Architecture and Design Intent: An Experience Report

Paul S Grisham, Matthew J. Hawthorne and Dewayne E. Perry
Empirical Software Engineering Laboratory (ESEL)
The University of Texas at Austin
{grisham, hawthorn, perry}@ece.utexas.edu

Abstract

As part of a graduate course on software architecture and design intent, we designed a class project in which teams of students performed software engineering tasks that required them to understand the design of an open source project and evolve the architectural design in response to a set of additional functional requirements. The students used intent-based design approaches and notation systems to document intent for architectural design features. We use the students' experiences with these methodologies to explore the potential usefulness of intent-based modeling approaches to system architecture, and also to gain insight into directions for further research.

1. Introduction

Software architecture has become the principal means by which requirements are transformed into a working, implemented system. Software architecture is an intermediate model of the transformation of the functional requirements into the implementation design. An understanding of the intent of design decisions – especially at the architectural level of abstraction – is critical to both initial development and system evolution.

The Perry/Wolf model of software architecture defines a software architecture as *elements*, *form*, and *rationale* [15]. The basic architectural elements are components and connectors. Form prescribes the basic structural relationships of the elements and constrains the properties of those elements. Rationale provides the justification for both the elements and the form. To date, software architecture research has focused mainly on methods of describing and reasoning about architectural elements and form. Research on architectural styles [1] and prescriptive models of software architecture [3,8] has progressed sufficiently to begin to allow software architects to move beyond merely using software architecture as a model for partitioning and organizing the implementation process and to allow software architecture to be used to optimize the design process. Unfortunately, rationale was only vaguely defined in the original paper, and interest in software architecture rationale as a research topic has languished until very recently.

Software design is an ill-structured problem [17]. Prior studies of designers showed that experienced designers approach initial software design, not as a linear process of rationally considered steps, but as an exploratory process, solving smaller problems opportunistically [8, 9]. Although some have proposed that software design be treated as a rational process [14], we believe that software design is the iterative solving of small, known problems and the use of those decisions to constrain larger, open-ended problems. Moreover, other work suggests that designers approach evolutionary design by asking critical questions about the relationships between design elements in the existing design and alternative choices [16]. We use the term design intent to describe the set of decision-making factors that software designers apply when designing a system. This includes not just design rationale, but also the application of prior best-practices – such as styles, patterns, and idioms – and models for naturalistic decision-making and situational analysis.

To explore software design intent, we developed a novel graduate course on software architecture rationale, "Software Architecture and Design Intent," at the University of Texas at Austin during the spring semester, 2006. The course was organized as a topical seminar course focusing on how design intent can be expressed and used in software architecture. It presented a historical context for software design, software architecture, and design rationale and then covered current research on how software designers express their intent and how design intent may be reused. One of our main goals for the class was to develop a unified view of decades of software design research with respect to expressing and reusing design intent and design rationale.

We designed the class project as a preliminary study into designing a case study on the effectiveness of design intent documentation systems. To this end, we exposed the students to several methods of discovering

and documenting design intent and asked them to provide feedback. Students documented and analyzed the architecture of a large, mature, well-documented software system with respect to how design decisions had been captured in the architecture. Our approach consisted of three steps:

1. Students inferred design intent from a system design without structured documentation;
2. Students engaged in a design activity, using a structured documentation system without a decision-support method; and
3. Students engaged in a design activity using a decision-support method.

We hoped that we would be able to gain a better foundation for empirical studies of software design rationale and design intent systems, and how to create a repeatable study that could be used over a period of time to improve existing documentation and decision-support methods.

This paper presents the results of that project from the perspective of how rationale might be used in evolving a large software project. Section 2 presents the project design, including a set of questions we had hoped to answer over the course of the semester. Section 3 presents the project results and the feedback we received from students on how they applied the concepts from the course to the analysis activity. Section 4 analyzes our results and attempts to carry these results forward into future research into software architecture rationale. Section 5 approaches the project from an education perspective, and presents ideas for improving the project, both in terms of improving the validity of the study and improving the teaching of the concepts of architecture rationale and design intent. We present a summary of our results and our conclusions in Section 6.

2. Class Project

2.1 Project Overview

The primary goal for the students was to analyze the architecture of DSpace, a document repository system for managing an online library of documents and publications [4]. DSpace is a joint product of the M.I.T. library system and Hewlett-Packard. DSpace was initially designed to manage collections of research papers and technical reports, though it is flexible enough to be used in a wide range of applications.

DSpace was chosen for our project for several reasons. DSpace is sufficiently large that no single student or student team could comprehend the entire system over the course of a semester. We wanted our students to think architecturally and be efficient in identifying relevant information from the documentation. DSpace is open-source, so the source code was available if students needed it, though we did not encourage reading sources.

DSpace is also mature, and represents the work of both industrial and academic developers. Design complexity is representative of medium to large-scale software projects. We wanted to avoid using a project that was tailored to the concepts we were presenting.

DSpace is also well-documented. The nature of the joint collaboration of DSpace means that there is a large collection of documentation available from the DSpace website. Students were encouraged to use these documents to answer their questions about the design of the system. There is no formal System Requirement Specification (SRS) or System Design Document (SDD) in the online documentation, but the documentation is highly accessible and presents a thorough overview of the design [5]. In addition to the design documentation, DSpace has a large amount of unstructured documentation, such as Wikis, message boards, and email archives. Specific issues may be found by searching over these resources.

Once students had a general understanding of the DSpace architecture, they were required to consider how to add a set of new functional requirements to DSpace. These additions required changes to the high-level architecture of the system. Students were not required to implement the new system, but only to analyze the impact of the proposed changes.

2.2 Project Description

Students self-formed groups of 2-4 students. All students had basic familiarity with software design and architecture. Several students had extensive industrial experience. The project was organized into phases. The phases followed the topics of the lectures to reinforce the concepts from the class and results from one phase were used the next. Each phase was also designed to address specific questions about how design intent is expressed and used during the project activities. These questions will form the basis for our discussion of the results in section 3.

2.2.1. Using Existing Architectural Information

Phase 1 required students to read through the online documentation for DSpace and become comfortable with the basic functional requirements and high-level design of the system. Students were required to answer a series of targeted questions about the suitability of the existing documentation to discovering the design intent of the existing architecture. Our primary goal in Phase 1 was to better understand how design intent is captured and used in informal documentation.

Q1: *Where did students identify sources of functional intent, architectural design and architectural rationale knowledge? Which sources were useful?*

Q2: *Were students effectively able to identify relationships between architectural entities in the existing product? What did they miss?*

Q3: *How was design rationale expressed explicitly and implicitly in informal documents? What semantic clues or other means did students use to identify architectural or design rationale knowledge?*

2.2.2. Using QOC Refinement

In *Phase 2*, students were given a new set of functional requirements to add to the existing DSpace design that adds a paid subscription service to the existing DSpace system. The new system should allow some documents to be freely available, while others are only accessible to paid members of the subscription service. The requirements specify a new access model, as well as member management, payment processing, and a new notification system for communicating with paid subscribers about updates to subscribed articles.

Some requirements were left vague or incomplete intentionally so that students would have to negotiate requirements and discuss their impact to the final product. Students were instructed to use QOC to assist them in negotiating the requirements and to document their evaluation of the impact of various design alternatives. QOC is a semi-formal system of representing alternatives and rationale as questions, options for answering those questions, and criteria for evaluating the suitability of those options [12]. Our primary goal in Phase 2 was to explore how explicit models helped negotiate requirements and facilitate decision-making.

Q4: *Did QOC help negotiate options, expose false or incomplete assumptions, or assist in team coordination? Was it cost-effective?*

2.2.3. CBSP Profile Analysis

In *Phase 3*, students were asked to use CBSP, a process for guiding architectural design that refines high-level functional requirements into architectural elements and properties [6,7]. In CBSP, functional and non-functional requirements are classified as a *C*omponent element or property, as a *B*us (or connector) element or property, or as a *S*ystem *P*roperty. Students were required to build a CBSP relevance profile which required that students generalize or merge their refined requirements into elements and properties according to the CBSP taxonomy. Our primary goal in Phase 3 was to explore how design intent is captured in intermediate models during the architecture derivation process.

Q5: *Is it useful to build an intermediate model between requirements and architectural design (i.e., architectural elements and relationships)?*

2.2.4. CBSP Refinement

In *Phase 4*, students continued using CBSP to continue refining their model by merging and generalizing CBSP elements and identifying requirements that did not map cleanly to the CBSP taxonomy. Although CBSP also provides a means of arbitrating an architectural style from a decomposition of functional requirements, we did not use this feature of CBSP. We were instead interested mainly in its methods of requirements decomposition. Our primary goal in Phase 4 was to explore the process of creating architectural elements from refined requirements.

Q6: *Were students able to transform requirements into a set of architectural entities to be evolved or added to the system design?*

Q7: *What requirements did not conform cleanly to architectural entities and relationships?*

2.2.5. Project Evaluation

A final phase, *Phase 5*, required students to complete their model refinement and submit a final project report. In addition to addressing issues from the previous phases, the students were encouraged, though not required, to show the association between their refined architectural elements and the original DSpace architecture. In addition to the final report, we conducted a final interview with students to gain their feedback on using the various techniques from the class and on applying architectural design intent in a system evolution context. The student reports and the transcripts of that interview form the basis for the qualitative analysis presented in the next section.

3. Results and Student Feedback

We designed the class project to obtain qualitative evidence to help in answering our research questions by requiring students to answer specific questions and provide feedback about their experiences while working through each phase of the project. The students' responses gave us insight into how useful intermediate intent models between requirements and architectural design are in evolving system architecture. To encourage students to provide useful feedback and hone their engineering analysis skills, we encouraged them to provide critical analysis about the usefulness of the architectural documents and approaches used in the class. In the remainder of this section, we summarize the students' responses for the project activities for each of our original research questions.

3.1. Using Existing Architectural Information

3.1.1. Q1. The DSpace system documentation includes a fairly complete, if informal, set of functional specifications and high-level architectural design documents. These documents provided useful information about functional and architectural intent.

Using the documentation, four of the six student teams identified the four major categories of functional intent (submit, archive, disseminate, manage/administer). One of the remaining teams identified similar functionality from a high-level functional description aimed at end-users (i.e., missing administration functionality); while another team listed architectural entities from the DSpace architectural diagram, including implementation-specific entities like "workflow engine". The documentation included major areas of functional intent, but the information was somewhat disorganized, spread across overlapping and conflicting sources.

All of the student teams were able to identify the basic 3-tier layered architectural style of DSpace. However, design rationale information proved to be more difficult for the students to locate. The DSpace project documentation does not include any explicit documentation of design intent or rationale. Although one team reported that they failed to find any information about architectural rationale, most found some architectural rationale information in unstructured documentation, such as online design discussions or Wiki articles about system components. Implicit rationale information was also found in the internal reference specification. In a few cases, students were also able to infer design rationale by reasoning forward from requirements or backward from architectural features.

The wide variation in how successfully different teams managed to locate and recognize design rationale in the existing documentation was due to a combination of the non-linear way in which students happened to follow through the hyperlink paths in the online system documentation and the team's level of effort. To the extent that student searches through the documentation were disorganized, we expected that there should have been a positive correlation between level of effort and probability of success. However, in this case it is equally likely that the poor organization of the unstructured documentation confused some of the students enough to adversely impact their chances of locating relevant information.

Most importantly, no degree of organization or effort will be effective if the searchers are unable to recognize the information when they find it. It appears that some students may have failed to recognize informal sources of architectural rationale information embedded in enhancement proposals and discussions about system functionality or design. It is difficult to locate or identify design rationale information in informal, unstructured documentation of use to system designers or maintainers.

3.1.2 Q2. Students were able to identify the basic architectural structure of the system, and most teams also provided some information about the nature of the architectural entity relationships. Some teams chose to characterize dependency relationships as functional or data dependencies, while others characterized them as temporal or logical.

3.1.3. Q3. Design rationale, where it existed, was expressed as descriptive text. Since rationale is informal, semantic clues include any statements that express some kind of logic or reasoning explaining why a particular component or architectural design pattern was selected. In some cases, a declarative comment on what style or pattern was selected provided clues about the design context and the designer's intent. Textual clues include any statement addressing "why" any aspect of the architecture was designed a certain way, any mention of architectural styles or patterns, and any justification of why a given third-party component was selected or rejected.

3.2. QOC Refinement

3.2.1. Q4. Most students found using QOC to be an intuitive process that helped them formalize their design decisions. Students found QOC to be helpful for exploring the design space because it forced them to consider alternatives. However, QOC lacks any mechanism for ensuring that designers consider optimal solutions and the students also found it to be more detailed than they needed in their negotiations, so that any design benefits were outweighed by the cost in time and effort.

Another factor that affected the QOC refinement process was the uneven quality and availability of DSpace architectural and design rationale information for different parts of the system. When design rationale information was available for parts of the system related to the extensions they were designing, students said the information helped them select better design options during QOC refinement, while the lack of such information made it more difficult to select optimal solutions during QOC.

One student suggested that performing requirements analysis before QOC, CBSP or similar post-requirements methodologies would enhance the value of the QOC process. Another student suggested partitioning requirements into separate QOC models to make it easier to discover more sub-questions.

Another suggestion was to perform QOC iteratively. One team reported that they obtained very positive results when they performed multiple QOC iterations in which they questioned assumptions and results from earlier phases, and even "questioned the questions". This enabled them to refine their design decisions using domain knowledge they gained in earlier iterations. They were then able to use QOC to help expose earlier false assumptions.

The most common suggestion for improving QOC itself was to add tool support. A tool that supported and enforced QOC model development during design space exploration and refinement would probably go a long way toward making QOC more attractive from a cost-effectiveness standpoint.

3.3. CBSP Profile Analysis

3.3.1. Q5. An explicit goal of CBSP is to "recast and refine" requirements into an intermediate model that will facilitate mapping the requirements to architectures [7]. Information gleaned from the CBSP phases of the class project relates directly to research about intermediate models between requirements and architecture. CBSP was useful as a framework for transforming requirements into architectural features. The CBSP subproject highlighted some of the potential benefits of building intermediate models between requirements and system architecture.

Many of the problems the students experienced with CBSP were caused by specific characteristics of the CBSP process, and impacted our ability to generalize our observations about the utility of intermediate model approaches to architectural design. Ambiguity and disagreement within project teams related to the 4-value CBSP relevance scale appeared during the CBSP Profile Analysis phase. Students had difficulty distinguishing between relevance levels – e.g., level 1 "partially" and level 2 "largely" – causing wide disparities between profiles that did not reflect actual disagreement.

As with QOC, students felt strongly that tool support would make CBSP a lot more user-friendly and cost-effective. They also said that the CBSP papers [6,7] were too vague about certain practical details to be used as an instruction manual. More annotated examples demonstrating CBSP profile analysis of relatively complex requirements that impact and interact with different subsets of the system architecture in a variety of ways would be especially helpful for students or practitioners using or evaluating CBSP.

3.4. CBSP Refinement

3.4.1. Q6. Results from the final CBSP refinement phase reflected the different engineering backgrounds of the students, with the most significant differences seen between students who had strong industrial software engineering backgrounds and those who did not. Software engineers who were already familiar with architectural styles and design patterns had some basis for understanding what it means to select architectural styles based on requirements. Students with little experience or who were from non-software backgrounds had more difficulty understanding the CBSP terminology. For example, the hardware engineers in the class pointed out that they have a very different understanding of "bus", "component", or "system" from how those terms are used in software architecture.

Some students indicated that they felt hampered by their lack of a formal or practical background in architectural styles and patterns. They suggested that providing more examples of architectural patterns and rationale would have helped them with the final stages of CBSP, where the CBSP categorization of requirements is used to guide the transformation of the requirements into sets of architectural entities, relationships and styles.

A number of students also indicated that the four CBSP properties were not sufficient for designing an architecture. The CBSP authors themselves acknowledge that their simplified set of properties is a known limitation of the system [7].

For simplicity, our class project instructions emphasized functional requirements (goals) over non-functional requirements (constraints). As a result, some of the more experienced students said they didn't know what system properties to design for. In actual development projects, nonfunctional requirements like dependability, extensibility, maintainability, security, and even cost, usually have a significant impact on the architectural design.

One idea that came up several times during the class project feedback discussion was to enhance CBSP and similar approaches with some kind of ontology of technical criteria to help guide the transformation from requirements to architecture.

3.4.2. Q7. To keep the scope of the project manageable, we did not explicitly include security concerns in the evolutionary requirements for the class project, but some teams considered security requirements anyway, because they logically follow from the explicit requirement that the subscription system support credit card payments. Cross-cutting concerns like security often blur the lines between the

CBSP categories. Except in rare cases where the functional requirements footprint of the security requirement is strictly limited, cross-cutting concerns can end up giving all the affected requirements and resulting architectural entities a "system-wide" flavor.

4. Analysis of Results

Students had little trouble understanding the basic DSpace system architecture from the system documentation, but some had problems identifying the basic units of functional intent or the semantics of architectural entity relationships. All of the teams identified the lack of architectural rationale information as an issue, although some teams managed to locate rationale information implicitly contained in unstructured documentation such as Wiki documents and informal project communications. The also attempted to infer rationale from functional or architectural information. The main obstacles to understanding the existing system included disorganization, lack of global searchability, and missing, duplicate, or contradictory information.

For some students, these difficulties may have been exacerbated by confusion about core concepts like functional and architectural intent, design rationale, etc. This indicates that we need to

(1) focus on more effective ways of organizing architectural information, including global meta-organization like topical indexes and search engines;
(2) focus on the right vocabulary, examples and metaphors to explain functional and architectural intent, design rationale and other architectural design concepts; and
(3) focus on the most effective means to present architectural information, including graphical diagrams for any entity relationships.

The lack of tool support was a major issue for both QOC and CBSP. A reasonable level of tool support is absolutely critical in order for any design methodology to be user-friendly and cost-effective enough to achieve widespread use,

5. Project Design Discussion

The student feedback we received highlighted several ways we could improve the class project to level the playing field for students and eliminate potential sources of uncertainty. We expected that our graduate students had enough background with software architecture concepts and techniques to be able to apply the advanced concepts from the course. We could have provided more detailed instructions about how to identify properties of architectural entity relationships in the existing system and how to model evolutionary changes to the design.

We neglected to take educational and industrial experience into consideration when allowing students to form into teams to complete the project. We could have been more thorough in providing background information about software architecture styles and patterns to any students who needed exposure to high-abstraction-level approaches to system design. We found when we considered industrial experience, the quality of the final report, both in terms of the quality of the work product and the quality of the analysis and criticism of suitability of the methods we considered was significantly greater for those students with significant work experience. We also found that students with significant work experience were also more likely to appreciate the benefits of intent rationale modeling but also to be extremely skeptical of the practical applicability of intent modeling.

The selection of QOC as a rationale model and CBSP as an architecture derivation technique influenced our final results. We felt that the learning curve on both systems was sufficiently low to allow students to rapidly learn them and start applying them to our project in a single semester. CBSP was not as suitable for a maintenance environment as we had hoped, and we could have used other, alternate approaches, or allowed students the choice of what approaches they preferred and then compared the results. For instance, we could have used KAOS [13] to perform requirements refinement and Preskriptor [2] to perform the architecture derivation. We also considered the use of an architecture language with explicit rationale modeling, such as Archium [11].

One significant consequence of the limitations of conducting this project in a single semester is that we did not have the time to tie the final work product back to the original DSpace project documentation. We wanted to have the students answer the Phase 1 questions about identifying sources of design intent and architecture rationale for their final work product and compare those answers to the Phase 1 answers. The students simply did not have time by the end of the semester to perform any additional analysis beyond completing the CBSP work products.

We selected the use of DSpace as a problem domain mainly for its realistic complexity and the availability of high-quality informal documentation. The use of a real project, rather than a simple problem, allows us to generalize the results of this project to better understanding how to guide future research into architecture and design intent. While we attempted to make the problem space as realistic as possible, we chose not to emphasize applicable nonfunctional constraints like security and dependability. Including

these types of requirements in a more rigorous case study or controlled experiment would be invaluable in making the project requirements more realistically complex and provide additional motivation to continue advancing the state of the art in intermediate modeling.

6. Conclusion

Our experiences of trying to teach architecture rationale and design intent to graduate students in the context of our class project revealed some interesting issues related to the software design process in general, and in modeling design intent knowledge in particular. Generally, our students had difficulty in understanding the benefits of intermediate intent modeling during software architecture design compared to the relative costs. We must develop ways to measure the added value of explicit intermediate architecture and intent modeling compared to the relatively cheap approaches of informal and unstructured documentation. Our study simply did not have time to perform the comparison between the original documentation and the final work product. Moreover we strongly believe that we must perform more comparisons between alternate requirements representations, architecture derivation techniques, and intermediate modeling notations.

7. Acknowledgments

We would like to thank the students in the inaugural section on "Software Architecture and Design Intent." Their hard work and constructive feedback made the class a success and a joy to teach. This research was supported in part by NSF CISE grant CCR–0306613 "Transforming requirement specifications into architectural prescriptions."

8. References

[1] Bhattacharya, S. and Perry, D.E. Predicting Architectural Styles from Component Specifications - Extended Abstract *5th IEEE/IFIP Working International Conference on Software Architecture (WICSA)*, Pittsburgh, PA, 2005.

[2] Brandozzi, M. and Perry, D.E. From Goal-Oriented Requirements to Architectural Prescriptions: The Preskriptor Process *International Workshop From Software Requirements to Architectures (STRAW'03)*, Portland, OR, 2003.

[3] Brandozzi, M. and Perry, D.E. Transforming Goal Oriented Requirements Specifications into Architectural Prescriptions *Workshop from Software Requirements to Architectures (STRAW1)*, Toronto, CA, 2001.

[4] DSpace Federation. Project Home Page. http://www.dspace.org/

[5] DSpace Federation. DSpace Internal Reference Specification: Technology & Architecture. http://www.dspace.org/technology/architecture.pdf

[6] Grunbacher, P., Egyed, A. and Medvidovic, N. Reconciling Software Requirements and Architectures with Intent Modeling. *Software and Systems Modeling*, *3* (3). 235-253.

[7] Grunbacher, P., Egyed, A. and Medvidovic, N. Reconciling Software Requirements and Architectures: The CBSP Approach *5th IEEE International Symposium on Requirements Engineering (RE 2001)*, Toronto, Canada, 2001.

[8] Guindon, R. Designing the Design Process: Exploiting Opportunistic Thoughts. *Human-Computer Interaction*, 5 (2&3), 305-344.

[9] Guindon, R., Curtis, B., and Krasner, H. A model of cognitive processes in software design: An analysis of breakdowns in early design activities by individuals. *2nd Workshop on Empirical Studies of Programming*, Washington, D.C., 1987.

[10] Hawthorne, M.J. and Perry, D.E. Exploiting Architectural Prescriptions for Self-Managing, Self-Adaptive Systems: A Position Paper *ACM SIGSOFT Workshop on Self-Managed Systems (WOSS'04)*, Newport Beach CA, 2004.

[11] Jansen, A.G.J. and Bosch, J. Software Architecture as a Set of Architectural Design Decisions *5th IEEE/IFIP Working Conference on Software Architecture (WICSA 2005)*, Pittsburgh, PA, 2005.

[12] MacLean, A., Young, R.M., Bellotti, V.M.E. and Moran, T.P. Questions, Options, and Criteria: Elements of Design Space Analysis. *Human-Computer Interaction*, *6* (3&4), 1991, 201-250.

[13] Massonet, Ph., van Lamsweerde, A.: Formal refinement patterns for goal-driven requirements elaboration. In: *FSE-4 - 4th ACM Symposium on the Foundations of Software Engineering*, San Francisco, ACM Press, 1996, 179–190

[14] Parnas, C.L. and Clements, P.C. A Rational Design Process: How and Why to Fake It. *IEEE Transactions on Software Engineering* 12 (2), 251-257

[15] Perry, D.E. and Wolf, A.L. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, *17* (4). 40-52.

[16] Sillito, J., Murphy, G. and de Volder, K. Questions Programmers Ask during Software Evolution Tasks. *14th ACM SIGSOFT Symposium on Foundations of Software Engineering*, Portland, OR, 2006.

[17] Simon, H.A. The Structure of Ill-Structured Problems. In *Developments in Design Methodology*, Cross, N. (ed.), 1984, 317-327.