# Case Study, Interrupted: The Paucity of Subject Systems that Span the Requirements-Architecture Gap

Mamadou H. Diallo
Department of Informatics
Donald Bren School of Information and Computer Sciences
University of California, Irvine
mamadoud@ics.uci.edu

Susan Elliott Sim
Department of Informatics
Donald Bren School of Information and Computer Sciences
University of California, Irvine
ses@ics.uci.edu

Thomas A. Alspaugh
Department of Informatics
Donald Bren School of Information and Computer Sciences
University of California, Irvine
alspaugh@ics.uci.edu

## ABSTRACT

A number of approaches for spanning the requirements-architecture gap have been published in recent years, and we sought to rigorously characterize the gap and to conduct a comparative evaluation of approaches to span the gap using a case study method on a realistic problem. However, our intentions were impeded by the problem of finding appropriate subject systems that included sufficient information in both requirements and architecture document. Most subject systems that we found contained either detailed requirements or detailed architecture description, but not both. In this paper, we report on our search and the seventeen most suitable subject systems with the hope of aiding others undertaking a similar study. We speculate on the reasons for the paucity of suitable subject systems and invite contributions and suggestions for our ongoing work.

## 1. INTRODUCTION

In recent years, the gap between requirements and architecture has been recognized in the software engineering community and many techniques and approaches for bridging it have been proposed. However, the techniques and approaches are not based on a rigorous characterization of the gap. Furthermore, little effort has been devoted in evaluating these methods and approaches. Evaluation of methods and approaches in software engineering is necessary to advance research in the field and to provide evidence to promote adoption.

The need to evaluate approaches to span the requirements-architecture gap is particularly acute, because this is a technology that is pertinent to the development of large software systems. Consequently, subject systems that are realistic in size and composition are needed. The subject systems need to have not only well-documented requirements specification, but also good architecture artifacts. In this paper, our focus is directed toward the problems in finding subject systems that are suitable for analyzing the gap between requirements and architecture, and evaluating the methods for closing the gap.

We looked for subject systems in locales ranging from academia to industry. Our selection was based on two criteria. The first criterion was that the subject system must have a well documented requirements specification. Ideally, the system should have requirement scenarios that describe the functional, non-function, data, and interface requirements. If the requirements specification does not include the scenarios, it should be possible to derive the scenarios from the available requirements with ease. The second criterion was that the architecture of the system to include required and provided interfaces for each component in the architecture. If these two criteria are not satisfied, it would difficult to study the gap between requirements and architecture and evaluate existing methods and approaches for bridging the gap. These criteria were derived in part from the expected inputs to existing approaches to span requirements and architecture.

We looked at various sources for subject systems and examples. However, after many months of searching, we found a surprisingly small number. In this paper, we describe the seventeen software systems with both some forms of requirements specification and architecture artifacts. Of these, fifteen did not contain enough information for a case study or evaluation, and the two that contained sufficient detail were toy systems.

This paucity has led us to speculate on the reasons behind our apparent difficulties. One possible explanation is that current approaches to spanning the requirements-architecture gap are based on a misunderstanding of how systems are actually developed. Perhaps developers do not create detailed versions of both because doing so would introduce redundancy and the need to maintain parallel documents. Another possible reason is that spanning approaches are expecting the wrong kinds of documents as input. Perhaps requirements are created, but not as scenarios, and architectures are created, but not as class and component diagrams. One final reason that we consider is that projects that demand both detailed requirements and detailed architecture are produced in certain sectors with particular characteristics. For example, the defense and aerospace industries supposedly create detailed documents at each step of the process because there is a need for high availability and reliability, but due to security issues they do not make these widely available. We suspect the answer is a combination of the three.

The rest of the paper is organized as follows. Section 2 reviews prior efforts to develop subject systems and exemplars for evaluation requirements and architecture research. In Section 3, we motivate our study by reviewing technologies designed to span the requirements-architecture gap. We describe our method and the 17 best subject systems that we found in Section 4. In Section 5, we speculate on possible reasons for our difficulty in finding appropriate subject systems and present concluding remarks in Section 6.

## 2. RELATED WORK

Finding an appropriate subject system is essential to conducting realistic case studies. Large software systems developed by industry are preferred over toy problems developed by teachers or students. The subject systems are even more valuable if they can be turned into exemplars, i.e. shared and used in multiple empirical studies, thereby facilitating comparisons between tools. Much of software engineering has turned to Open Source projects for source code and data for empirical evaluations. However, Open Source projects are not suitable subject systems for evaluations of some kinds of technology, due to factors such as software process and team composition. In particular, Open Source projects cannot be used to evaluate approaches for spanning requirements and architecture, because they typically lack the necessary documentation. Consequently, the search for subject systems must turn elsewhere.

Researchers in requirements engineering and software architecture have recognized the need for common subject systems, or exemplars. There are a number of these available for either requirements or architecture, but rarely do they contain details about both aspects of the system.

There are many well-known requirements specification exemplars including the meeting scheduler, a patient monitoring system, a package routing problem, a turnstile problem, lift problem, production cell problem, and the generalized railroad crossing problem. Due to their importance in requirements specification, some studies have focused on evaluating the examplars themselves. Feather et al. [?] have investigated the purposes that a number of examplars can serve. These exemplars typically consist of a short description, e.g. a few paragraphs, and researchers use them to demonstrate the capabilities of a new technology. It should be noted that these are exemplars and not standardized subject systems. Researchers often add features to the problem to draw attention to breakthroughs in technology, e.g. specification for real-time or security. Studies such as the one by Wing [?], who used the library problem to compare twelve requirements specification approaches, are the exception rather than the rule.

The software architecture community has recognized the need for subject systems and exemplars more recently. It has been acknowledged that there is a lack of architecture artifacts from example or real systems that can be used to analyze and assess techniques and approaches in software architecture. There are two separate efforts, by Bredemeyer Consulting [?] and Grady Booch [?] to publish a collection of examples on a web site. The former is collecting software architecture case studies as well as architecture project artifacts. The main issue with the provided architecture artifacts in this website is that must of architecture diagrams show only a very high-level decomposition of systems, which are not very useful. The latter focuses on developing an architectural reference for software-intensive systems, a handbook of software architecture. This is a promising project, but still in its early days. There has been some limited use of standardized subject systems for evaluating architecture recovery tools in a structured demonstration format, one using the WELTAB III election tabulation system and organized by Elliot Chikofsky, and another using the xfig [?] drawing tool.

Despite the availability of some requirements and architecture exemplars, these subject systems have relatively few artifacts that can be examined. However, when it comes to systems with both requirements and architecture artifacts, to our knowledge, there has not been any effort to create, identify, or share appropriate systems. For analyzing and evaluating techniques and approaches developed to bridge the gap between requirements and architecture, example systems with complete requirements and architecture are necessary.

## 3. METHODS FOR BRIDGING REQUIREMENTS-ARCHITECTURE GAP

### 3.1 Requirements-Architecture Gap

Requirements and architecture have been traditionally treated as two separate processes in software development lifecycle. However, treating requirements and architecture as distinct activities has had widespread impact on tools and techniques that have been created, the research communities that have emerged, and more recent efforts to bridge the gap.

The separation of requirements and architecture into different phases can be traced back to the earliest software development process models (which in turn can be traced back to hardware development lifecycles). The process models for software development impose consistency and structure of the different activities by separating them into phases. In particular, the Waterfall Model [?] divides the software development lifecycle into requirements specification, design, implementation, testing/validation, integration, and maintenance phases. Requirements and architecture were placed in separate phases because it was felt that the former concerned with the problem domain while the latter concerned with the solution domain. The problem domain is the space in the world where the software system will live. The question of what is needed is addressed, but not how this can be achieved. The solution domain is the conceptual space where the system will be specified. It is here that questions related to how the software system will be built are addressed. As Grünbacher and et al. [?] pointed out, "part of the challenge in bridging requirements and architecture is due to the fact that they use different terms and concepts to capture the model elements relevant to each." This separation was made in order to articulate the difference between the problem and solution domains as a conceptual gap.

This division has led to the emergence of distinct technologies for requirements and for architecture. Not only have different techniques and approaches have been developed to model each phase separately, but also different notations have been used to document the models. Even in agile software development process models, such as Extreme

Programming [?], where there are no phases, different key practices have been developed for requirements and others for architecture. These differences in processes, techniques, and artifacts have resulted in a semantic gap between requirements and architecture.

Many researchers have argued that the separation of requirements and architecture causes problems in the overall development of software systems. Garlan and Perry [?] are among the first to point out that "Architectural design has traditionally been largely informal and *ad hoc*, which result in difficulties in communication, analysis, and comparison of architectural designs and principles". Similarly, Grünbacher and et al. [?] wrote, "Little guidance and few methods are available for the refinement of software requirements into an architecture satisfying those requirements". This view is also shared by others such as Chung et al. [?], who stated "One key task that remains a difficult challenge for practitioners is how to proceed from requirements to architectural design" and Rajasree et al. [?] who argued "Software development methodologies practiced today, fail to address the synergy between the requirement engineering process and architectural design". This approach of building architectures in an *ad hoc* manner is a source of errors that can lead to software systems that do not meet stakeholders' expectations. These problems are further evidence of the requirements-architecture gap.

## 3.2 Methods for Requirements-Architecture Gap

A number of methods for bridging the gap between requirements and architecture have been proposed. In general, the methods focus on improving the software architecture design process by making use of requirements specification directly. These methods can be grouped into two main categories: i) architecture generation methods based on requirements refinement; and ii) architecture analysis and evaluation methods based on requirements.

The first category aims to rationally choose the optimal architecture of a system based on the tradeoff analysis of requirements quality attributes, such as modifiability, security, and performance. Most of these methods are based on requirements scenarios and make use of iterative and concurrent development of requirements and architecture. An example of such an approach is ATAM (Architecture Tradeoff Analysis Method) [?], which models a set of quality attributes such as performance, availability, and security into measurable quantities that can be analyzed. The tradeoff analysis of the conflicting attributes helps to establish the right balance that leads to the best possible architecture in regard to requirements. Another example of this category is the CBSP (Component-Bus-System-Property) method [?]. CBSP proposes the design of an architecture by refining the requirements into an intermediate model and mapping this model into a set of architecture concepts. Other methods in this category include the ART-SCENE environment [?], the pattern oriented software development: moving seamlessly from requirements to architecture [?], and the method for moving from requirements to architecture design using goals and scenarios proposed by Liu and Yu [?].

The second category of methods is intended to assess the fitness of a given architecture with respect to its requirements. Most of the methods in this category are also based on requirements scenarios, where an architecture is evaluated against a set of scenarios. The method proposed by Bose [?], scenario-driven analysis of components-based software architecture models is good example of this category of methods. The method checks the consistency between requirements and architecture by analyzing the model of the architecture behavior (represented as a finite state machine) and the model of the requirements scenarios (represented in sequence diagrams). Other notable scenario-based architecture evaluation methods are SAAM (Scenario-based Architecture Analysis Method) [?], ALMA (Architectural Level Modifiability Analysis) [?], PASA (Performance Assessment of Software Architecture) [?], and ATAM (Architecture Tradeoff Analysis Method) [?], compared in a study conducted by Babar and Gorton [?]. ATAM is hybrid method as it can be used for both, designing and evaluating software architectures. The architecture analysis methods proposed by Nord and Soni [?], which introduces a global analysis of factors that influence software architectures, is an example of a method in this category that does not make use of scenarios.

For the above mentioned methods and others to be improved and adopted, they need to be evaluated with respect the goals they set to achieve. Good examples systems that can challenge the methods is a necessary condition for success of the evaluation.

## 4. SUBJECT SYSTEMS AND EXAMPLES

In the following sections, we describe the strategy we followed to find the subject systems, the results of the search and some observations resulting from the exercise.

## 4.1 Strategy for Finding Subject Systems

Despite the high number of software systems that have been developed and running, we found a surprisingly small number of good example systems after many months of search. In our search, we used two selection criteria. The first criterion was that the subject system must have a well documented requirements specification. Ideally, the system should have requirement scenarios that describe the functional, non-function, data, and interface requirements. If the requirements specification does not include the scenarios, it should be possible to derive the scenarios from the available requirements with ease. The second criterion demanded that the architecture of the system to include required and provided interfaces for each component in the architecture. If these two criteria were not satisfied, it would difficult to study the gap between requirements and evaluate existing methods and approaches for bridging the gap because all the required information for these purpose would not be available.

The 17 systems with the most detail in their requirements and architecture documents are summarized in Figure 1. Before going on describing these systems, we review some of the sources that we examined.

### *4.1.1 Published Empirical Studies.*

We looked at published empirical studies in requirements and architecture. The subject systems are typically not described in detail and difficult to use in a replication or in another study. They have either detailed requirements and general architecture or vice versa, or lack details of both. We have not selected any of those systems.

#### 4.1.2 *Academia.*

We looked at systems developed in academic settings and we found that they have very rudimentary requirements and architecture artifacts. An example of this types of systems is the Alloy Analyzer, developed by the Software Design Group at MIT or ArchStudio produced at the University of California, Irvine. They are mostly systems developed by researchers for a specific purpose, where documenting the processes is not very important. Occasionally researchers will create a subject system for use in an evaluation, but these tend to be small and limited in scope.

#### 4.1.3 *Industry.*

Finally, we looked at industrial systems. We found few systems that are publicly available. One large system we came across is "the Clouds and the Earth's Radiant System" developed by NASA. This system has large requirements documents, but does include all the functional, non-functional, data, and interfaces requirements. Furthermore, the architecture of the system is limited to class diagrams. This system and others are also summarized in Figure 1.

#### 4.1.4 *Bredemeyer Consulting.*

Another source we looked at for examples system is the Bredemeyer's website [?]. This website contains software architecture case studies and architecture project artifacts. The summary of the interesting examples we found in this site is included in Figure 1. In general, the architecture artifacts are limited to architecture diagrams that present the high-level decomposition of the systems. The architectural details as well as the requirements from which the systems the architectures are developed are rudimentary.

#### 4.1.5 *Software Architecture Handbook.*

Another website that promises to provide good architecture examples is Grady Booch's Software Architecture Handbook site [?]. This website is still under construction and has only a few architecture diagrams. If all the systems listed in this website are well documented, not only in terms of architecture but also requirements, it would be a very good source.

### 4.2 Search Results

The table in Figure 1 summarizes the best seventeen systems we were able to locate. We selected these systems because they all have some forms of requirements and architecture artifacts. The first column show the name of the system. The second column categorizes the systems based on their sizes, small, medium, or large. The categorization is based on both the available requirements and architecture design artifacts, for example, the functionality provided and the type of the systems. The third column gives the source of the systems. The explanation of the available parts in the requirements and architecture artifacts are explained in columns four and five respectively. As these two last columns show, only the AquaLush system from a textbook by Fox [?] and the PIMS system used in a textbook by Jalote [?] have complete requirements and architecture documents. The other fifteen systems are missing details about the requirements, architecture, or both.

We were able to find a long list of subject systems, but as we analyzed the artifacts they provided, the list became shorter and shorter. First, we eliminated all the systems that do not have some form of requirements specification, such as those in the Software Architecture Handbook. At the time of our investigation the website had a number of architecture diagrams for large systems without any mention of their requirements. We were hopping to find the requirements for these systems, but could not. Second, we eliminated all the systems that lack an architecture description, for instance, the specification exemplars. These examplars have been specified using different approaches, so we were hoping to find some kind of design for them, but were unsuccessful.

### 4.3 Observations

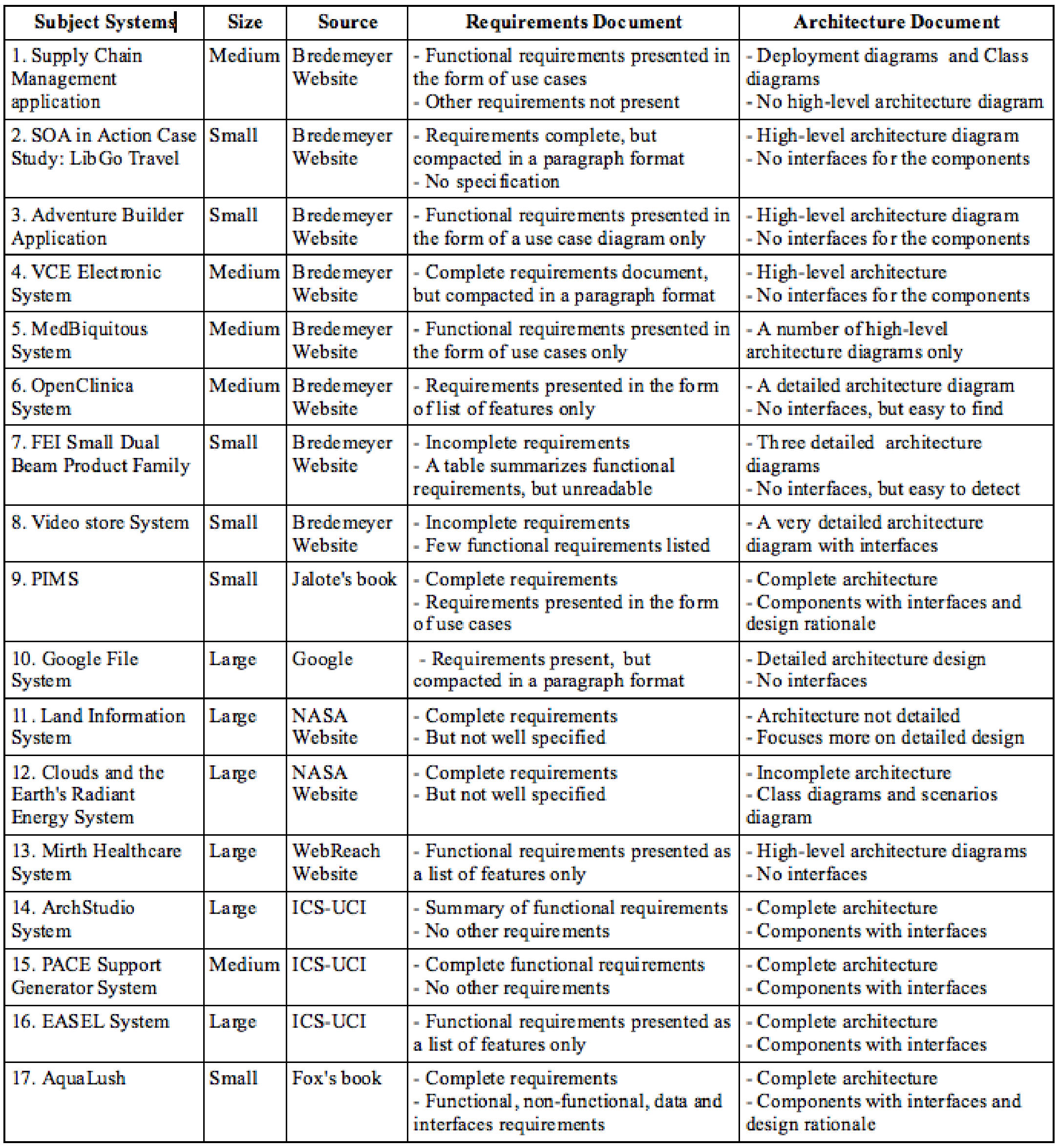
After arriving at this small list, we examined the subject systems for commonalities. We found that the systems could be place in one of four categories: i) toy systems (with detailed requirements and detailed architecture); ii) systems with detailed requirements, but sketchy architecture; iii) systems with detailed architecture, but sketchy requirements; and iv) systems with sketchy requirements and sketchy architecture. The only systems with both detailed requirements and detailed architecture that we found were the AquaLush [?] and PIMS [?] toy examples from texbooks. This suggested to us that the documents were not processual caches of knowledge, but rational reconstructions of knowledge. Even systems from large industry such NASA or Google did not have complete artifacts for both requirements and architecture. In the next section, we speculate on possible underlying reasons for the paucity of complete systems.

## 5. DISCUSSION

After months of searching, we were able to locate only 17 software systems with documentation on both requirements and architecture. This paucity of suitable subject system leads us to speculate on the possible reasons.

One possible explanation is that current approaches to spanning the requirements-architecture gap are based on a misunderstanding of how systems are actually developed. Perhaps developers do not create detailed versions of both because doing so would introduce redundancy and the need to maintain parallel documents. Of the 17 most complete systems, only two toy systems have complete requirements and complete architecture documents. The remaining systems contain either detailed requirements, but a sketchy architecture, or sketchy requirements and sketchy architecture, but never detailed versions of both. This result leads us to believe that there is an underlying connection between requirements and architecture through which, possibly, information in one may be redundant with information in the other.

Another possible reason is that spanning approaches are expecting the wrong kinds of documents as input. Perhaps requirements are created, but not as scenarios, and architectures are created, but not as class and component diagrams. The proposed techniques and approaches for bridging the gap between requirements and architecture are based on an implicit assumption about the nature of the gap. There aren't any studies that have attempted to characterize the gap explicitly. This potential misunderstanding can be addressed through empirical studies, such as the one we were attempting to conduct. However, based on the results of this study, it is apparent that conducting such empirical studies



| Subject Systems | Size | Source | Requirements Document | Architecture Document |
|---|---|---|---|---|
| 1. Supply Chain Management application | Medium | Bredemeyer Website | - Functional requirements presented in the form of use cases<br>- Other requirements not present | - Deployment diagrams and Class diagrams<br>- No high-level architecture diagram |
| 2. SOA in Action Case Study: LibGo Travel | Small | Bredemeyer Website | - Requirements complete, but compacted in a paragraph format<br>- No specification | - High-level architecture diagram<br>- No interfaces for the components |
| 3. Adventure Builder Application | Small | Bredemeyer Website | - Functional requirements presented in the form of a use case diagram only | - High-level architecture diagram<br>- No interfaces for the components |
| 4. VCE Electronic System | Medium | Bredemeyer Website | - Complete requirements document, but compacted in a paragraph format | - High-level architecture<br>- No interfaces for the components |
| 5. MedBiquitous System | Medium | Bredemeyer Website | - Functional requirements presented in the form of use cases only | - A number of high-level architecture diagrams only |
| 6. OpenClinica System | Medium | Bredemeyer Website | - Requirements presented in the form of list of features only | - A detailed architecture diagram<br>- No interfaces, but easy to find |
| 7. FEI Small Dual Beam Product Family | Small | Bredemeyer Website | - Incomplete requirements<br>- A table summarizes functional requirements, but unreadable | - Three detailed architecture diagrams<br>- No interfaces, but easy to detect |
| 8. Video store System | Small | Bredemeyer Website | - Incomplete requirements<br>- Few functional requirements listed | - A very detailed architecture diagram with interfaces |
| 9. PIMS | Small | Jalote's book | - Complete requirements<br>- Requirements presented in the form of use cases | - Complete architecture<br>- Components with interfaces and design rationale |
| 10. Google File System | Large | Google | - Requirements present, but compacted in a paragraph format | - Detailed architecture design<br>- No interfaces |
| 11. Land Information System | Large | NASA Website | - Complete requirements<br>- But not well specified | - Architecture not detailed<br>- Focuses more on detailed design |
| 12. Clouds and the Earth's Radiant Energy System | Large | NASA Website | - Complete requirements<br>- But not well specified | - Incomplete architecture<br>- Class diagrams and scenarios diagram |
| 13. Mirth Healthcare System | Large | WebReach Website | - Functional requirements presented as a list of features only | - High-level architecture diagrams<br>- No interfaces |
| 14. ArchStudio System | Large | ICS-UCI | - Summary of functional requirements<br>- No other requirements | - Complete architecture<br>- Components with interfaces |
| 15. PACE Support Generator System | Medium | ICS-UCI | - Complete functional requirements<br>- No other requirements | - Complete architecture<br>- Components with interfaces |
| 16. EASEL System | Large | ICS-UCI | - Functional requirements presented as a list of features only | - Complete architecture<br>- Components with interfaces |
| 17. AquaLush | Small | Fox's book | - Complete requirements<br>- Functional, non-functional, data and interfaces requirements | - Complete architecture<br>- Components with interfaces and design rationale |

**Figure 1: Summary of Subject Systems Artifacts**

is difficult due to the lack of appropriate subjects systems, leading to a circular chicken-egg problem.

One final reason that we consider is that projects that demand both detailed requirements and detailed architecture are produced in certain sectors with particular characteristics that demand both detailed documentation and prohibit their dissemination for research purposes. For example, the defense and aerospace industries supposedly create detailed documents at each step of the process because there is a need for high availability and reliability, but due to security issues they do not make these widely available. While our search has been focused on the Internet and the literature, we have also consulted our colleagues for leads. Our fellow researchers were aware of suitable industrial examples, but they were only available to researchers who had relationships with the corporation or government agency, and could not be shared.

We suspect our difficulties stem from a combination of the three reasons presented. In part they are due to a misunderstanding of software development in practice, part mismatch in expectations by approaches to spanning the requirements-architecture gap, and part peculiarities of systems that have detailed requirements and architecture documentation.

## 6. CONCLUSION AND FUTURE WORK

Our planned case study was aimed at acquiring a better understanding of requirements-architecture gap, so that we could subsequently conduct a comparative evaluation of tools and approaches for spanning the gap. To achieve this objective, we needed to find suitable, realistic subject systems. These subject systems had to satisfy two criteria. The first criterion was that the subject system must have a well documented requirements specification while the second criterion called for detailed architecture other than just a diagram showing the high-level decomposition of a system.

After many months of searching, we were unable to find results of subject systems satisfying our criteria. Among the only 17 best subject systems that we found, only two toy systems from text books contained sufficiently detailed requirements and architecture artifacts. The others did not contain enough information either in the requirements, architecture, or both, for a case study or evaluation.

This surprising outcome lead us to a further analysis resulting with some speculations on the reasons behind this phenomenon. We made the following speculations that we believe are the causes behind the lack of good subject systems: (i) current approaches to spanning the requirements-architecture gap are based on a misunderstanding of how systems are actually developed; (ii) developers do not create detailed versions of both because doing so would introduce redundancy and the need to maintain parallel documents; (iii) spanning approaches are expecting the wrong kinds of documents as input; (iv) projects that demand both detailed requirements and detailed architecture are produced in certain sectors with particular characteristics.

We believe also that for our speculations to be confirmed or rejected, further studies need be conducted on this matter. In the future, we intend to take this study further by appealing to industry. Obtaining real systems from industry from different disciplines in necessary to conduct more detailed studies of the requirements and architecture artifacts. Another possible direction of this study is to interview system developers in industry on the process they follow to go from requirements to architecture.

This need for subject systems that include details from both requirements and architecture is indicative of a larger problem, that of finding subject systems that include artifacts from multiple phases or activities in the software lifecycle. Such subject systems are needed to evaluate techniques such as specification-based testing, architecture-based testing, and assessing the quality of requirements by using the documents to write tests. We expect that efforts to locate subject systems for these studies would be mutually beneficial to our efforts to find subject systems for empirically studying the requirements-architecture gap.