context of software products. Based on empirical evidence from a comparative case study, we promote four positions: (1) the requirements/architecture alignment problem for software products is inherently different than the same problem for tailor-made software; (2) bridging the Twin Peaks corresponds to defining and enacting a stepwise evolution of the product architecture; (3) communication tasks are ascribed to the product manager rather than the architect; and (4) integrated and cross-disciplinary tools are key to maintain requirements/architecture alignment. We argue that these positions motivate and characterize future research in the field.**

I. Introduction

Understanding the complex interplay between requirements engineering (RE) and software architecture (SA), and ensuring the alignment between these disciplines, is a longstanding concern in software engineering [1], [2]. The active debate throughout the Twin Peaks workshop series [3] shows the contemporary relevance of the topic.

We focus on a specific case of the Twin Peaks problem[1]: software products [4]. Unlike tailor-made software systems, software products are designed, built, and maintained by a software producing organization (SPO) that aims to reuse or adapt the same product for *multiple*, *varying* customers [5], [6]. The large number of stakeholders continuously generate requirements [7], forcing the SPO to select specific requirements whilst taking into account the (limited) availability of financial and human resources and intense time pressure [8]. These aspects have a profound impact on architectural considerations, making the alignment challenge for software products inherently different.

We agree with the claims that requirements/architecture alignment is essential for software product success [9], [10]. However, we also acknowledge that SPOs are immature, for they have insufficient available knowledge on *how* to achieve the alignment and whether the costs *justify* the benefits [6]. The documentation that SPOs maintain on software architectures is scarce from day one [6], [11]. The absence of clear and unambiguous architectural diagrams and documentation makes alignment or co-evolution according to conventional approaches difficult. The few available approaches on the intersection of SA and RE are unsable for the Twin Peaks problem for software products. They focus on generating software architecture from requirements [12] or apply Twin Peaks in the context of Software Product Lines [13], which fails to consider product software without a line architecture.

To mitigate this problem, we extend our previous work [12] by means of comparative case study research, and propose the Reciprocal Twin Peaks model. This is a conceptual framework that defines *how* product managers and software architects can effectively collaborate in product software development through the exchange of concrete information artifacts.

This model sheds light on the nature of the Twin Peaks problem for software products, and paves the way for the creation of artifacts, methods, and tools that help SPOs bridge the Twin Peaks. The cost savings and improved product quality we expect these instrumentations to deliver, will ultimately *justify* the investment of pursuing alignment.

We present four positions that are interspersed throughout the paper. We first elaborate on the distinctness of the alignment problem for software products in Section II, leading to **Position 1**. We introduce the Reciprocal Twin Peaks model to explain how the problem should be solved in Section III, leading to **Position 2**. Next, we discuss the main findings of our empirical evaluation of a preliminary version of the model in Section IV, justifying **Position 3** on communication responsibilities. Finally, we draw conclusions and present future research opportunities in Section V, emphasizing the need of integrated and cross-disciplinary tooling in **Position 4**.

II. The Alignment Problem for Software Products

The importance of involving software architects in software product management (SPM) is frequently discussed in literature. Lindgren et al. [10] note that having software architects participate in release planning reduces the chance that important quality requirements are overlooked. Similarly, the SPM Competence Model [14] prescribes involving internal stakeholders like the software architect in requirements gathering and product roadmapping to improve SPM maturity. Vlaanderen et al. [15] position the software architect as the technical counterpart to the product manager.

Concrete advantages are claimed by several authors. Van Eck et al. [16] assert that alignment reduces operating costs and supports identifying new product opportunities. Helferich et al. [9] argue for the very same benefits and provide a concrete product opportunity example: tailoring distribution

[1] The challenge of bridging the Twin Peaks of requirements and architecture by ensuring alignment between these disciplines and their artifacts.

channels and product-related service offerings to different needs of customers segments and their respective product members. However, it is still unclear what the concrete processes are where both disciplines meet and benefit from each other, which prompts us to formulate our first position:

Position 1 *The Twin Peaks problem for product software is inherently different from that for their tailor-made cousins, due to the continuous inflow of requirements in high volume, an evolving product architecture, and a release-based lifecycle.*

SPOs sell the product in parallel to continuously incoming market requirements and software development [7]. This necessitates multiple, sequential releases of the same product to keep up with ever-changing requirements. The result is software products that, unlike tailor-made software, traverse many (concurrent) Twin Peaks over their lifecycle.

Furthermore, in its start-up phase the average SPO starts out without paying much attention to proper architectural design and documentation, but rather focuses on quickly shipping a minimum viable product [6], [11]. Years later, they cannot escape the consequences: a large, undocumented software architecture and numerous implicit requirements that are not recorded anywhere. The following paraphrase of a case at a financial SPO from the Netherlands illustrates these consequences:

> The product code is spread across three version control systems, each hosting code that is written in different programming languages. Two and a half years ago, our company began reconstructing an architectural model of the software product. So far, we captured just 25% of the entire product. These models cover the software components that are currently relevant for new product development. The core components are not covered at all.

Cases like this are not new. Already in the 1980s, Lehman [17] noticed that the change and decay process for software (architecture) continues until changes becomes too costly. In a large-scale review of software architectures [18], 30-50% of all architectural issues were found to originate from the fundamentals of product architecture and design.

To cope with these architectural issues, various (conceptual) approaches and tools have been developed, some of which having a strong architectural focus. Architectural Knowledge Management tools such as Archium [19] or ADDSS [20] facilitate sharing and reusing architectural knowledge among a broad variety of stakeholders [21]. Other approaches on the intersection of SA and RE directly draw inspiration from the Twin Peaks model to further relate software requirements and architecture [22]–[25]. However, none of these are widely adopted in business or academia. For example, the component-bus-system-property (CBSP) approach by Grünbacher [23], which delivers a "proto-architecture" to prescribe further architectural development, has been applied a limited number of times since its introduction [26], [27].

Moreover, both tool types are fundamentally inadequate for the Twin Peaks problem for software products. Despite the commendable intention to involve both software architects and requirements engineers, in practice the two disciplines do not use shared tools, with product managers using business-oriented tools [28], and software architects using technical tools closer to the software code [29].

Summarizing, software products are a difficult case for requirements/architecture alignment, due to the speed and volume of incoming requirements, the release-based lifecycle, and the existence of an evolving architecture. To avoid cost-ineffective changes late in a product's lifecycle, it is of utmost importance to establish alignment at its conception.

III. The Reciprocal Twin Peaks Model

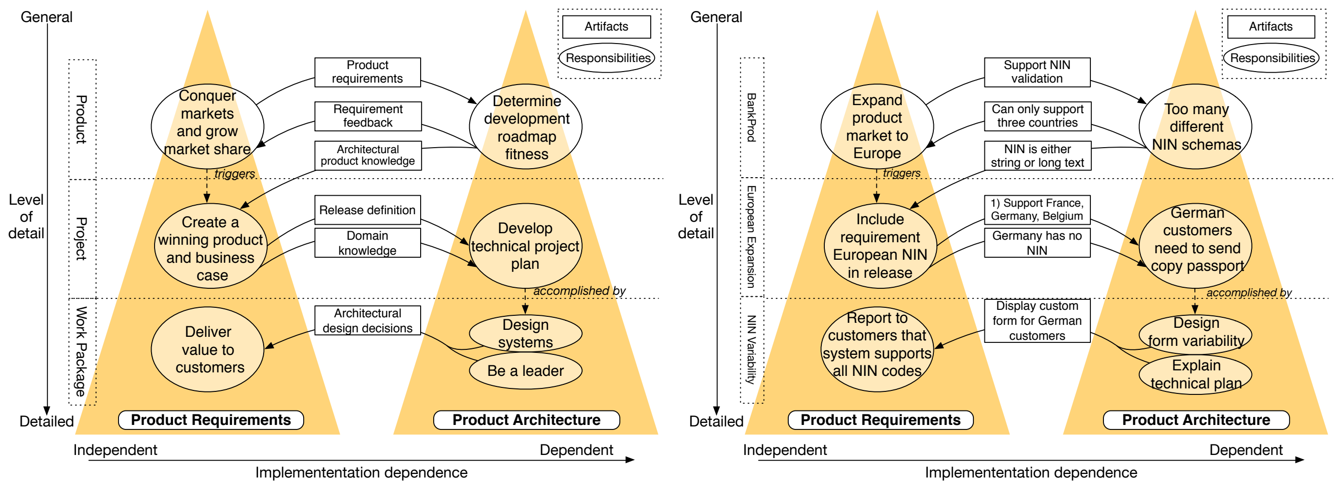
While the Twin Peaks model effectively explains the conceptual nature of the tight SPM and SA relationship, it does not explicate *how* to achieve (iterative) alignment in practice. To overcome this limitation, in earlier work, we introduced the Reciprocal Contributions Model (RCM) [12]. It describes the relationship between the responsibilities of product managers and software architects as defined by literature in both fields [28], [30], and highlights the mutual contributions that enable these stakeholders to fulfill their own responsibilities. Building on this, we formulate a position:

Position 2 *Bridging the Twin Peaks for product software corresponds to defining and realizing a stepwise architectural evolution that fits with the current roadmaps for the product and the architecture.*

The RCM explains the key artifacts to approach aligning product managers and software architects; however, it fails to communicate the level of detail of each element. While new requirements are rather abstract, a release definition contains fairly detailed requirements, and architectural design decisions (ADDs) are expressed in terms of concrete features. Integrating the RCM with the Twin Peaks model helps show each artifact's level of detail on the Y-axis, and highlights the higher implementation dependence of the architecture peak.

We substantiate this position by proposing the *Reciprocal Twin Peaks model of product requirements and architecture* (RTP), presented and exemplified in Fig. 1. The RTP validates and extends the RCM by (i) integrating it with original Twin Peaks [2], and (ii) including results from a comparative case study with multiple SPOs. Note that while the RCM depicts the concrete roles product manager and software architect, RTP centers on product requirements and architecture, to emphasize that RTP is applicable irrespective of the specific organizational roles and positions. Section IV details the differences between the RTP and RCM.

The RTP model consists of four primary elements: (1) two peaks; (2) SPM and SA responsibilities, denoted as ovals, adapted from previous work on SPM [28] and SA [30]; (3) artifacts, denoted as rectangles, produced trying to reach the responsibilities as we identified in [12]; and (4) artifact flows. We explain each element from top to bottom by means of a running example: BankProd, a fictitious software product of a company from the Netherlands that, among


General
Level of detail
Detailed
Product
Project
Work Package
Artifacts
Responsibilities
Conquer markets and grow market share
Product requirements
Requirement feedback
Architectural product knowledge
Determine development roadmap fitness
triggers
Create a winning product and business case
Release definition
Domain knowledge
Develop technical project plan
accomplished by
Deliver value to customers
Architectural design decisions
Design systems
Be a leader
Product Requirements
Product Architecture
Independent
Dependent
Implemententation dependence
BankProd
European Expansion
NIN Variability
Expand product market to Europe
Support NIN validation
Can only support three countries
NIN is either string or long text
Too many different NIN schemas
Include requirement European NIN in release
1) Support France, Germany, Belgium
Germany has no NIN
German customers need to send copy passport
Report to customers that system supports all NIN codes
Display custom form for German customers
Design form variability
Explain technical plan

(a) The Model (b) Applied to BankProd

Fig. 1: The reciprocal Twin Peaks model of product requirements and architecture

its functionalities, processes customers information including their national identification number (NIN). In Figure 1b, the model's general responsibilities and artifacts are replaced by their BankProd instantiations.

On a daily basis Alice, BankProd's product manager, receives new market requirements. Alice notices an increasing number of potential customers from abroad, which prompts her to investigate whether expanding the European market is a sound investment. She now intends to collaborate with Bob, the lead software architect of BankProd, to traverse the *two peaks* from a coarsely specified requirement to a successful system implementation that fully supports all EU NINs. Alice and Bob bridge the two peaks by refining the incoming market requirement through the three levels of the RTP model:

a) Product level: Expand product market to Europe is a specific instance of the product management responsibility to conquer markets and grow market share. While working on this responsibility, Alice produces an artifact that is relevant to both SPM and SA: the product requirement to support NIN validation. This artifact flows to the software architect, Bob, who examines the requirement's fitness with the development roadmap, and provides the following requirement feedback: "It is impossible to support all EU countries cost-effectively. Only three can be supported within an acceptable time-frame, which ones have priority?". Depending on the turnover three countries will generate, Alice needs to decide to proceed with a project or put the requirement on hold.

b) Project level: Believing that expanding to Belgium, France and Germany will grow market share cost-effectively, Alice starts a new "European expansion" project. She instantiates the project-level responsibility to create a winning product and business case by planning a new release and soliciting architectural product knowledge from Bob. The architect indicates that the current NIN database field is an integer and should become either a long text or a string depending on the max length of a NIN. Next, the product manager includes the product requirement in the release definition for the planning of the next release, and provides the domain knowledge that schemas for Belgium and France are available on Wikipedia, but that Germany has no NIN at all. These artifacts are input for Bob to develop a technical project plan, instantiated by German customers need to send copy passport, which requires implementing variability in the software product. Bob kicks off development of this feature.

c) Work Package level: Bob accomplishes the project through completing multiple work packages. In this example, Bob leads the development team in the responsibility to design the system by explaining his approach and rationale. The developers implement the requirement by displaying different forms depending on the customer's nationality. This ADD flows back to the product manager, which delivers the requirement's value to customers by releasing the new product version.

While the figure shows the responsibilities and the flows, it does not highlight an important aspect of the refinement process: not all requirements flow through all the responsibilities and artifacts as a linear route. In fact, in SPOs, the majority of requirements never reach the end of the twin peaks or are never communicated to the software architect as a product requirement to begin with. Recall that SPOs gather far more requirements from the market than they can implement. Thus, a product requirement can pass through the first stages until it is sufficiently specified for development, but might never be included in an actual release definition.

IV. Insights from the Case Study

To improve the first version of the RCM model, we conducted a comparative case study at SPOs to evaluate whether each of its elements and responsibilities are identifiable in practice. As part of a course on SPM at Utrecht University, 16 groups of two master students investigated the collaboration between software architects and product managers at an SPO. Following a case protocol, they conducted interviews with relevant actors of the company and wrote a detailed case study report. One part of the assignment was to describe the fit of RCM's responsibilities and artifacts with the case company's processes. These reports were manually processed by the first author, coding relevant passages for each question using Nvivo. Of the 16 reports, 10 were of sufficient quality to use as data for this evaluation. 9 SPOs are based in the Netherlands and 1 in Denmark. 5 have more than 200 employees, 3 between 50 and 200, 2 less than 50. Table I displays the results of the evaluation, indicating how many elements of the RCM were identified. Elements with gray shading were identified less frequently in practice.

TABLE I: Number of case studies in which the elements were identified. Major deviations from the RCM are shaded in gray

Responsibilities (n=10)	# cases
Conquer markets and grow market share	9
Create a winning product and business case	9
Deliver value to customers	9
Communicate with stakeholders	**6**
Develop project strategy	**6**
Design systems	9
Be a leader	9

Artifacts (n=10)	# cases
Product requirements	10
Requirements feedback	9
Architectural product knowledge	**6**
Release definition	8
Product context	**6**
Architectural design decisions	9

The results are largely positive: five responsibilities have only one negative indicator and four artifacts have up to two. All of these are incidental differences at different SPOs that occur for a wide variety of reasons. For example, the interviewee expected another name for the goal, or a third role was responsible for creating an artifact. Their arguments do not indicate a fundamental theoretical problem with these responsibilities and artifacts. However, the situation differs for four elements: out off the ten responding product managers, four indicated they could not recognize these elements in their organization. We analyze their comments below.

Goal: Communicate with stakeholders. Taylor [30] claims that software architects are in frequent contact with various stakeholders; however, this is not always the case for SPOs. The product manager is a liaison for external stakeholder requests [12], representing the software architect. This nuance, however, is not captured in the RCM and is easily lost on interviewees. All negative reports adhere to the following sentiment captured in one case study report: *"This goal is recognized, but not for the software architect. The software architect only communicates with the software product manager with regard to requirements, who functions as a broker for the communication with the customer"*. Based on this finding, we formulate the following position:

Position 3 *The software product manager is responsible for stakeholder communication, while the software architect should concentrate his communication efforts on internal technical stakeholders.*

The product manager expresses the wishes of non-technical stakeholders to architects. Then the software architect estimates the impact of new requirements considering previous architectural design decisions, short and long term plans, and stakeholder interests to formulate relevant requirement feedback. The product manager reports this feedback to non-technical stakeholders. Therefore, we redefined this goal in the RTP as "determine development roadmap fitness".

Goal: Develop project strategy. Although previous work [12], [30] explain that developing a project strategy is about finding the right technical solution taking into account product context, this distinction is not apparent in the RCM. Three interviewees stated that the product manager is responsible for the project strategy as a whole, while the architect is only responsible for the technical aspects. With this in mind, we rephrased the goal to "develop technical project plan".

Artifact: Architectural product knowledge. Two product managers have no need for this artifact because they are not responsible for the release definition, or have sufficient knowledge because they were previously software architects. Other two report that architectural product knowledge is not an explicit artifact. We chose to keep this artifact in the RTP because we support the view that codifying knowledge is important to minimize reliance on individuals.

Artifact: Product context. The feedback is similar to that of architectural product knowledge. Contrary to that artifact, we changed product context to *domain knowledge* in the RTP, for that term is a more concrete representation of the type of exchanged information, which is an essential part of the Zave and Jackson's classical requirement problem [31].

Additional responsibilities and artifacts. The several suggestions that we gathered did not contain any discernible common themes that made us change the model. Just one additional artifact was mentioned twice: release definition feedback. The architect might have technical suggestions for the release definition, which the product manager cannot identify on his own. This topic will be part of future work.

V. Conclusion

We have argued that the Twin Peaks problem for product software is an inherently different problem than that in the original Twin Peaks model (Position 1). We have explained that the key to bridging the Twin Peaks for product software is the definition and realization of a stepwise architectural evolution within the constraints imposed by the product and architecture roadmaps (Position 2). Through a comparative case study, we demonstrate that in practice, software architects should not focus on stakeholder communication (Position 3).

By specializing the Twin Peaks model for software products into the RTP model, we devise the theoretical foundations to bridge the twin peaks. Our final position states that this is possible only through adequate software tooling.

Position 4 *Integrated and cross-disciplinary software tools are essential for the alignment between product requirements and architecture, through the establishment and maintenance of consistency and traceability.*

These tools have to span across both disciplines, and are to be used jointly by software product managers and software architects. We posit that existing tools supporting the alignment are inadequate because the two roles do not share them, with product managers using business-oriented tools [28], and software architects using technical tools [29]. Additionally, the deep integration we seek goes beyond the coordination of activities and management of artifacts offered by traditional Application Lifecycle Management and its tools [32].

The alignment between requirements and architecture is achieved when these tools ensure the consistency between the exchanged artifacts, to avoid misunderstandings, and the traceability of requirements [33]. Future work will focus on developing such tools, with an emphasis on employing automated reasoning techniques to help maintain alignment and extract information on requirements and architectures from unstructured documents, the de-facto standard in SPO practice.

Acknowledgements

The authors would like to thank the students that participated in the software product management course for conducting and reporting on excellent case studies at international software producing organizations. Furthermore, we thank the other members of the Organisation and Information Software Architecture Group, Erik Jagroep and Leo Pruijt, for providing their perspective on the relationship between product requirements and architecture.

References

[1] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap," in *Future of Software Engineering*. ACM, 2000, pp. 35–46.

[2] B. Nuseibeh, "Weaving Together Requirements and Architectures," *Computer*, vol. 34, no. 3, pp. 115–119, Mar. 2001.

[3] J. Cleland-Huang, P. Avgeriou, J. E. Burge, X. Franch, M. Galster, M. Mirakhorli, and R. Roshandel, Eds., *TwinPeaks 2014*. ACM, 2014.

[4] L. Xu and S. Brinkkemper, "Concepts of Product Software," in *European Journal of Information Systems*, vol. 16, no. 5, 2007, pp. 531–541.

[5] C. Ebert, "Software Product Management," *Crosstalk*, vol. 22, no. 1, pp. 15–19, 2009.

[6] S. A. Fricker, "Software Product Management," in *Software for People*, ser. Management for Professionals. Springer, 2012, pp. 53–81.

[7] M. Khurum and T. Gorschek, "A Method for Alignment Evaluation of Product Strategies among Stakeholders (MASS) in Software Intensive Product Development," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 23, no. 7, pp. 494–516, 2011.

[8] E. Carmel and S. Sawyer, "Packaged software development teams: What makes them different?" *Information Technology & People*, vol. 11, no. 1, pp. 7–19, 1998.

[9] A. Helferich, K. Schmid, and G. Herzwurm, "Product Management for Software Product Lines: An Unsolved Problem?" *Communications of the ACM*, vol. 49, no. 12, pp. 66–67, Dec. 2006.

[10] M. Lindgren, C. Norstrom, A. Wall, and R. Land, "Importance of Software Architecture during Release Planning," in *IEEE/IFIP Conference on Software Architecture*. IEEE Computer Society, 2008, pp. 253–256.

[11] R. Deias, G. Mugheddu, and O. Murru, "Introducing XP in a start-up," in *International Conference on eXtreme Programming and Agile Processes in Software Engineering*, 2002, pp. 62–65.

[12] G. Lucassen, J. van der Werf, and S. Brinkkemper, "Alignment of Software Product Management and Software Architecture with Discussion Models," in *Software Product Management (IWSPM), 2014 IEEE IWSPM 8th International Workshop on*, Aug 2014, pp. 21–30.

[13] M. Galster, M. Mirakhorli, J. Cleland-Huang, J. E. Burge, X. Franch, R. Roshandel, and P. Avgeriou, "Views on software engineering from the twin peaks of requirements and architecture," *SIGSOFT Softw. Eng. Notes*, vol. 38, no. 5, pp. 40–42, Aug. 2013.

[14] W. Bekkers, I. van de Weerd, M. Spruit, and S. Brinkkemper, "A Framework for Process Improvement in Software Product Management," in *European Conference on Software Process Improvement*, 2010, pp. 1–12.

[15] K. Vlaanderen, S. Jansen, S. Brinkkemper, and E. Jaspers, "The agile requirements refinery: Applying scrum principles to software product management," *Information and Software Technology*, vol. 53, no. 1, pp. 58–70, 2011.

[16] P. A. T. van Eck, H. M. Blanken, and R. J. Wieringa, "Project GRAAL: Towards operational architecture alignment," *International Journal of Cooperative Information Systems*, vol. 13, no. 3, pp. 235–255, September 2004.

[17] M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, Sept 1980.

[18] J. Maranzano, S. Rozsypal, G. Zimmerman, G. Warnken, P. Wirth, and D. M. Weiss, "Architecture Reviews: Practice and Experience," *IEEE Software*, vol. 22, no. 2, pp. 34–43, March 2005.

[19] A. Jansen and J. Bosch, "Software Architecture as a Set of Architectural Design Decisions," in *IEEE/IFIP Conference on Software Architecture*, 2005, pp. 109–120.

[20] R. Capilla, F. Nava, S. Pérez, and J. C. Dueñas, "A Web-based Tool for Managing Architectural Design Decisions," *SIGSOFT Software Engineering Notes*, vol. 31, no. 5, Sep. 2006.

[21] P. Liang and P. Avgeriou, "Tools and Technologies for Architecture Knowledge Management," in *Software Architecture Knowledge Management*, M. Ali Babar, T. Dingsyr, P. Lago, and H. van Vliet, Eds. Springer Berlin Heidelberg, 2009, pp. 91–111.

[22] J. G. Hall, M. Jackson, R. C. Laney, B. Nuseibeh, and L. Rapanotti, "Relating Software Requirements and Architectures using Problem Frames," in *International Requirements Engineering Conference*, 2002, pp. 137–144.

[23] P. Grünbacher, A. Egyed, and N. Medvidovic, "Reconciling Software Requirements and Architectures with Intermediate Models," *Software and Systems Modeling*, vol. 3, no. 3, pp. 235–253, 2004.

[24] K. Pohl and E. Sikora, "COSMOD-RE: Supporting the Co-Design of Requirements and Architectural Artifacts," in *International Requirements Engineering Conference*. IEEE Computer Society, 2007, pp. 258–261.

[25] P. Avgeriou, J. Grundy, J. G. Hall, P. Lago, and I. Mistrk, *Relating Software Requirements and Architectures*, 1st ed. Springer, 2011.

[26] H. Vogl, K. Lehner, P. Grunbacher, and A. Egyed, "Reconciling Requirements and Architectures with the CBSP Approach in an iPhone App Project," in *International Requirements Engineering Conference*, 2011, pp. 273–278.

[27] C. Chen, D. Shao, and D. Perry, "An Exploratory Case Study Using CBSP and Archium," in *Workshop on Sharing and Reusing Architectural Knowledge—Architecture, Rationale and Design Intent*. IEEE, 2007.

[28] C. Ebert, "The Impacts of Software Product Management," *Journal of Systems and Software*, vol. 6, no. 80, pp. 850–861, 2007.

[29] M. Babar and I. Gorton, "Software Architecture Review: The State of Practice," *Computer*, vol. 42, no. 7, pp. 26–32, July 2009.

[30] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2010.

[31] P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 1, pp. 1–30, Jan. 1997.

[32] J. Kriinen and A. Vlimki, "Applying Application Lifecycle Management for the Development of Complex Systems: Experiences from the Automation Industry," in *Software Process Improvement*. Springer, 2009, vol. 42, pp. 149–160.

[33] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," in *International Requirements Engineering Conference*, 1994, pp. 94–101.