Analysis of Architecturally Significant Requirements for Enterprise Systems

Nan Niu, *Member, IEEE,* Li Da Xu, *Senior Member, IEEE,* Jing-Ru C. Cheng, and Zhendong Niu

***Abstract*—In designing and developing enterprise systems, systems engineers must consider the requirements that drive the important architecture decisions. Architecturally significant requirements tend to have a global impact on the underlying software infrastructure, and therefore need to be thoroughly examined. Despite the increasing effort in engineering enterprise systems' requirements, little is known about the analysis of architecture interactions and tradeoffs. In this paper, we propose a framework consisting of an integrated set of activities to help tackle requirements analysis in practice. Specifically, we leverage the quality attribute scenarios to elicit implicit yet significant requirements, to model requirements interplays, to manage terminological interferences, and to determine change impacts. We apply the proposed framework to a customer relationship management software system. The results show that the framework offers concrete insights and can be incorporated into an organization's systems practice with a moderate cost.**

***Index Terms*—Enterprise systems, requirements engineering, software architecture, systems engineering.**

I. Introduction

ENTERPRISE systems have become a critical enabler for large organizations to streamline, integrate, coordinate, and extend their business processes [1]. In practice, many system engineers exploit such a system to make sure that the enterprise-wide information can be stored in a standard fashion, related between functional levels, and communicated across management hierarchies. In another word, the capability of unifying the information at the enterprise level is integral to a systems approach to mitigating the problem of handling multiple fragmented information sources within an organization.

An important part of an enterprise system is software architecture, which consists of a set of structures and relationships for reasoning about the functionalities, qualities, and behaviors of the system [2]. From the systems engineering point of view, software architecture bridges between the stakeholder goals and the implementation of business functions. The decisions made during architectural analyses, therefore, not only shape what the enterprise system actually delivers, but also constrain the priority in which the requirements are being fulfilled. The requirements that play a crucial role in determining the enterprise system's software architecture are referred to as architecturally significant requirements [2].

The literature on software architecture for enterprise systems has received increasing attention lately. Different software architecture proposals have been presented to support the development of a wide range of industrial applications [3]–[5], including domain-specific enterprise systems [6], real-time controls [7], safety-critical infrastructures [8], agent-based platforms [9], and service oriented approaches [10]. For any enterprise system, determining architecturally significant requirements is of vital importance. This is not only because merely a subset of all the requirements can be treated as architecturally significant, but also due to the fact that the underlying architecture imposes the shaping limits to how the enterprise system can evolve throughout its life span. While changes conforming to the software architecture can be readily accommodated, the ones against the architectural blueprint are unlikely to be addressed adequately without serious and sometimes prohibitive reengineering efforts.

When dealing with an enterprise system's software architecture in practice, systems engineers need to consider multiple and oftentimes conflicting requirements [11]. For instance, tradeoff exists between the planned downtime required to address enterprise system's maintainability and the availability of the system. In other words, selecting certain requirements to be architecturally significant simultaneously excludes some other requirements being considered as the driving forces for the development of the enterprise system.

Making tradeoffs at the architectural design level has become particularly relevant to IIIE (Industrial Information Integration Engineering) [1], where enhancing industrial

Manuscript received July 16, 2012; revised February 13, 2013; accepted February 20, 2013. Date of publication March 29, 2013; date of current version August 21, 2014. The work of N. Niu was supported by the U.S. Army Engineer Research and Development Center Award W912HZ-10-C-0101 and the U.S. National Science Foundation Award CCF-1238336. The work of L. Da Xu was supported in part by the National Natural Science Foundation of China Award 71132008, the Changjiang Scholar Program of the Ministry of Education of China, and the U.S. National Science Foundation Award 1044845.

N. Niu is with the Department of Computer Science and Engineering, Mississippi State University, Starkville, MS 39762 USA (e-mail: niu@cse.msstate.edu).

L. D. Xu is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China, and also with the Department of Information Technology and Decision Sciences, Old Dominion University, Norfolk, VA 23529 USA (e-mail: lxu@odu.edu).

J.-R. C. Cheng is with the Information Technology Laboratory, U.S. Army Engineer Research and Development Center, Vicksburg, MS 39180 USA (e-mail: ruth.c.cheng@usace.army.mil).

Z. Niu is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China (e-mail: zniu@bit.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2013.2249892

1932-8184 © 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

fabrication and manufacturing processes often requires striking a balance between the various quality factors [12], [13]. Despite the increasing number of proposals for designing enterprise systems, little is known about analyzing software architecture tradeoffs to tackle the dependencies, interactions, and interplays of architecturally significant requirements [14], [15]. Lack of this knowledge is a serious problem since systematic methods are not yet available for systems engineers to reason, analyze, and determine a suitable software infrastructure to parallel modern industrial automation.

In this paper, we fill the gap by presenting a practitioner-oriented approach to analyzing and evaluating architecturally significant requirements for enterprise systems. We review the key requirements in enterprise system's design and leverage the quality attribute scenarios [16] to assess the degree to which software architecture choices influence the fulfillment of the architecturally significant requirements. Our aim is to equip systems engineers with an innovative way to understand software design's strengths and weaknesses. This novel understanding, in turn, is expected to lead to informed decisions on the selection of an enterprise system solution best fitting the business purposes and best satisfying stakeholder goals. We evaluate our approach on an organization's customer relationship management software. The study shows that software architecture tradeoff analysis offers concrete insights into enterprise system's design and evolution, and that scenario-based analysis can be incorporated into an organization's systems engineering with a moderate cost.

The remainder of the paper is organized as follows. Section II lays the background of our research. Section III details our scenario-based software architecture analysis approach. Section IV presents an application of our approach and discusses the empirical study results. Section V draws concluding remarks and outlines avenues for future work.

II. State of the Art in Enterprise System Architecture

In the past decade, enterprise systems have emerged as a strategic and effective tool for integrating and extending the business processes across a supply chain's functional boundaries at both intra- and inter-organizational levels [1]. Standard enterprise system packages, such as as enterprise resource planning (ERP) and customer relationship management (CRM), have become the core information processing capabilities for many industries. In the meantime, the enterprise system market exemplified by ERP applications is one of the fastest growing and most profitable areas in the software industry. Such a tight coupling of enterprise systems and software engineering makes architectural analysis particularly important.

When making architectural decisions, taking the key driving factors into account is indispensable. Among these factors, business goals and stakeholder needs that have a broad scope of impacts are candidate requirements to be treated as architecturally significant. For example, for a logistics management software solution, performance and accuracy are extremely crucial in delivering high-quality routing, scheduling, and dispatching services. In contrast, requirements like persistence, though relevant, may require only a secondary and thus relatively flexible treatment during software implementation.

TABLE I
ENTERPRISE SYSTEM'S ARCHITECTURALLY SIGNIFICANT REQUIREMENTS

Software Architecture	**Sample Application Domain**	**Significant Requirements**
General-purpose software packages	Enterprise resource planning [17, 18]	Customizability, Flexibility
Domain-specific architecture	Capital-budgeting or mortgage-pricing [1, 6]	Reusability, Extensibility
Distributed architecture	Manufacturing supply chain management [9, 19]	Performance, Efficiency
Agent and multiagent systems	Mass customization manufacturing [20, 21]	Adaptability, Autonomy
Service-oriented architecture	Automated multisensory fusion [4, 22]	Interoperability, Composability

It is interesting to note the relationship between architecturally significant requirements and non-functional requirements [11]. In software engineering, functional requirements describe what the system does and nonfunctional requirements describe how well system functions are accomplished. A sample functional requirement in the enterprise system domain is integrating accounts payable and receivables. However, such a function is achieved differently in a modern ERP package like SAP R/3 and a traditional computer application like spreadsheet. Nonfunctional requirements, such as accuracy and reliability, represent global concerns on a system's development and operational costs. Therefore, we argue that NFRs are natural candidates to be viewed as architecturally significant. However, the inverse is not necessarily true. Persistence in the logistics management software mentioned above illustrates this distinction.

Real life enterprise system development pays special attention to the significant requirements. In Table I, we list software architecture solutions commonly proposed for engineering enterprise systems. For each architecture, Table I also provides the sample application domain and teases out the architecturally significant requirements. Even though Table I is not meant to be exhaustive, it helps to reveal the contemporary systems-oriented concerns in the literature. A more detailed discussion of enterprise system's requirements can be found in [1], [3], and [5].

Several observations can be drawn from Table I. First, different architecture solutions do distinguish in their respective significant requirements. This is in line with one of the primary roles of such requirements in which they drive the selection among myriads of alternative designs and ultimate implementations in software engineering [11]. Second, the architecturally significant requirements represent only a small subset of all the requirements. The main reason for the significant requirements to be highly selective is because if they are incorrect, incomplete, inaccurate, or lack details, then an

enterprise architecture based on them will also likely contain errors. Third, architecturally significant requirements do not always manifest themselves in an explicit way. This requires a systematic approach to uncovering implicit yet significant requirements. We next present a practitioner-oriented framework for supporting the identification of architecturally significant requirements and the examination of the architecture tradeoffs.

III. Analysis of Architecturally Significant Requirements

In this section, we first highlight the practical challenges faced by systems engineers in analyzing the enterprise system's architecturally significant requirements. We then present a framework consisting of an integrated set of activities to facilitate the practitioner's requirements engineering tasks. Our framework is illustrated through a running example characterizing a typical enterprise system development scenario. We evaluate the applicability and the effectiveness of the proposed framework via an industrial case study in the next section.

A. Challenges of Engineering Significant Requirements

In a seminal paper, Zave [23] provides one of the clearest definitions of requirements engineering by stating that its primary purpose is concerned with the real-world goals for, functions of, and constraints on software systems. Nuseibeh and Easterbrook [24] further characterize the main requirements engineering activities which include: planning and eliciting requirements, modeling and analyzing requirements, communicating and agreeing upon requirements, and realizing and evolving requirements. Using this characterization, we identify the challenges associated with the engineering of enterprise system's requirements in each of the activity categories.

In the planning and eliciting phase, a challenge is to identify not only explicit but also implicit architecturally significant requirements. Take logistics management as an example, while performance is an explicit need, robustness may be an implicit requirement that the software architecture shall support in a primary fashion. This is because real time dispatching requires the software-intensive system to generate routes in a prompt as well as a fault tolerable way. Thus, systems engineers need support for eliciting implicit requirements that drive software architecture design.

For requirements modeling, a main challenge is to reason about the tradeoffs among the interrelated and sometimes competing needs. For instance, enhancing the security aspect of the reporting module of an ERP system via a multistep authentication mechanism can potentially hurt the usability of the software. Because architecturally significant requirements typically represent global concerns, their interactions are complex [14]. Positive interactions indicate that satisfying a requirement will help fulfill another one, e.g., ensuring integrity contributes positively to a software system's accuracy. On the contrary, negative interactions imply that certain requirements are inherently conflicting to each other. A classic example in designing algorithms such as scheduling is that the performance improvement made in time is often at the cost of space, and vice versa. Analyzing architecturally significant requirements thus requires sound support for recognizing these subtle interactions and modeling them appropriately.

As far as communicating requirements among the stakeholders is concerned, a great challenge relates to the terminological interference problem that we identified in our earlier work [25], [26]. In particular, there are four possible situations when different stakeholders perceive a share problem domain:

1) consensus where stakeholders use terminology and concepts in the same way;
2) correspondence where they use different terminology for the same concepts;
3) conflict where they use the same terminology for different concepts;
4) contrast where stakeholders differ in terminology and concepts.

We interpret each correspondence and conflict as instances of terminology interference. For example, what one person calls responsiveness might correspond to performance in another person's description. For enterprise systems targeting at the mobile platform, one stakeholder might interpret usability as easy to learn, another as economically exploiting the screen estate, and yet another as less error prone to be interacted with. Therefore, identifying and managing terminological interferences are crucial for effective communication of an enterprise system's requirements.

When the architecturally significant requirements are engineered, their implications on future evolution of the enterprise system cannot be overlooked. Because tradeoff exists among the requirements and only a selected subset can be embodied in the underlying architecture, any modification that mismatches, violates, or otherwise pushes the architecture in a different direction will be difficult to accommodate. Take the real time dispatching software system as an example; if accuracy, robustness, and mobility are built into the architecture that results in applications in handheld devices like smart phones and tablets, then making modifications to achieve a socket-based multistep authentication security level may require significant reengineering effort. This is analogous to the evolution scenarios of a standard classroom: changing it to a remote education classroom will be easy by adding necessary devices, changing it to a conference room will take longer due to restructuring and remodeling, and changing it to an Image Maximum (IMAX) theater would be next to impossible from the software and systems engineering's cost-effectiveness perspective.

In sum, requirements engineering is well recognized in industry to be critical to the success of any major development project, especially the enterprise systems directly related to the business success and performance. Studies have shown that requirements defects cost 10-200 times more to correct once fielded than they would if they were detected during early architectural analysis stages [15], [24]. However, engineering an enterprise system's requirements is not without challenges. We have identified in this subsection the main challenge associated with each of the four requirements engineering phases, namely, implicit requirements during elicitation, trade-

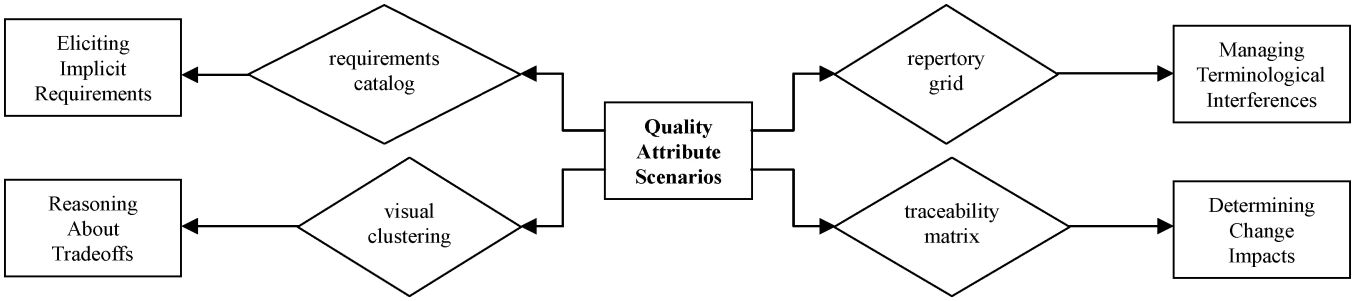


Fig. 1. Framework facilitating requirements analysis for enterprise systems.

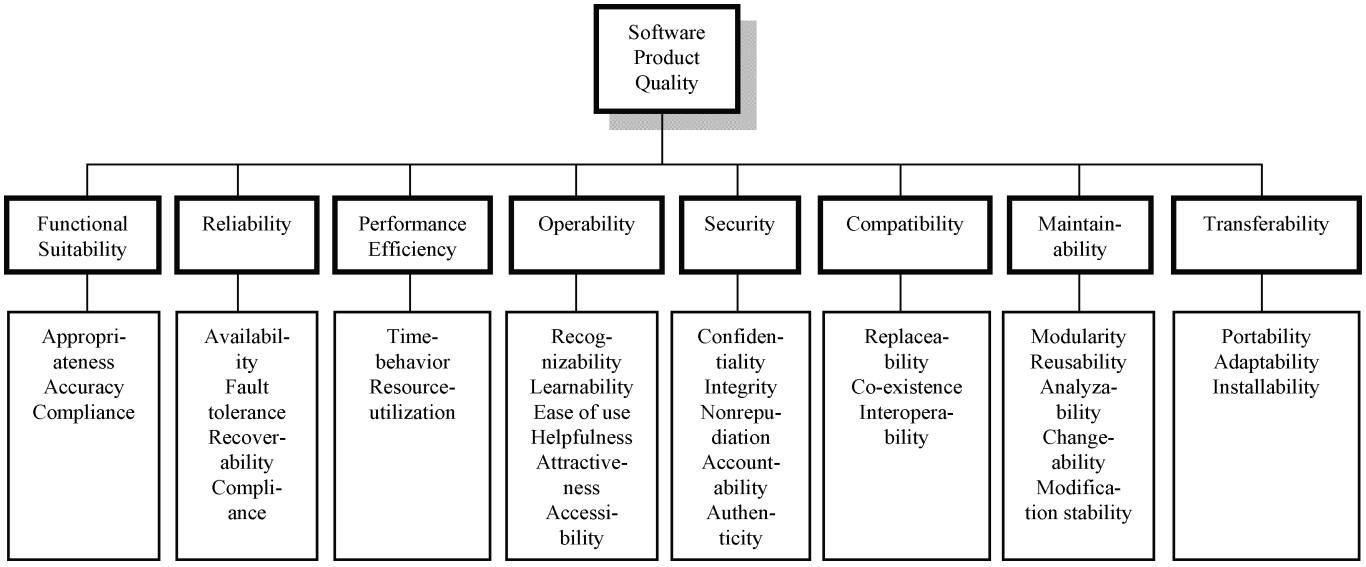


Fig. 2. Sample catalog codifying non-functional requirements, also known as quality attributes (adapted from [11]).

off analysis during modeling, terminological interference during communication, and reasoning about change impact during evolution. We next propose a framework to help tackle these challenges.

B. Tackling the Challenges via Quality Attribute Scenarios

Fig. 1 shows an overview of the proposed framework, in which the quality attribute scenarios [16] play a central role. In essence, scenarios are brief narratives of expected or anticipated system uses from both user and developer views. For our purpose, scenarios provide a look at how the enterprise system would satisfy the architecturally significant requirements in various use contexts. We therefore adopt the quality attribute scenarios which are created for describing a system interaction with respect to some quality attribute [16]. Explicitly relating to the quality aspect of the intended system has also been the main difference between these scenarios and the use case scenarios of object-oriented software engineering or user stories of agile software development.

To illustrate the proposed framework, we use a running example which is taken from the case study discussed later. The subject module is on reporting customer transactions for anomaly detection and market trend analysis. For a business-critical module like reporting, eliciting the correct set of architecturally significant requirements is of paramount importance. As shown in Fig. 1, an effective tool for the elicitation task is the quality attribute catalog [11], which codifies and classifies existing nonfunctional requirements into a manageable hierarchy. While no catalog can be absolutely complete, a catalog like the one in Fig. 2 helps organize the common requirements and therefore acts as a checklist of potentially significant concerns. For our running example, functional suitability (e.g., accuracy) and security (e.g., confidentiality) represent important categories for eliciting (especially implicit) requirements driving software architecture design.

Having identified a list of architecture drivers, the systems engineers need to devise the operational definition for these abstract yet hard-to-quantify requirements. This is where scenarios could make direct contributions. Fig. 3a displays a partial repertory grid built during our case study (discussed later in Section IV). The rows represent elicited requirements that are significant to the software architecture. Each requirement is specified by two opposing poles: the positive end (annotated with "+") and the negative end (annotated with "-"). The columns represent the scenarios describing how the enterprise system is used under various circumstances. Because these scenarios must relate to quality attributes, their contributions to the requirements are qualitatively measured by the scheme defined in Fig. 3(b). Note that such rating scales are popular among requirements engineering methods (e.g., [11] and [26]).

	Sce_1	Sce_2	…	Sce_m	
-Accuracy	2	4	…	3	+Accuracy
-Anonymity	4	2	…	5	+Anonymity
- Confidentiality	4	2	…	5	+Confidentiality
…	…	…	…	…	…
-Requirement_n	3	4	…	1	+Requirement_n

(a)

Scale	Interpretation	Degree of support for the right (positive) end of Fig. 3a
5	Scenario makes the requirement	Completely
4	Scenario helps the requirement	Partially
3	Neutral relationship	Neutrally
2	Scenario hurts the requirement	Partially not
1	Scenario breaks the requirement	Completely not

(b)

Fig. 3. Repertory grid and qualitative rating scales. (a) Sample repertory grid. (b) Rating scale used in the repertory grid.

The repertory grid shown in Fig. 3 is an effective tool in identifying the terminological interference problems related to requirements communication. In Fig. 3(a), for example, anonymity and confidentiality appear to be receiving the same ratings from all the usage scenarios. If these terms are indeed used interchangeably, then a terminological correspondence needs to be established. Otherwise, we could further ask the systems engineer to come up with a scenario where these two concepts show distinctions. In our case study, one of the systems engineers distinguished anonymity from confidentiality via a scenario where the reporting manager may know the customer from whom an informal complaint is issued. In this case, anonymity is not supported. However, confidentiality will be ensured if the reporting manager protects the customer's identity when describing the complaint issue with other management teams. As illustrated in this example, the repertory grid together with the qualitative ratings serves as a direct communication means for clarifying the architecturally significant requirements that would otherwise remain ambiguous or concealed.

To gain insights into the interrelationships among requirements, we leverage visual clustering that has been recently developed [27]. Fig. 4 illustrates visual clustering of subsets of requirements in our case study. Each requirement is rendered as a circular node in the 2-D space. The distance between the nodes is calculated based on the textual similarity of the requirements. Our visual clustering tool [27] employs a force-based graph drawing algorithm to balance the clustering layout. A couple of features related to requirements tradeoffs are worth mentioning. First, the size of the node implies the significance level of the requirement: the bigger the node, the more interconnections and thus significance the requirement has. Second, the layout algorithm automatically positions more homogeneous and matching requirements into a coherent cluster, whereas potentially conflicting requirements are isolated in different clusters. In Fig. 4, for example, the representative requirement of the cluster on the left relates to consistency and the one on the right is concerned with availability. As will be discussed later in Section IV, these two architecturally significant requirements reveal a competing need for the organization's enterprise system.

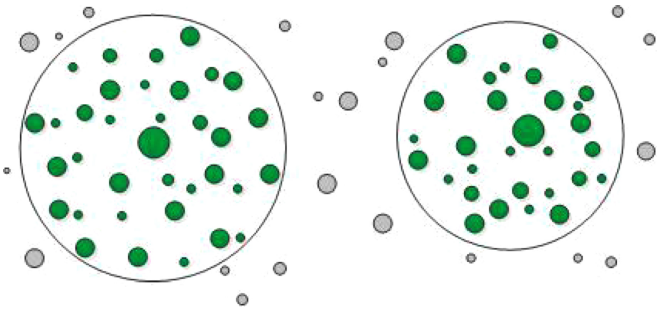


Fig. 4. Visual clustering of architecturally significant requirements.

The determination of change impacts, as shown in Fig. 1, is supported by the requirements traceability matrix that tracks the life of a requirement in both forward and backward directions [28]–[30]. It is important to note that the proposed activities are not separated from each other, but are integrated within a coherent framework. For example, the repertory grid shown in Fig. 3 can be used for refining how similarity is computed for visual clustering. The visual clustering results of Fig. 4, in a similar vein, can be used to facilitate the identification of significant requirements (e.g., via the heuristic of node size). This complements the reliance of catalog to uncover significant requirements. The fact that anonymity – an architecturally significant requirement in our case study – is not part of the catalog of Fig. 2 illustrates the seamless integration of the proposed framework. Overall, the framework is intended to codify a set of best practices in requirements engineering to support the design and analysis of enterprise systems. We next present an industrial case study to assess the applicability and effectiveness of the proposed framework.

IV. Case Study

We applied the framework to the engineering of an enterprise system's requirements in practice. Our partner organization is an interdisciplinary research and development center that specializes in formulating and implementing big data solutions for many constituencies and programs. The software-intensive system under our investigation is a customer relationship management (CRM) solution targeted for a leading electronics manufacturing provider in the southeast of the United States. In order to honor the confidentiality agreements, we will use the pseudonym CNY for the CRM software project. Before detailing the actual study context, we describe the rationale of choosing the case study as our empirical evaluation basis.

A. Why a Case Study?

The main reason that we adopt case study as the basis for our empirical evaluation is that the investigation of a contemporary phenomenon is suitable for addressing the how and why questions that can otherwise be difficult to answer through controlled experiments. Essentially, the benefits and obstacles of using our proposed framework are only likely to be evident for the ongoing real-world project, under conditions that cannot be replicated in the laboratories. In particular, the study of applying requirements engineering methods like ours

cannot be separated from the organizational context and the effects many take weeks or months to appear.

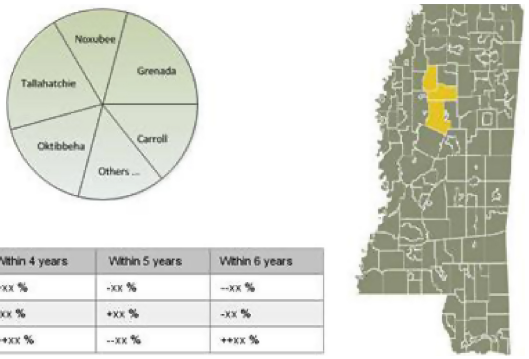
We therefore designed an exploratory case study [31] in collaborating with the CNY project team. An exploratory case study is appropriate for preliminary inquiries in which it is not yet clear which phenomena are important, or how to measure these phenomena. In our case, we were particularly interested in understanding the practical impacts of the set of activities proposed in our framework on the engineering of CNY's architecturally significant requirements. While the handling of single requirement such as stability [8] and extensibility [12] is discussed in the literature, we know little about the issues arising when multiple requirements interact or even interfere with each other. Because of this, we designed the case study to explore how enterprise system's requirements and architecture tradeoffs are handled in practice.

B. Results

We collaborated with the CNY team in eliciting and analyzing the architecturally significant requirements. One of such requirements is shown in Fig. 5. The need here is to use the geographical regions to analyze and predict sale trends so as to better serve regional customers. Other CNY requirements involve organizing product campaigns, personalizing customer services, optimizing recruiting plans, maintaining customer loyalty, and the like. The CNY team supported our research by sharing with us several documents, including the concept of operations, software requirements specification, software design description, and so forth. Studying these materials helped us to understand the context of CNY's customer relationship management. Our main analysis was then performed with a group of six domain experts in a half day joint application development (JAD) workshop [32]. The experts hold three main roles for the CNY project: business analyst, database administrator, and software architect.

During the workshop, we worked with CNY's domain experts in developing quality attribute scenarios. We found the effort involved in carrying out the activities of our proposed framework was moderate and the process was straightforward so that it could be integrated into an organization's software and systems engineering practice. From our experience, scenario generation was much like software testing. We cannot prove that we have a sufficient number of test cases, but we can determine a point at which adding new test cases yields only negligible improvement. In practice, the available resource is another factor. With the help of CNY's domain experts, we devised about half a dozen scenarios in only half an hour, which turned out to be adequate for our analysis.

In the process of devising the quality attribute scenarios, the requirements with significant impact on the CNY's underlying software architecture emerged. To our surprise, the nonfunctional requirements catalog shown in Fig. 2 was found interesting but to have only marginal values by the domain experts. The main concern was that the catalog was too general to be fit for CNY's particular development context. We therefore argue that any one-size-fits-all catalog can serve only a starting point for eliciting architecturally significant requirements.



County name	Within 4 years	Within 5 years	Within 6 years
Tallahatchie	+xx %	-xx %	--xx %
Grenada	-xx %	+xx %	-xx %
Carroll ...	++xx %	--xx %	++xx %

Fig. 5. Sample customer relationship management requirement in CNY.

The construction of repertory grid was very well received during the JAD workshop. To some extent, the process of qualitatively relating usage scenarios to quality requirements was more important than the final product of the grid itself. Such a knowledge acquisition process effectively enabled the exchange of stakeholders' viewpoints in a novel and exciting way. The CNY's domain experts were able to recognize subtle but important discrepancies and resolved them on the fly as needed.

The tradeoff analysis was facilitated by both the repertory grid and the visual clusters, and was iterated for several rounds to resolve ambiguities and consolidate expert opinions. As an example, the requirements analysis helped the CNY team to derive an actionable decision on trading off consistency for availability. In CNY, timeline consistency handled the common case efficiently and with clean semantics, but it was not perfect. Occasionally, an entire customer datacenter would go down and then any records mastered in that datacenter would become unwriteable. This scenario exposed the important tradeoff between consistency, availability, and partition tolerance. According to the CNY's domain experts, only two of those three requirements could be guaranteed and thus in reality the CNY project only had a choice between consistency and availability.

The JAD workshop offered much more insights than simply identifying the tradeoff. Specific architecture designs were also proposed and discussed. If a datacenter went offline, possibly with some new updates not yet being propagated to other replicas, the CNY team decided to either preserve consistency by disallowing updates until the datacenter was recovered or preserve availability by violating timeline consistency and allowing some updates to be applied to a nonmaster record. The tradeoff, in summary, related to the contingency that timeline consistency provided a simple semantics for how record updates were propagated, and flexibility in how application could trade off read latency for currency. Uncovering tradeoffs of this nature was regarded as having great value by the CNY team.

Table II summarizes the main results of our exploratory case study. In Table II, the challenges associated with engineering an enterprise system's requirements are listed in the leftmost column. Our proposed framework, shown in Fig. 1, helps tackle these challenges via a scenario-centric framework of integrated activities. The middle column of Table II provides concrete examples from the CNY's requirements analyses. The

TABLE II
RESULT SUMMARY OF THE EXPLORATORY CASE STUDY

Challenge	Examples	Occurrences
Elicit implicit requirements	Robustness, fault tolerance, dependability	4
Manage terminological interferences	Correspondence between performance and responsiveness	7
Reason about tradeoffs	Consistency vs. availability	3
Determine change impacts	Access control impacting timeline consistency	4

total number of occurrences of each challenge category is shown in the rightmost column of Table II. It is encouraging to note that a nontrivial number of requirements problems were identified and analyzed during a relatively short, 30-minute JAD session. On the other hand, the generated insights were viewed as high-quality and valuable to the CNY team. As a matter of fact, devising scenarios to make significant requirements measurable and analyzing software architecture tradeoffs were not just an academic exercise, but something CNY considered incorporating in daily practice because they could identify many potential problems early in the engineering life cycle and at a moderate cost.

C. Threats to Validity

Several factors can affect the validity of our exploratory case study [31]. Construct validity concerns establishing correct operational measures for the concepts being studied. The main construct in our case study is architecturally significant requirements. While we offered guidance based on nonfunctional requirements catalog (Fig. 2), the research community has not reached to an agreed-upon definition of this important construct. In fact, our experience shows that general purpose catalog will unlikely capture a complete and deep set of architecturally significant requirements for particular application domains. The best measure adopted in our case study, therefore, came from the subjective opinions of CNY's project members.

Regarding the internal validity, a major limitation of our study design is the researchers were also the JAD workshop moderators. This compounds the problem of experimenter bias, because the researchers may manipulate the study to obtain the expected outcome. We mitigate such a threat mainly by choosing an exploratory case study as the basis of our empirical evaluation. We intentionally avoid using an explanatory or causal study because we wanted to concentrate more on reporting our experience than on trying to prove our predetermined hypotheses. This step cannot remove the threat of experimenter bias entirely; only replication with neutral participants can address this issue.

The results of our study may not generalize beyond CNY's organizational and situational characteristics, a threat to external validity. Nevertheless, our investigation of the contemporary project within its real-life context, together with the validation carried out in a real industry setting, provides a firm footing for applying the requirements engineering framework for the analysis of enterprise systems. Finally, in terms of reliability, we expect that replications of our study should offer results similar to ours. Of course, the requirements under study may differ, but the underlying trends should remain unchanged.

V. CONCLUSION

A large group of today's systems engineers focus on developing software for enterprise systems, and other industrial computer systems since factory automation is now driven by information technologies. This is particularly beneficial in a supply chain environment where enterprise systems have become a critical enabler for modern manufacturing and service enterprises to streamline processes, thereby achieving effectiveness, efficiency, competency, and competitiveness [33]–[35]. Despite the emerging trend that more informatics is carried out early in the engineering life cycle, the current enterprise systems literature and practices have yet to parallel this evolution. Specifically, the research on analyzing architecturally significant requirements for enterprise systems remains inadequate to deal with today's industrial and systems automation. Thus, there is a critical need for supporting the engineering of significant requirements and making tradeoffs about an enterprise system's software architecture early in the development life cycle.

In this paper, we identified the challenges confronting the requirements engineering enterprise systems, reviewed a contemporary set of architecturally significant requirements, proposed a practitioner-oriented framework for analyzing the architecture tradeoffs, and conducted an empirical study to evaluate our approach. The study on an organization's customer relationship management software demonstrated our approach's applicability and usefulness. On one hand, only moderate effort was required to carry out the analysis. On the other hand, the discovered requirements, interactions, and architecture tradeoffs provided valuable and practical insights into enterprise system design and evolution.

There are several dimensions in which our work can be continued. First, more in-depth empirical studies are needed to lend strength to the preliminary findings reported here. Second, identifying and codifying industrial-oriented architecturally significant requirements catalogs and analysis patterns [6] are in order. Third, in terms of evaluating the enterprise system's software architecture alternatives, it is worth comparing a separation-of-concern approach like ours with a component-based one like [5].

ACKNOWLEDGMENT

The authors would like to thank all the management and staff at the partner company for their invaluable collaboration.

REFERENCES

[1] L. D. Xu, "Enterprise systems: State-of-the-art and future trends," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 630–640, Nov. 2011.
[2] L. Blass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2003.

[3] H. M. Kim, S. Lu, J. S. Kim, and B.-D. Kim, "Parallel, multistage model for enterprise system planning and design," *IEEE Syst. J.*, vol. 4, no. 1, pp. 6–14, Mar. 2010.
[4] K. J. Rothenhaus, J. B. Michael, and M.-T. Shing, "Architectural patterns and auto-fusion process for automated multisensor fusion in SOA system-of-systems," *IEEE Syst. J.*, vol. 3, no. 3, pp. 304–316, Sep. 2009.
[5] M. Fonoage, I. Cardei, and R. Shankar, "Mechanisms for requirements driven component selection and design automation," *IEEE Syst. J.*, vol. 4, no. 3, pp. 396–403, Sep. 2010.
[6] S. Wu, L. Xu, and W. He, "Industry-oriented enterprise resource planning," *Enterprise Informat. Syst.*, vol. 3, pp. 409–424, May 2002.
[7] H. Dagdougui, R. Minciardi, A. Ouammi, M. Robba, and R. Sacile, "A dynamic decision model for the real-time control of hybrid renewable energy production systems," *IEEE Syst. J.*, vol. 4, no. 3, pp. 323–333, Sep. 2010.
[8] M. Y. H. Low, M. Zeng, W. J. Hsu, S. Y. Huang, F. Liu, and C. A. Win, "Improving safety and stability of large containerships in automated stowage planning," *IEEE Syst. J.*, vol. 5, no. 1, pp. 50–60, Mar. 2011.
[9] V. Kumar and N. Mishra, "A multi-agent self correcting architecture for distributed manufacturing supply chain," *IEEE Syst. J.*, vol. 5, no. 1, pp. 6–15, Mar. 2011.
[10] D. Parlanti, F. Paganelli, and D. Giuli, "A service-oriented approach for network-centric data integration and its application to maritime surveillance," *IEEE Syst. J.*, vol. 5, no. 2, pp. 164–175, Jun. 2011.
[11] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Boston, MA, USA: Kluwer Academic, 2000.
[12] B. O'Brien, N. M. Ijumba, and H. Whitehead, "Asset management optimization through integrated systems thinking and n-1 contingency capability for refurbishment," *IEEE Syst. J.*, vol. 5, no. 3, pp. 321–331, Sep. 2011.
[13] M.-F. Tsai, C.-K. Shieh, T.-C. Huang, and D.-J. Deng, "Forward-looking forward error correction mechanism for video straming over wireless networks," *IEEE Syst. J.*, vol. 5, no. 4, pp. 460–473, Dec. 2011.
[14] W. N. Robinson, S. D. Pawlowski, and V. Volkov, "Requirements interaction management," *ACM Comput. Surv.*, vol. 35, pp. 132–190, Jun. 2003.
[15] N. Niu, A. Y. Lopez, and J.-R. C. Cheng, "Using soft systems methodology to improve requirements practices: An exploratory case study," *IET Softw.*, vol. 5, pp. 487–495, Dec. 2011.
[16] I. Ozkaya, L. Bass, R. L. Nord, and R. S. Sangwan, "Making practical use of quality attribute information," *IEEE Softw.*, vol. 25, no. 2, pp. 25–33, Mar.–Apr. 2008.
[17] N. Niu, M. Jin, and J.-R. C. Cheng, "A case study of exploiting enterprise resource planning requirements," *Enterprise Inform. Syst.*, vol. 5, pp. 183–206, May 2011.
[18] Q. Liang, "Situation understanding based on heterogeneous sensor networks and human-inspired favor weak fuzzy logic system," *IEEE Syst. J.*, vol. 5, no. 2, pp. 156–163, Jun. 2011.
[19] H. Rahnama, A. Sadeghian, and A. M. Madni, "Relational attribute integrated matching analysis (RAIMA): A framework for the design of self-adaptive egocentric social networks," *IEEE Syst. J.*, vol. 5, no. 1, pp. 80–90, Mar. 2011.
[20] M. Tu, J.-H. Lin, R.-S. Chen, K.-Y. Chen, and J.-S. Jwo, "Agent-based control framework for mass customization manufacturing with UHF RFID technology," *IEEE Syst. J.*, vol. 3, no. 3, pp. 343–359, Sep. 2009.
[21] H. K. Chan, "Agent-based factory level wireless local positioning system with zigbee technology," *IEEE Syst. J.*, vol. 4, no. 2, pp. 179–185, Jun. 2010.
[22] C. Trummer, C. Ruggenthaler, C. M. Kirchsteiger, C. Steger, R. Weiß, M. Pistauer, and D. Dalton, "Searching extended IP-XACT components for SoC design based on requirements similarity," *IEEE Syst. J.*, vol. 5, no. 1, pp. 70–79, Mar. 2011.
[23] P. Zave, "Classification of research efforts in requirements engineering," *ACM Comput. Surv.*, vol. 29, pp. 315–321, 1997.
[24] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," in *Proc. Future Softw. Eng.*, Limerick, Ireland, 2000.
[25] N. Niu and S. Easterbrook, "Managing terminological interferences in goal models with repertory grid," in *Proc. Int. Req. Eng. Conf.*, Minneapolis, St. Paul, MN, USA, 2006, pp. 296–299.
[26] N. Niu and S. Easterbrook, "So, you think you know others' goals? A repertory grid study," *IEEE Softw.*, vol. 24, no. 2, pp. 53–61, Mar.–Apr. 2007.
[27] S. Reddivari, Z. Chen, and N. Niu, "ReCVisu: A tool for clustering-based visual exploration of requirements," in *Proc. Int. Req. Eng. Conf*, Chicago, IL, USA, 2012, pp. 327–328.
[28] N. Niu and A. Mahmoud, "Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited," in *Proc. Int. Req. Eng. Conf* , Chicago, IL, USA, 2012, pp. 81–90.
[29] A. Mahmoud and N. Niu, "A semantic relatedness approach for traceability link recovery," in *Proc. Int. Conf Program Comprehension*, Passau, Germany, 2012, pp. 183–192.
[30] A. Mahmoud and N. Niu, "Using semantics-enabled information retrieval in requirements tracing: An ongoing experimental investigation," in *Proc. Int. Comput. Softw. Appl. Conf.*, Seoul, Korea, 2010, pp. 246–247.
[31] R. K. Yin, *Case Study Research: Design and Methods*. Newbury Park, CA, USA: Sage Publications, 2003.
[32] J. Wood and D. Silver, *Join Appl. Develop.* Wiley, 1995.
[33] L. Li, "Effects of enterprise technology on supply chain collaboration: Analysis of China-linked supply chain," *Enterprise Inform. Syst.*, vol. 6, pp. 55–77, Feb. 2012.
[34] S. Li, L. Xu, X. Wang, and J. Wang, "Integration of hybrid wireless networks in cloud services oriented enterprise information systems," *Enterprise Inform. Syst.*, vol. 6, pp. 165–187, May 2012.
[35] C. Wang, "Editorial: Advances in information integration infrastructures supporting multidisciplinary design optimization," *Enterprise Inform. Syst.*, vol. 6, p. 265, Aug. 2012.

Nan Niu (M'08) received the B.Eng. degree from the Beijing Institute of Technology, Beijing, China, the M.Sc. degree from the University of Alberta, Alberta, Canada, and the Ph.D. degree from the University of Toronto, Toronto, ON, Canada, all in computer science.

Currently, he is an Assistant Professor of computer science and engineering at Mississippi State University, Starkville, MS, USA. His current research interests include software engineering, requirements engineering, program comprehension, enterprise systems, and industrial informatics.

Li Da Xu (M'86–SM'11) received the M.S. degree in information science and engineering from the University of Science and Technology of China, Anhui, China, in 1981, and the Ph.D. degree in systems science and engineering from Portland State University, Portland, OR, USA, in 1986.

Dr. Xu serves as the Founding Chair of IFIP TC8 WG8.9 and the Founding Chair of the IEEE SMC Society Technical Committee on Enterprise Information Systems.

Jing-Ru C. Cheng received the Ph.D. degree in computer science from Penn State University, University Park, PA, USA, in 2002.

She has been a Computer Scientist at the U.S. Army Engineer Research and Development Center, Vicksburg, MS, USA, since 2002. Her current research interests include parallel algorithm development, software tool development for scientific computing, and multiscale multiphysics code development.

Zhendong Niu received the Ph.D. degree in computer science from the Beijing Institute of Technology, Beijing, China, in 1995.

He was a Post-Doctoral Researcher at the University of Pittsburgh, Pittsburgh, PA, USA, from 1996 to 1998, a Research/Adjunct Faculty Member at Carnegie Mellon University, Pittsburgh, PA, from 1999 to 2004, and a Joint Research Professor at the Information School, University of Pittsburgh, from 2006. Currently, he is a Professor and the Deputy Dean of the School of Computer Science and Technology, Beijing Institute of Technology. His current research interests include informational retrieval, software architectures, digital libraries, web-based learning techniques.

Dr. Niu was a recipient of the IBM Faculty Innovation Award in 2005 and the New Century Excellent Talents in University of MOE of China Award in 2006.