In Proc. OOPSLA 2006, ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications. October 22-26, Portland, Oregon, USA.

Software Architecture at a Large Financial Firm

George Fairbanks
Carnegie Mellon University
School of Computer Science
5000 Forbes Avenue
Pittsburgh, PA, 15213, USA
fairbanks@cmu.edu

Kevin Bierhoff
Carnegie Mellon University
School of Computer Science
5000 Forbes Avenue
Pittsburgh, PA, 15213, USA
bierhoff@cmu.edu

Desmond D'Souza
Kinetium, Inc.
9901 Spicewood Mesa Drive
Austin, TX, 78759, USA
desmond.dsouza@kinetium.com

ABSTRACT

System builders have historically used informal software architecture models to understand options, make choices, and communicate with others. Research into software architecture over the past fifteen years has indicated that more precise architecture models may be beneficial. At a large financial firm, we applied precise software architecture techniques on four software projects and this experience has revealed a number of practical issues. We made the following observations across the projects: 1) Architecture models can be used to bridge gaps between business requirements and technology, 2) A small collection of techniques and a detail knob are practical and useful in a variety of projects, 3) Architecture modeling techniques amplify the skills of the architects, 4) A model of domain concepts and relationships is helpful when building architecture models, and 5) It is difficult to know when to stop adding detail to your architecture model. We believe that these observations motivate future research and can help practitioners make software architecture more effective in practice.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Design, Documentation

Keywords

Experience report, finance

1. INTRODUCTION

Daily operations at many companies rely on services provided by complex enterprise software systems. Software helps or even enables many companies to do their business but software is usually not their area of expertise. Conversely, software engineers understand software but typically not the business it is written for. This disconnect has to be addressed when building or integrating enterprise software. Success requires effective collaboration of software engineers and subject matter experts to ensure that the software being created actually provides the services needed by the business.

Software architecture [14] promises to aid this difficult task. Architecture has been a focus of software engineering research for fifteen years [8] and researchers have identified various benefits of incorporating software architecture into software development projects, including reduced cost of development [1].

In our view, software architecture involves modeling the software being built at a high level, thus expressing the domain, goals (or requirements), architectural structure, and behavior. As such, it addresses some of the classic challenges of software engineering. For instance, Sommerville includes lack of clarity, requirements confusion, and requirements amalgamation as common problems in system requirements ([15] p. 127). Informal requirements may appear clear to the subject matter expert because of her domain knowledge but the software engineer, lacking domain knowledge, needs a more precise specification.

Practitioners are starting to apply software architecture on industrial projects [5, 12]. This paper reflects on our experiences over the past year with applying software architecture techniques at a large financial company. This company recently decided to employ precise software architecture techniques based on object-oriented principles in the early stages of their projects. The firm hopes to improve its existing practice for developing software by using a more precise approach that leverages modern results of architecture research and practice.

We worked on four projects of significant size and importance alongside company employees who had not previously used these techniques. These four projects are noteworthy for their different natures. The first was a greenfield project, unencumbered with legacy code, while the second was a brownfield project to enhance an existing system. The third project was focused on vendor product selection and integration. The fourth was an overarching project whose goals were to coordinate the efforts of three others and communicate this design to senior management.

Most technologies must be adapted from the pure research before they can be applied in industrial settings [11]. Our architecture modeling technique is a synthesis of ideas from practice and academic research. Four elements form the backbone of the technique: goals models, component and connector models, information models, and behavior models. The models are based on objects and their interactions and rely on notations such as the UML [13]. The four models are tightly interconnected and thereby allow the architect to cross-check her models for completeness and consistency, yielding more precise models. Details on our architecture modeling technique are presented in section [3].

Precise software architecture models were effective in uncovering problems with designs in progress. Our experience herein sup-

OOPSLA'06, October 22–26, 2006, Portland, Oregon,USA.

ports hypotheses from the research community [14] but we found ourselves confronted with practical challenges. What is the role of software architecture in a company whose area of expertise is not software? Is the software architecture for a brownfield project the same as for a greenfield project? Is a systematic approach to software architecture useful (compared to just doing the best design we can)? Are there effective sanity checks for our understanding of the system to be built, given that domain experts only have limited time to validate our models? Where does architecture end? We made specific observations on each of these challenges. In section ?4 of this report, we describe anecdotal evidence for the following themes that we found to be true across the four projects.

- Architecture models can be used to bridge gaps between business requirements and technology.
- A small collection of techniques and a detail knob are practical and useful in a variety of projects.
- Architecture modeling techniques amplify the skills of the architects.
- A model of domain concepts and relationships based on object-oriented principles is helpful when building architecture models.
- It is difficult to know when to stop adding detail to an architecture model.

Our experience, even though preliminary and incomplete, is notable for three reasons. First, it demonstrates how research results on software architecture can be applied in an industrial setting. Second, it can motivate future research. Finally and most importantly, we believe that our observations can guide practitioners in their own efforts to apply software architecture. While we do not believe software architecture will be a silver bullet, our experience indicates it is an improvement over current practice.

The remainder of this paper is organized as follows. Section **??** gives an overview of the technique we used. The projects we worked on are introduced in section **??**. Section **??** provides evidence for our specific observations. Limits of these observations are investigated in section **??**, and we conclude in section **??**.

2. ARCHITECTURE MODELING TECHNIQUE

Our architecture technique is primarily a synthesis of existing published techniques. Its four primary parts are: a goals model, a component and connector model, an information model, and a behavior model. We specifically avoid prescribing a project management style even though our preference is to apply this technique in an iterative process. We view software architecture as an engineering task to be completed regardless of the team organization or the sequence of construction.

2.1 Sources

Our architecture modeling technique is best seen as a synthesis of existing modeling techniques and applied to the domain of software architecture. Many challenges of software architecture have been addressed in other contexts and it is natural to choose from known-good approaches.

The treatment of domain concepts follows from the precise modeling of objects in Catalysis [4]. In general, our modeling of domain types has not been as detailed as in Catalysis but it is reassuring to know the depth is there if needed.

The treatment of components and connectors is based on the central ideas from Shaw and Garlan [14]. The pragmatic application of these ideas to UML2 [13] is taken from the work of Cheeseman and Daniels [2]. Component and connector models were generally drawn as UML2 composite structure diagrams.

Behavior models in the form of scenarios are taken from Catalysis while Role Activity Diagrams (RADs) are from Oulds business process modeling [10].

The use of goals models patterned after KAOS [9] and allows the expression of competing architecture desires as in ATAM analyses [7], as well as items more in the business domain than the software domain. The goals models also contain concepts from Jacksons problem frames [6], specifically to structure the goals models and connect them with the domain types.

2.2 Elements

Four models provide the backbone of our architecture technique. A goals model expresses the highest level intent of the system. A component and connector model expresses the runtime entities in the system. The information model expresses the vocabulary for the other models, including types found in the domain. The behavior model expresses the dynamics of the system as it performs its intended functions.

While these four models provide the backbone to express the functionality of the system, other models are added as necessary to cover other quality attributes such as security or transactions. The additional models can use sophisticated domain-specific modeling notations or can be as simple as some ad hoc tables in a spreadsheet. The four elements are described in the following sections.

2.2.1 Goals model

The highest level goal expresses the reason for the systems existence. Each goal is decomposed into sub-goals and domain properties that collectively achieve the goal. This hierarchical decomposition proceeds until the sub-goals are small enough to be directly accomplished.

Obstacles to accomplishing goals are also captured in the hierarchy. Strategies for overcoming the obstacle are expressed with additional goals in the goals model.

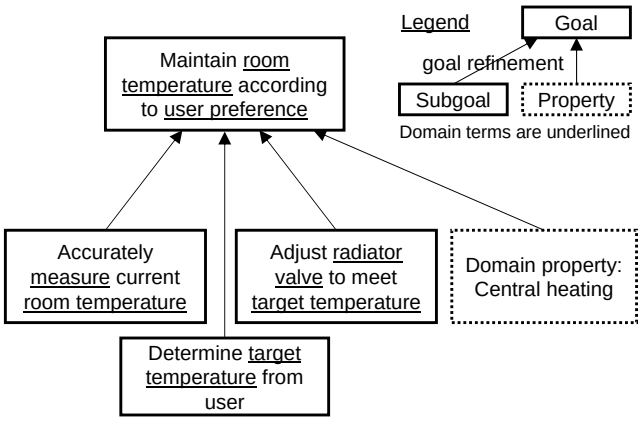

Figure 1: Example goals model

A set of sub-goals is assumed to be conjoined to achieve satisfaction of a goal, but annotations can be used when sub-goals represent competing strategies. In practice, our evaluation of goal satisfaction is subjective and yields yes-no decisions. The work on KAOS describes an objective technique to evaluate partial goal satisfaction that provides additional analysis capacity at the expense of greater

effort. There are additional techniques to evaluate alternative strategies based on degree of goal satisfaction using a combination of domain-based and balanced-score-card-based approaches, trading off more objective analysis with greater effort.

Domain properties are facts and assumptions about the domain that support the analysis of the goals model. Terms and relationships present in the goals are expressed in the information model (see section 2.2.3 below).

Ideally the goals model would form a simple tree but it is often the case that a sub-goal may support more than one higher-level goal. Michael Jacksons example of the skin of a rocket being used to provide an aerodynamic surface as well as a container for the propellant shows how one sub-goal can satisfy two goals. In these cases we attached the sub-goal to multiple parent goals.

The goals models can be represented textually, using a simple indented view in a word processor, or graphically, using a box-and-line diagram (*Figure 1* gives an example). Graphical diagrams take more effort to maintain but are more quickly understood by non-architects and clearly express the cases where goals have multiple parents. Finally, goals models in the style of problem frames can be created to express domain details more richly.

Goals are connected to the domain concepts they either control or observe (use as inputs). Decomposing a higher-level goal typically relies on domain properties (central heating in our example). Goal decompositions often follow a pattern (called a frame by Jackson) such as the control pattern in our example.

Goals models can start out quite informal and be tightened up over time. This property makes them useful at stages of the project when there are many unknowns. Goals models can also help in the partitioning of a large task across multiple teams.

2.2.2 Component and connector model

The component and connector model expresses the runtime components, connectors, and ports in the system. For the most part, our use of these models is conventional so the description of it here will be brief and focus on a few points of difference.

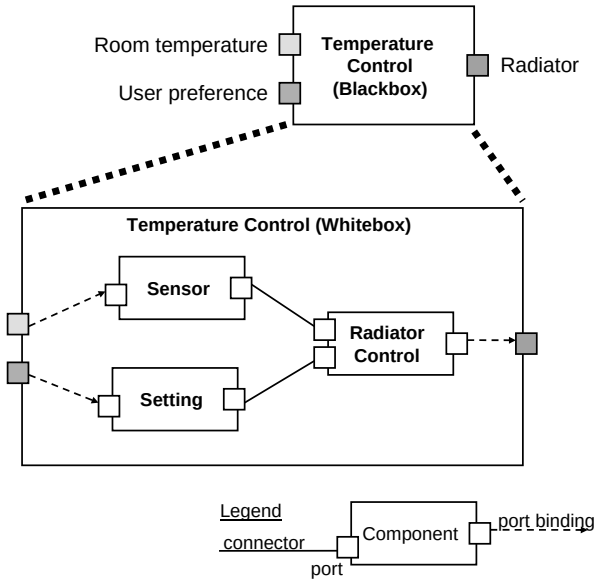

Figure 2: Example component and connector model

For many systems it is sufficient to create just two levels of refinement, which we call the blackbox and whitebox (example in Figure 2). The blackbox component and connector model depicts the system to be built as a single component and also contains external systems that it interacts with (omitted in the example). In the whitebox component and connector model the components on the inside of the system to be built are shown along with bindings to the blackbox ports. Limiting modeling to two levels of refinement provides clarity when it works, but occasionally the architect is forced into more than two levels and this simple nomenclature can work against clarity.

As a weak surrogate for richer descriptions for ports and connectors, we sometimes use a simple naming convention. The port is prefixed with either provided or required to imply suppliers or consumers and given a name corresponding to the types that flow across it.

2.2.3 Information model

The information model expresses the terms in the domain and relationships between the types. It is not a stored data model but instead a conceptual model. Our models are often related through refinement, though the refinement is rarely formally expressed because of the effort required. For example, there is usually an information model that documents the types and relationships from the goals model, another for the blackbox component and connector model, and another for the whitebox component and connector model. In detailed modeling, each port can have its own information model describing the relationship between types from the domain and datatypes passed along the connector.

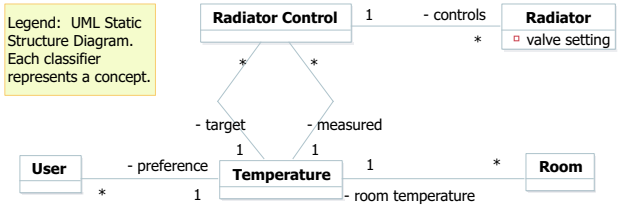

Figure 3: Example information model

Used diligently, the information model ensures consistent usage of vocabulary and reduces the chance that subject matter experts, architects, and developers will have different interpretations of domain terminology. Invariants can be used to express the relationships between domain terms, e.g., relating a persons age with her birth date.

Information models can be represented as textual tables in a word processor or graphically using UML static structure diagrams (example in Figure 2.2.3). In every case it includes a definition of the concepts used. In our example, temperature would be defined as an absolute measurement in degrees Fahrenheit (rather than qualitative measures like hot). In that sense the information model fulfills the role of a glossary.

2.2.4 Behavior model (use case model, scenario or RAD)

The behavior model expresses the behavior of the system. Often this is the most difficult part of modeling architecture and so we use a variety of techniques that vary in their expressiveness and difficulty.

Scenarios are an ordered sequence of actions performed on the system by actors. A scenario describes one possible use path through the system, not all possible paths. They are easy to create, effective at engaging subject matter experts, and refutable. However, it is also impossible to describe all possible system behaviors with scenarios and time-consuming to keep them updated as the architecture evolves.

The UML use case model is a graphical map of use cases that provides an at-a-glance overview of who uses a system and what they can do.

In order to model all possible system behavior, we use Role Activity Diagrams (RADs). A RAD is a graphical representation of use cases that expresses both who participates as well as the permissible ordering. Parallel activities can be depicted because RADs are based on Petri nets. Simple RADs are easy to create and understand, but this can fall away quickly with slightly more complex RADs.

2.3 Detail knob

The benefits of architecture models must be weighed against the costs, especially the time it takes to develop them. For each of the elements listed above, we have a conceptual detail knob that we can twist to build simple or complex versions of the models. For each project, and even for different times on the same project, we set the detail knob to balance the benefits with the time investment in the architectural models.

For goals models, it is the least effort to create textual versions and to focus on the highest level goals. More detail can be added by using the problem frames style of goals models and by adding more sub-goals.

For component and connector models, starting with a textual list of components, connectors, and ports is the least effort. Switching to a graphical representation of these components and ports takes more effort but provides models that are easier to visualize. Detailed port and connector descriptions provide more value and can be analyzed with respect to various quality attributes and protocol conformance.

For information models, a simple textual dictionary of domain types provides substantial value. The addition of invariants to encode relationships and presentation as a graphical UML static structure diagram both help precision but take more time.

For behavior models, a list of supported use cases provides an overview of system functions. When presented as a graphical use case diagram it is easier to visualize. Scenarios do not require much up-front effort but keeping more than just a few updated takes time. RADs take the most time but provide details on sequencing of behavior not found in the other models.

Choosing the setting for the detail knob is an important part of deciding on the process for using the architecture technique. This report does not prescribe process details but it is easy to imagine, for example, that in a spiral process the architect would set the detail knob low on the first pass and higher on subsequent passes.

3. SOFTWARE PROJECTS

All of these projects took place at a large financial firm. Many large financial firms, including this one, have emerged from repeated mergings of smaller firms, each with its own set of information systems, yielding a great variety of systems within the firm. Reference data is often fragmented across these multiple systems, making conceptually simple tasks rather difficult.

The firm is beginning to use software architecture modeling and these projects are among the first. Some architects are full time employees while others are contractors but all participate as peers on the project teams. Most software projects within the company, including these projects, are developed by a team comprised of players from different departments.

Precise modeling and software architecture were identified by senior management as tools that could help improve software quality and project efficiency. Adherence to the old process did not require the use of any particular software engineering techniques but did require the use of specific document templates that effectively imposed a waterfall style process. Since there were no pre-existing uniform techniques in place, nor any design metrics, it was not possible to take measurements to show improvement.

The following sections describe four projects where the architecture modeling technique was applied and at least one of the authors was the lead architect. The first three projects deal with non-proprietary technology and we have some freedom to discuss their domain details but for the last project, labeled just Project D, we can describe only its use of the architecture technique.

3.1 Identity and Entitlement Management: Documentation and coordination

This project dealt with identity and entitlement management. In small companies, keeping track of employees and what resources they have access to is straightforward. In large companies where employee records might be stored in multiple repositories and the number of systems they might have access to numbers in the thousands, the job of tracking entitlements becomes a significant challenge. An entitlement is an ability to do something to a resource, for example, the ability to login to a server or the ability to execute a transfer of up to $10,000 between accounts.

This project arched across three constituent projects: Entitlement review, provisioning/de-provisioning, and authentication/authorization. The latter two can be purchased from vendors while at the time it was not possible to purchase an acceptable entitlement review application. All worker entitlements are supposed to be reviewed, so those entitlements must be collected and presented for review. Some of those systems can be provisioned and de-provisioned through a central software application and some entitlements can be checked at runtime via the authentication/authorization application. In a smaller company it might be possible to connect every system with entitlements to the provisioning/de-provisioning system but in this large company there were too many legacy systems for that option to be practical.

The technology goals of using software architecture models was to coordinate the three constituent development projects, identifying in advance possible points of concern, enabling planning, and ensuring well-informed product purchases. The communication goal was to aggregate the three designs and communicate to senior management how they collectively would solve identified business problems.

Procedurally, work on this project started by mining the design documentation from the three constituent projects. In two of the three projects, this documentation had been built by vendors whose products were final candidates for purchase. Consequently, the design documents contained a variety of models ranging from detailed designs to interface definitions to architectural models. In the end, a stack of component and connector models had been created with the most abstract model showing the identity and entitlement management component and its connections to external systems; its refinement showed the components for the three constituent projects, their connections between each other, and the bindings to the higher-level ports. The tidy refinement of the final models was not mirrored in the creation of the models the creation of the models involved repeated back-and-forth between discovery

of details about the lower-level projects and the revision of models to express them.

A goals model had been created for the parent of this project, so the goals model for identity and entitlement management was built to demonstrate satisfaction of the higher level goals. Similarly, goals for the three constituent projects were built to demonstrate satisfaction of the identity and entitlement management goals.

Subject matter experts were presented with a rough draft of a scenario and participated in its cleanup. The primary driver for the behavior of the system was a single large (30-step) scenario describing the full lifecycle of a worker as it relates to the use of this system. This scenario was built at the blackbox level for the project and was later extended at the whitebox level to express sequencing of behavior between the three constituent projects. Occasionally other scenarios were sketched but were not maintained over time or included in the documentation.

In summary, this project used the simple style of goals models with refinements up to its parent project and down to the constituent projects. Component and connector models for both the blackbox and whitebox were created. A detailed information model was created but only at the level of the goals model (it was not refined to add new concepts that appeared in the whitebox). RADs were initially created to express the system behavior but over time only the single end-to-end scenario was kept updated.

3.2 Entitlement Review: Brownfield Design

The entitlement review project is a constituent project underneath the identity and entitlement management project. The system has been evolving for a few years and collects entitlement data daily from many systems in the company. Reviewers can browse data for the workers they are responsible for and can conduct periodic official reviews to attest that the workers have no more entitlements than necessary. The system was in use by just one division of the company and we designed extensions to support is use by the whole company.

The purpose of creating architecture models for this project was to express the requirements, communicate these requirements to the development team, and to design a solution that was compatible with the peer identity and entitlement management systems.

Management decided to use the implementation team for the existing product to build the next version. Since the architects and the implementation team were in different divisions of the company, the priorities of the two were not aligned initially. The architecture modeling for this project started out poorly because the development team had no design models, would not share implementation artifacts like the database schema or codebase, and was too busy working on other projects to meet with architects to document the existing system. We were able to build models from the details we did know but our confidence in them was low because we had no experts to validate them. The working relationship improved over time but there was not time to make improvements to the models of the existing system, which impaired the way that the architecture techniques could help the project.

An additional hindrance was the need for the implementation team to receive documents in a particular, non-architectural format. Consequently, we produced architecture models and shoehorned them into the document template. As such, most design discussions did not make reference to the architecture models until late in the engagement. The implementation team has warmed up to the models, however, and has agreed to make them the central mechanism for discussing the design in the next set of enhancements scheduled to follow the current set.

The project used the simple goals model; a detailed information model that expressed many domain terms, synonyms, and some invariants; a minimal behavior model because of limited information; and an acceptable blackbox component and connector model but a known insufficient whitebox model.

3.3 Provisioning/De-provisioning: Product selection

The provisioning/de-provisioning project is a constituent project in identity and entitlement management. The purpose of the system is to provide a central place to administer workers entitlements. Administrators can create or remove entitlements using a single user interface and, through connections to managed systems, the actual entitlements are changed on the affected systems.

The goal of creating architecture models for this system was to ensure that the product selected would match the needs of the business, to define a common model of entitlements to be shared by all programs, and to produce a whitebox component and connector model of the system to enable the creation of workflow scripts.

A significant challenge on this project was the collection of information to create architecture models. The team within our company that was evaluating the vendor product was not co-located with the architects and was under tight deadlines to demonstrate feasibility, leaving little time to discuss what they had learned. The vendor lacked the kinds of documents that would help the architects build an architecture model. A significant obstacle, initially not detected, was that the vendor and our company used the same terminology with different definitions. A detailed information model of the vendor product enabled us to identify and overcome this obstacle. Eventually a purchase decision was made without having complete confidence in the compatibility of this product with the overall identity and entitlement management project.

This project used the simple goals model like other identity and entitlement management projects. The blackbox model was detailed and had ports appropriate to support what was known of the vendor product. The whitebox model was known to be deficient since the vendor had no documentation and a limited amount of time was allocated to discover the architecture.

3.4 Project D: Greenfield Design

Project D is concerned with the architecture of a system that is to be developed over several years. The project is aimed at bringing clarity into this longterm effort early on. The system is best described as a greenfield development effort to provide functionality that no existing system in the company covers.

Even though the necessity for the system had been recognized, the requirements for the system were only understood in the broadest terms. Project D developed the business and software architecture for the system based on input from subject matter experts, primarily through a precise goals model. This precise goals model in turn required a comprehensive domain model. Finally, blackbox and whitebox architectures of the system were derived from the goals and domain models.

The goals and domain models were the core deliverables of project D. They required substantial effort to produce and the subject matter experts rated them as the greatest valueadd of the project. The difficulties in developing goals and domain models for the system arose mostly from the nature of the system as a visionary system that even subject matter experts had only vague and conflicting ideas about. While domains in the other projects were well understood and the subject matter experts were able to focus on articulating the system functions, in Project D the domain was novel and forced us to create a domain model for the system as part of the project.

We elicited the goals model from the subject matter experts through example scenarios of what should be possible to do with the system. Using these scenarios, the architect created a draft of a goals model that that was then refined with the subject matter experts. This process proved to be surprisingly efficient in discovering goals and the domain of the system.

4. OBSERVATIONS

In reflecting on these four projects we have noticed similarities that are described in this section as themes. In each project we were able to use the models as a central discussion point between the subject matter experts and the technologists. We found that our same set of techniques, if allowed to vary in the level of detail, could be used on projects with quite different character. Our use of information models, even at the most abstract levels of architecture, was important in expressing the understanding of the domain. Unfortunately, software architecture modeling is not a silver bullet but architects should expect that learning the techniques will make them more effective. Finally, we still find it challenging to decide when to stop modeling and move on to other development activities.

4.1 Bridge from business to technology

Across the four projects a strong theme was the use of models to bridge the gap between the business and technology domains. The best example from the identity and entitlement management program was the use of architecture models to communicate the design of the system to management and other interested teams during a meeting. The presentation included the goals model, information model, component and connector models, and an excerpt of the end-to-end scenario. It was effective enough that the audience could immediately ask relevant detailed questions about areas that concerned them.

Software architecture decisions are often of such high level that it is impossible to strictly categorize them as either business or technology decisions. For example, in an early stage of entitlement review we addressed a problem regarding data quality. The concern was that the existing team receiving data feeds could not resolve problems with the increased number of data feeds, as resolving each problem required contacting the feed provider and negotiating a resolution. We considered two solutions. The first was to keep the data collection centralized but to delegate responsibility for data quality issues to the feed provider. The second was to partly decentralize the data collection, thus limiting the number of feeds and the number of groups that the central team would have to coordinate with. Both alternatives address the same goal but are remarkable in that one is a technology solution and the other is a business responsibility solution. This demonstrates how architecture often sits on the boundary between business decisions and technology decisions and how the goals modeling can uncover such options.

A challenge in the provisioning/de-provisioning project was to find a representation for entitlements that worked across the vendor products. We were able to use architecture models, including scenarios, to communicate with management how significant this problem was and that it was not just a data translation issue. Based on the shared understanding facilitated by the architecture model, management allocated resources to solve the problem.

The use of architecture models to bridge gaps between business and technology was most apparent in project D. In this project there were no implemented solutions in existence and no technologist could start writing code until the problem was described and understood. Many iterations were required between subject matter experts and architects before both were satisfied with the solution. The groups communicated their ideas and expressed their concerns through the architecture models. In addition to being the primary vehicle for conveying design proposals, the precise architecture models exposed fuzzy terms and fuzzy thinking. The hierarchical nature of the models aided this iterative process. In particular, goals are decomposed into sub-goals and types are decomposed into subtypes, allowing the group to quickly zoom in from high-level overviews to the relevant detailed models.

4.2 Collection of techniques plus detail knob

The architecture modeling technique provided a backbone of four elements to express the core functions of the system but allowed us flexibility in choosing the level of detail. In each project we set the detail knob differently to respond to the needs of the project since, for example, it is not a good investment of time to model implementation details when you are planning on purchasing a vendor product. Our finding was that despite the differences between the projects, the core set of techniques was largely sufficient to express our intent. The level of detail used on each project is summarized in Table 1.

All of the projects except for project D used the simple style of goals modeling that lacked the problem frames style of integrating domain details. While project D was unprecedented and its goals still quite unclear, the other projects could rely on a general shared understanding of the domain as a substitute for detailed goal modeling.

When concerns arose we were able to turn the detail knob up on that particular area. For example, when it became apparent that the existing entitlement review application and the provisioning/de-provisioning application might have incompatible views of entitlements, it was possible to write more detailed scenarios and build more detailed information models.

The entitlement review project required the creation of additional models beyond the core set. A spreadsheet was built to encode the application user roles and the set of entitlements each had. Another spreadsheet was built that tracked the referential integrity of two source data feeds over time as it was cleaned up and became more complete.

4.3 Model of domain concepts

Our observation that a model of domain concepts is useful at the architecture modeling level is not novel but neither is it universally recognized. Subject matter experts may be in a hurry to describe a systems functions and technologists may be in a hurry to describe how those functions will be implemented, but we have found it essential to build an information model that underpins both and ensures that concepts and relationships are well understood.

In the entitlement review project, subject matter experts from many domains contributed to the project requirements. We discovered that their terms might overlap but they did not always agree on relationships or definitions. On this project synonyms were common so we used convenience attributes and invariants to encode them.

A central challenge on the provisioning/de-provisioning project was the structure of entitlements. Each system to be provisioned had its own model of entitlements and we needed to produce a model that covered them all and was able to encode role-based access control. After creating an information model that we hoped would be sufficient, we discovered that often resources are provisioned indirectly. For example, the provisioning system might actually create new entries in an LDAP server in order to entitle access to another system. The precise encoding of our understanding as an information model enabled us to express our understanding and to

	Models			
Project (character)	Goals Model	Component and Connector Model	Information Model	Behavior Model
Identity and Entitlement Management (documentation)	Simple	Detailed blackbox, detailed whitebox	Sufficient for simple goals model	Detailed single end-to-end scenario
Entitlement Management (brownfield)	Simple	Acceptable blackbox, insufficient whitebox	Detailed	Minimal
Provisioning / Deprovisioning (product selection)	Simple	Detailed blackbox, minimal whitebox	Sufficient for simple goals model	Minimal
Project D (greenfield)	Detailed	Acceptable blackbox, idealized whitebox	Detailed	Simple scenario

Table 1: Level of detail by projects

detect when our design was incompatible with new requirements.

Unlike the other three projects, project D entered into a domain still being explored by the subject matter experts (and the irony of their title was not lost on the group). Rather than using the information model to simply detect overlapping terminology or ensure knowledge transfer between subject matter experts and architects, project D used the information model as a key working model by the subject matter experts themselves to encode alternative possibilities and grow their understanding of this new domain. The precision of the model enabled them to detect inconsistencies with their proposals and to communicate them to the architects and other subject matter experts.

4.4 Architecture techniques amplify skill

Some tasks have the property that, after a person has been trained to do them, the work of one person cannot be differentiated from another. For example, after teaching Ann and Bob to fill out timecards, we dont expect that one will do it better than the other.

Software architecture modeling can amplify the skills of an architect but cannot guarantee success [5]. While we should expect that after training they will have greater insight and have access to more precise techniques, we should not be surprised when Ann can use the techniques effectively and Bob struggles.

All models are abstractions of more complicated systems. In the hands of an expert, models can be used to comprehend properties of systems that would otherwise be too complex. Models can fail to be useful either if they focus on unhelpful properties or if they cannot be analyzed by the architect. Two of our projects were examples of these failings.

On the provisioning/de-provisioning project, over the course of many meetings our architects were unable to validate that our overall model of entitlements was compatible with the vendors model despite having access to published documentation, internal documentation, and even the vendors implementation team. The vendors internal, implementation-level model made extensive use of meta-modeling ideas like key-value pairs but did not express what instances would be present at runtime. The vendor had produced an accurate model that abstracted away details necessary to answer our question. It is our belief that had the vendors team been knowledgeable regarding architecture modeling that their models would be more helpful for our task and less like literal drawings of the data structures.

On the entitlement review project, one of the goals was to educate an apprentice architect. After training he was able to produce syntactically correct architecture models, the same as the more experienced architects. Due to his lack of experience, however, his models at the time tended to be straightforward expression of what he had learned from subject matter experts and he was not yet able to use the formalization to detect inconsistencies or expose gaps in the design. As a consequence, the architecture models did not help him to improve the quality of the system design.

Not all subject matter experts in project D embraced the architecture models. Those that rejected the technique outright continued to create documentation with imprecisions and contradictions that would have been avoided with the goals and information modeling techniques. Those that embraced the models avoided these problems and were able to help the architects remove problems in the design. This experience suggests that the models were a catalyst in designing the system.

4.5 When to stop adding detail

With both an ability to turn up the detail knob on every model type and an ability to borrow more modeling techniques from the source techniques, it was often tempting to continue adding detail to our models. In formal or informal reviews, architects often asked each other why they chose the level of detail they did, or even expressed the opinion that more detail should have been added in particular places. Adding more detail must always be traded off with an additional time investment to add that detail. While we cannot offer a universal rule, we can describe some cases and our decision process. In general, we traded off model creation effort with the models ability to answer questions.

The choice of where to stop the model was easiest in the identity and entitlement management project because each of its three constituent projects had an architecture model itself. This was not license to add all possible detail because in the whitebox component and connector view showing the three constituent projects there were approximately sixty ports, either for communication to the peer systems or to external systems. While we had the modeling capability to document the datatypes and operations for each of those ports, we declined to add this detail and forego the ability to detect problems at that level.

The provisioning/de-provisioning project was targeted at product purchase from the start. The important questions to be answered by the model included whether or not the vendor product could support our model of entitlements and if it could connect to the other projects. We did end up creating a whitebox component and connector model of the vendor product because it supported the creation of workflows and the workflow authors would need to know that level of architecture.

The choice of where to stop was most challenging on the brownfield entitlement review project. The enhancements for the next version ranged from changes visible at a highest level of modeling to small changes to input datatypes. One particular incident stood out: The developers informed us that one of our requirements would entail changes to 110 stored procedures. While this work was inevitable for this release, we wanted to be sure to avoid similar problems for subsequent releases and the architects thought

they knew how to prevent future problems. Two problems emerged: First, the developers would resent the architects encroaching on their detailed design, and second, we did not want to invest the effort to make all of our models sufficiently detailed to encode all such details. One camp of architects subscribed to the crisp boundary theory where a line was drawn and the architecture stopped there. The crisp boundary architects would not tell the developers how to avoid implementation problems but instead write quality attribute requirements, in this case, that future changes of a certain nature must be able to be made within a certain time. The other camp of architects advocated a design wedge theory where more architecture details were modeled at the top levels but tapered off as the model approached the implementation components. In the end we produced a crisp boundary architecture model and had an informal chat with the developers about implementation options.

Due to the novel nature of project D, the key question was feasibility rather than balancing various quality attributes. Because of this restriction, architecture modeling progressed until it became apparent that any given component could be constructed or perhaps already existed. Each component had a corresponding high level goal motivating it and in most cases its subcomponents were not specified to allow for latitude in implementation. It was possible to stop modeling at the highest level of components because the key question of feasibility could be answered at that level.

5. LIMITATIONS

Ideally, this report would be produced by an independent party that was neither invested in the application of the architecture technique nor responsible for its development, as were the authors of this report. To the best of our abilities we have tried to keep our observations objective.

Furthermore, an ideal report would compare some quality attributes of project delivery with and without the software architecture modeling. Since such metrics were not available in this organization before we started we were not able to make a meaningful comparison.

While this report focuses on the first year of usage of the architecture technique, we do not yet know how it will fare during its longer-term application across the whole firm. Moreover, we do not present evidence of reproducibility at other firms with different existing customs. However, our observations can serve as an early sanity check of our approach and guide future development and research.

Finally, the architects participating in these projects already have a track record of successful project delivery and are, in general, highly knowledgeable regarding software engineering theory and practice. Since not all architects will have a similar background, we have little evidence that the architecture technique could be learned and effectively applied by other architects. On the other hand, we did provide some indications that inexperienced architects benefit from using the technique.

It is possible that many readers of this report, as experts in software architecture, will be disappointed at the apparent gap between what has been shown in research contexts and what we have presented here from an industrial context. For example, we suggest the creation of UML models instead of using a special-purpose architecture description language and we have never been able to prove that our systems have any strong property, such as absence of deadlocks. In the experience of the authors, however, the techniques that we have employed are a significant improvement compared to the norm in industry projects. The concepts of components, connectors, ports, goals, and refinements are rarely represented with any precision in the all-too-common PowerPoint architectures. From this perspective, the use of UML with its mature tool support is a pragmatic choice and the lack of formalism in other areas is justified because their benefits might not outweigh the time investment required.

6. CONCLUSIONS

We have presented an architecture modeling technique that we believe has pulled the best ideas from various research publications. We summarized anecdotal evidence from its usage in four projects of different nature across five themes of observations. These observations address practical problems of software architecture that we believe have to be faced by many software architects in practice. These projects were performed at a large financial company that bears similarity to many other information technology departments that the authors have experienced. Thus we believe our observations can guide the application of software architecture by practitioners in similar situations.

Our observations are based on anecdotal evidence and suggest future research questions. For example, we make the observation that levels of detail of different models are possibly tied to the nature of the project. Quantitative data from a variety of projects could expose relationships between these variables.

Another unresolved issue regards how architecture modeling is used within a company and by whom. It seems inappropriate to expect every person in the company to learn the technology. If only some learn it, how should they interact with those that do not? We have tentatively identified three levels of knowledge regarding the models: In Level 1 someone can read a model produced by another. In Level 2 someone can create syntactically correct models. In Level 3 someone can use the models to discover flaws in proposed designs, to identify areas of the domain that have not yet been modeled, and to evaluate various quality attributes. Future reports or research could address which people should get which level of training.

We have not addressed how to decide the balance between investing time in software architecture versus proceeding with what has already been modeled. This is an important technology transition question as time and money are always scarce commodities on commercial projects. In many ways the research community has taken the high road and investigated highly formal models that require considerable time investment with commensurate value in special domains, e.g., high reliability systems. Many industry managers responsible for non-exotic projects would ask What benefit can I get if I invest 1, 2, or 3 weeks in building an architecture model?

The response of the project teams to the architecture models has been generally positive. There are some team members who are reluctant to learn another technique. Others question the time investment compared to just starting coding. Still others support the techniques in principle but when deadlines get tight they revert to their old ways. But overall the people who have participated have been happy with the results. They tell us that the models encode a clear description of the problems and solutions, sometimes indicating that the clarity we can demonstrate in the problem definition is the greatest value.

7. ACKNOWLEDGEMENTS

We thank Jonathan Aldrich, Bradley Schmerl, David Garlan, Bill Scherlis, the SSSG, and the anonymous reviewers for helpful feedback on this material. The first two authors wish to acknowledge support through the US Army Research Office (ARO) under grant number DAAD19-01-1-0485, the National Science Foundation un-

der grant CCR-0113810, and a research contract from Lockheed Martin ATL. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ARO, the U.S. government, Lockheed Martin, or any other entity.

8. REFERENCES

[1] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.

[2] John Cheesman and John Daniels. *UML Components: A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2000.

[3] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, Reading, Massachusetts, 2002.

[4] Desmond F. D'Souza and Alan Cameron Wills. *Objects,Components and Frameworks With UML: The Catalysis Approach*. Addison-Wesley, 1998.

[5] George Fairbanks. Why can't they create architecture models like "developer x"?: an experience report. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 548–552, Washington, DC, USA, 2003. IEEE Computer Society.

[6] Michael Jackson. *Problem Frames: Analyzing and Structuring Software Development Problems*. Addison-Wesley, 2000.

[7] Rick Kazman, Mark Klein, and Paul Clements. Atam: Method for architecture evaluation, 2000.

[8] Jeff Kramer and Jeff Magee. Engineering distributed software: a structural discipline. In *ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 283–285, New York, NY, USA, 2005. ACM Press.

[9] Emmanuel Letier and Axel van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. *SIGSOFT Softw. Eng. Notes*, 29(6):53–62, 2004.

[10] Martin Ould. *Business Processes - Modeling and Analysis for Re-engineering and Improvement*. John Wiley and Sons, Chichester, 1995.

[11] Samuel Redwine, Jr. and William Riddle. Software technology maturation. pages 189–200, August 1985. This paper was cited as the best/most influential paper by the program committee for ICSE 18 in 1996.

[12] Roshanak Roshandel, Bradley Schmerl, Nenad Medvidovic, David Garlan, and Dehua Zhang. Using multiple views to model and analyze software architecture: An experience report. Technical Report USC-CSE Technical Report USC-CSE-2003-508, USC, 1003.

[13] Jim Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[14] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., 1996.

[15] Ian Sommerville. *Software Engineering*. Addison-Wesley, 7th edition, 2004.