Making Metaphors that Matter

Michael Keeling
Vivisimo
Pittsburgh, PA, USA
Email: mkeeling@neverletdown.net

Michail Velichansky
Buzzhoney
Pittsburgh, PA, USA
Email: michailv@gmail.com

***Abstract*—The system metaphor was proposed in Extreme Programming as a lightweight alternative to more rigorous architecture practices, but many teams have trouble using metaphors effectively to improve team communication or evaluate a system's architecture. This is no surprise as few agile teams have had training in architecture practices let alone in how to create metaphors. In our experience this does not invalidate the metaphor, but rather shows that more guidance is needed in the proper use of metaphors. This paper outlines one team's positive experience using system metaphors in the development of a medium-sized, service-oriented, enterprise system. Specific guidelines for creating effective metaphors are presented along with concrete examples.**

***Keywords*-software architecture; Agile; Extreme Programming; system metaphor; architecture styles; architecture patterns**

I. Introduction

As introduced in Extreme Programming, the system metaphor is a simple story that describes a software system so everyone including customers, programmers, and managers can understand how the system works [1]. The system metaphor is one of the primary tools in an Agile team's toolbox for describing a system's architecture but it is also one of the least used, and many Agile teams have had mixed results applying the technique [2] [3] [4]. The system metaphor is meant to improve team communication and architectural understanding, but teams' metaphors often fall short of what is necessary to build software effectively.

This has led some to conclude that the system metaphor should not play an important role in Agile software development at all, but this seems unrealistic. In our experience, metaphors tend to show up whether we mean to use them or not, regardless of the development process, simply because software is abstract – there's no physical manifestation to the artifacts we build. It's no surprise, then, that many engineers resort to the metaphor in an attempt to be helpful and to find some way of controlling the fluid, complex mass of code, requirements, and rules they're trying to build [5].

The question, then, is not whether to use the metaphor, but how to make sure that the metaphors we do use help us meet our goals. We think metaphors can be a valuable asset as long as your team agrees on what constitutes a "good" metaphor as we outline here. By applying these guidelines, metaphors can improve team communication, prevent confusion, and help keep the team positive, productive, and agile.

II. Project Background and Context

Over a period of about nine months we created a set of software tools to help shoppers of a large, regional retailer plan their shopping outings using a browser-based web application. Both authors were contractors working for Buzzhoney at the time. A context diagram for this application is shown in figure 1. Effectively sharing design decisions among our team of eight engineers, designers, and managers was challenging. While each team member had at least two years of software development experience, most were new to both software architecture practices and agile software development. And the project was not trivial. The completed software would need to integrate with an existing website maintained by the retailer while satisfying requirements for maintainability, reliability, security, scalability, and performance.

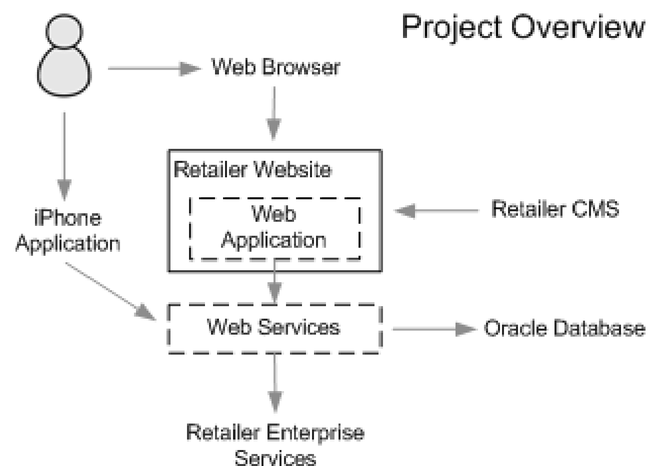

Figure 1. The web application we built lived within the context of a larger enterprise system.

Knowing that the system metaphor had a mixed track record we were initially hesitant to rely exclusively on metaphors for describing the system we were to build. In fact we had no intention of using metaphors to describe our architecture at all. It was not until we noticed that we

978-0-7695-4370-3/11 $26.00 © 2011 IEEE
DOI 10.1109/AGILE.2011.28

IEEE computer society

were using metaphors anyway in our day-to-day discussions that we truly embraced the idea of capturing architectural decisions as metaphors rather than only as sketches in a notebook or on a whiteboard, or as a document that few people would read and find useful. As there were many unknowns in the project, we had a strong desire to keep our documentation light to avoid rewriting documents every time we learned something new about the system we were building. The system metaphor seemed like the right tool for the job.

Describing our architecture using metaphors helped the team understand how the design satisfied specific quality attribute concerns and helped us avoid making inappropriate low-level design decisions that would violate our architecture, though using metaphors effectively did not come naturally. Initially the metaphors we used to describe the system only confused the team, so much so that we agreed to stop using metaphors altogether for a while.

In spite of this moratorium on metaphors, they still cropped up in code and in conversation. Rather than fighting what seemed to be a natural tendency for communication we chose to embrace metaphors and figure out how we could consistently create useful ones. Examining the metaphors that seemed to work for the team as well as those that did not, we identified a set of simple guidelines for evaluating metaphors. By deliberately applying these guidelines, we found that the metaphors we created were consistently useful. They provided excellent architectural guidance without lengthy documentation which was particularly helpful given that stakeholder needs changed throughout the project.

The following section describes our six basic guidelines for evaluating metaphors in detail and provides some example metaphors taken from our project.

A good metaphor:

1) **Represents a single view.**
   Don't try to describe your entire system with one metaphor.
2) **Deals with only one type of structure.**
   Use metaphors of things for static structures, metaphors of actions for dynamic structures.
3) **Gives clear guidance concerning design decisions.**
   It should act as a guide rail for implementation by communicating design, discussion, and rationale.
4) **Sheds light on system properties.**
   It should assist you in telling meaningful stories about how specific properties are promoted or inhibited
5) **Draws on a shared experience.**
   It should be relatable and bring to mind the discussions had when designing or describing a system's architecture.
6) **Is meaningless without explanation.**
   A metaphor can't stand on its own. You have to create a shared experience.

III. Guidelines for Evaluating Metaphors

What makes a metaphor useful? What makes it not? Our primary criteria for judging a metaphor was how well it assisted team communication. Did the metaphor cause confusion or did it clarify thinking? Did the metaphor provide specific guidance or did it merely resemble our design? When a teammate mentioned a metaphor, did it immediately bring to mind a set of design decisions and quality attribute considerations?

Within a few weeks we had identified several system metaphors and we began to notice that the most effective metaphors shared certain traits. These traits became the set of guidelines we used for evaluating and identifying metaphors for the remainder of the project.

A. A good metaphor represents a single view.

Too often, metaphors attempt to describe a system as a whole. In our experience, all but the most trivial systems become too complicated for this kind of approach. Common "whole system" metaphors are things like "a city" or "a subway system," which do not provide much useful information. Essentially, they only say "this complex system is like a complex system," which is not that useful to anyone trying to implement the design.

Rather than using a single metaphor to describe the entire system, we found that communication was better served by having many metaphors, each dedicated to a different view of the system and sometimes even different stakeholders. This is a radical departure from the traditional system metaphor approach in which there exists a single metaphor used by all stakeholders. The end result, however is the same – a vocabulary for discussing the design of the system being built. The notion of using multiple views to describe a system is now a common software architecture design practice [6] - [10] and we felt that it was only natural to extend the practice to the system metaphor.

One of the best examples of a metaphor that represents a single view is our "Bento Box" metaphor. The "Bento Box" demonstrated how the client-side code (in this case, JavaScript) was organized. To someone familiar with architectural styles, the "Bento Box" looks like a view from the Module Viewtype that applies the layered style [6]. Focusing only on how the code was organized in one part of the system not only made it easier for us to devise a useful metaphor but also allowed us to create a shorthand for discussing the concerns for this part of the system, in this case what functions and objects another object was allowed to access directly and how that influenced modifiability and maintainability.

How did we arrive at a Bento Box? Looking at a diagram on a whiteboard (see figures 2 and 3) showing our design, the team was concerned about the "spaghettification" of the code. The decision was made to allow it, as long as the spaghetti-code was contained within a particular layer. At

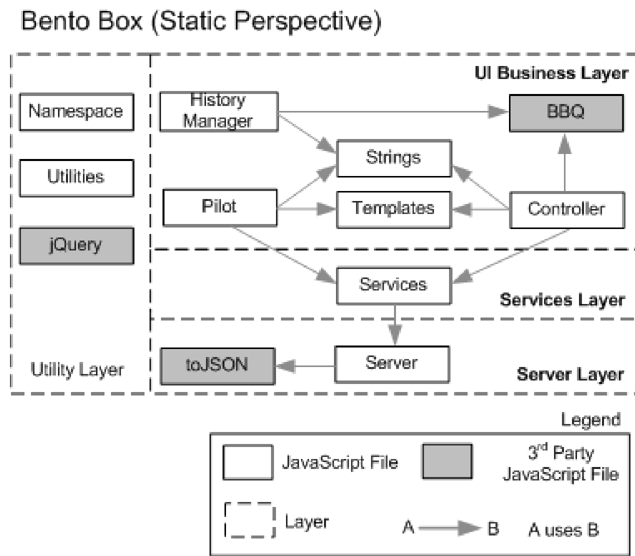

Figure 2. The UI Business Layer drew concerns of “spaghettification” which ultimately led to the “Bento Box” metaphor. Thinking of the architecture as a bento box helps describe the separation of layers and remember the responsibilities for each layer.

Figure 3. The “Bento Box” metaphor was both useful and delicious.

that point, one teammate explained to another jokingly, “So it’s kind of like a Bento Box. You have your noodles in this compartment, your vegetables here, some chopsticks that can touch everything, all in their own area. As long as the noodles stay in their compartment, then the architecture is okay!”

Describing the architecture using multiple metaphors, each a different view of the system, does not imply that you can not or should not have a metaphor which depicts an overview or the overall context of the system. Such a view of a system is certainly helpful. Simply put, such an overview of the system should not be the *only* metaphor you create. In our experience, the more unique the metaphor is to the system, the better you will be able to exploit that metaphor for communication, implementation, guidance, and understanding system properties.

B. A good metaphor deals with only one type of structure.

One of the reasons using a single metaphor to describe the entire system was confusing to us was that it was often unclear what kinds of structures we were actually talking about, be it static code, executing processes, hardware, or even data. A common practice in software architecture design is to clearly discern between static, dynamic, and physical structures [10] (other similar concepts are Viewtypes from [6] and views from [9]). We found that this worked extremely well with metaphors too.

Generally, the best success we had was by using *metaphors of things* to represent static structures and *metaphors of actions* to represent dynamic structures. The “Bento Box” is a stacked box used to organize food (it keeps the noodles contained in their own compartment) so it was a natural fit for describing how some of our code was organized.

Our “Pilot-Navigator” metaphor (shown in figure 4) is a single view dealing with only dynamic structures and describes how the web application moved from page to page. Just like a pilot flying a plane actually arrives at some destination, so too did the flow of control through our objects arrive at a specific view requested by a user. The act of flying from place to place helped us think about the dynamic interactions in this part of the system and how our system achieved desired modifiability properties.

C. A good metaphor gives clear guidance concerning design decisions.

Our most effective metaphors ultimately helped us make higher-level, architecturally important decisions relevant and accessible while the team was dealing with lower-level, implementation details. Maintaining architectural integrity is, after all, a team responsibility and qualities cannot be assured if design rules are hidden or hard to understand. Poor decisions made during implementation can undermine even the best planned architecture. In this way, a good metaphor acts as a guide rail for implementation.

The “Bento Box” was very clear about the responsibility of each layer and how those layers interacted. Formally, the “Bento Box” applies the *allowed-to-use* relation in which any layer is only allowed to use another layer directly below it, but this was never at the forefront of team members’ thoughts. Though never formally defined, these rules were essential to our effectiveness when refactoring code and building new features.

When adding a new object, for example, teammates discussed design alternatives using the metaphor: “You can’t

make that function call, we don't want to mix the server layer of the Bento with the noodles." Simply mentioning the "Bento Box" during a discussion instantly recalled all the rationale for the structures, and allowed the team to quickly communicate concerns and avoid architecture violations.

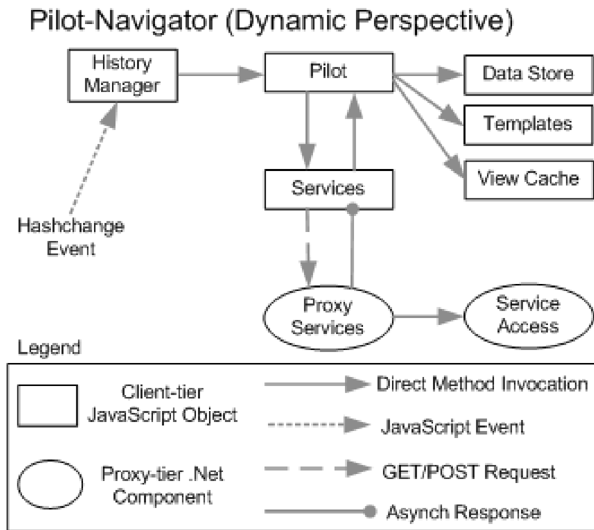

Figure 4. The history manager acts as a navigator, telling the pilot where to go to get data from the services and how to apply that data to templates.

D. A good metaphor sheds light on system properties.

While structures and guidelines are an essential piece of a software system's architecture, understanding why those structures and guidelines exist in the first place is just as important. Since the metaphor is a lightweight tool placed directly in the hands of developers, clearly understanding how the ideas captured in the metaphor promote or inhibit specific properties is extremely important. Most often these ideas were shared through storytelling with the metaphor playing a central role in the story.

Our stories, while informal and generally imprecise, were similar in spirit to quality attribute scenarios [11]. The stories we told almost always dealt with a single quality attribute at a time and strove to capture how the system was supposed to behave and how our design decisions supported that behavior.

For example, maintainability and modifiability were two of the most important qualities which drove our decision making for the structures behind the "Bento Box" metaphor. To help a teammate understand how our architecture decisions supported this choice, we would tell a simple story about eating lunch. How is maintainability promoted? "Part of maintainability is about clean code - code is not clean when it's all jumbled up. Without a bento box, my dessert, noodles, and sushi might get all mixed up which won't taste very good." How is modifiability promoted? "Since everything is in it's own compartment, if I want to eat dessert first I know exactly where to go in the bento."

Our "Werewolf" metaphor showed how static structures mapped to dynamic structures at runtime, and how those dynamic structures were expected to behave. Reliability was an important driver in choosing this design. Specifically, when a shopper attempted to use the site with JavaScript disabled only a minimal subset of functionality would be available. For example, neither autocomplete nor drag-and-drop were expected to function without JavaScript. The imagery of the "Werewolf" was perfect in this case. "When JavaScript is turned on, the system is a human - predictable, reliable, civil, usable. But on the rare occasion when there is a full moon, that is when someone disabled JavaScript, the man turns into an unpredictable, vicious wolf with no regard for reliable behavior."

Over time storytelling became an easy test for determining the fitness of a metaphor. If it was difficult or awkward telling a story or if the story didn't make sense, then the metaphor likely was not a good fit. Such was the case with our failed "Ordering Fajitas at a Restaurant" metaphor. When this metaphor was introduced, it took so long to tell a story describing the metaphor that the person who invented it got lost in the details of his own story! This is clearly not a metaphor that would be easy to remember!

E. A good metaphor draws on a shared experiences.

Common experience (both technical and non-technical) is the bedrock of a great metaphor. The best metaphors are ones that the team arrives at together. Our initial metaphors were often descriptive, encapsulating the drawings, discussions, arguments, meetings, and past work that all go into designing a system's architecture. That shared experience of creating the metaphor also became part of the data payload the metaphor carried with it every time one of the team members heard it.

As developers we often reuse designs from previous software we've built and designs we've studied. Past projects become part of the shared experience a team can draw from as well as best practices in software development. Architectural styles (pipe-and-filter, layers, client-server, and so on) are metaphors too [12] and though styles from a style catalog might have a rich and established payload of meaning and rationale behind them, it's still important to create a shared context. What an architectural style means can still vary greatly from developer to developer, and it's easy for someone to focus on the wrong part of the metaphor.

"MVC in JavaScript" was one such misstep in the team's use of metaphor. One team member used it simply to impart the idea of a framework where data was applied to templates in a standardized way. Another team member, however, expected to find an established MVC framework in use, while a third began building an object-relational model like the one he was used to seeing in other MVC frameworks. It

wasn't until we discussed the problem and designed some solutions together that we were able to reconcile the different pictures each of us imagined based on our own experiences.

Finally, while we strive for technical precision as engineers, a team should not feel restricted only to serious- or technical-sounding metaphors if that's not the team's natural tendency. Many of our most successful metaphors were born from references to food or pop culture. Metaphors like the "Bento Box" and the "Werewolf" have no inherent technical meaning, while their quirkiness helped spark discussion and buy-in from the team. In fact, using non-technical-sounding metaphors can help create a clear distinction between terminology of styles ("layers", etc.) and the metaphors used to describe those styles which may carry additional, project specific meaning.

F. A good metaphor is meaningless without explanation.

When a new teammate is brought on board he or she will lack the shared experiences that went into creating the metaphor. Bring that person up to speed with diagrams and prose! Descriptive prose, storytelling, and visuals (such as diagrams on a whiteboard or piece of paper) are essential to understanding how a metaphor describes a system. Once intent and rationale behind a design are commonly understood, you may not need to keep any documentation. The metaphor should immediately jog your memory and remind you of important decisions and rules.

An early failed metaphor was the "coat hangers" metaphor used to describe an event binding system. "It's like coat hangers," one team member described some code he had written. "You keep hanging event listeners onto this object, and that way different parts of the system can 'listen in' to each other." This explanation proved useless to another team member, who kept wondering, "How is event binding like a coat hanger? What's supposed to be the closet? Are there coats that go on the hanger?" So while this metaphor helped the code's author understand the system, it was presented without adequate explanation and no visual representation which only further confused teammates reading his code.

The lesson is that just because a metaphor is useful to one person, or one team, doesn't mean it will be useful to anyone else. We must always be ready to adjust our metaphors based on the shared experience of trying to explain the metaphor to a new team member. Just like good code, good metaphors should not be regarded as written in stone, and we should be prepared to refactor them to make way for improved metaphors that better serve the needs of the team and the project.

It is important to recognize that a metaphor is not the architecture. Every software system has an architecture whether it is chosen or not. We noticed that even though the metaphor we used to describe a set of design decisions - structures, properties, rationale, and code - changed, the design decisions themselves did not. A metaphor is a representation of the architecture and if that representation is no longer useful then a new one should be sought out.

IV. Metaphors in Code

Once we started to create useful metaphors it was not uncommon to see some of our metaphors show up in the code we wrote. Communication is not limited purely to in-person discussions, or even documentation. Most of us seek to write self-documenting code, and applying metaphors to code directly is one way we can help to clarify the code within the team. Unlike the "Bento Box," which was represented purely in diagrams and sketches, the "Pilot-Navigator" metaphor was actually represented in the code itself. JavaScript object and function names actually took on the lexicon that had been established for that system, with a "navigator" object that handled the initial event and a "pilot" object that implemented "flyTo" functions. This helped reinforce the team's understanding of how the system worked, and also made it very easy to mentally identify what part of the system you were looking at when viewing the code.

For example, when looking at the Utilities JavaScript file, it took an extra mental step to remember that the Utilities layer was part of the "Bento Box" metaphor (chopsticks might have been a better name). No such step was necessary when looking at the "Pilot" object file. This ease of identification can make a team's life a lot easier over the (long) life of a complex project.

While using metaphors to create a representation of the architecture in self-documenting code is helpful, applying metaphors to the code directly in this way does introduce the problem of changing nomenclature if the metaphor changes, as well as creating a boundary for anyone trying to read the code without knowledge of the metaphor. So while it may not be appropriate in all situations, we found that in some cases it can make reading, understanding and discussing code that much easier.

V. When to Use Metaphors

Reflecting on our experience and looking beyond this project, we recognize that system metaphors as we have defined them may not be a great fit for all projects. Thinking about our guidelines for creating great metaphors, there are many situations when metaphors may not be an appropriate choice as the only means of documenting a software system's architecture.

We cannot emphasize enough how important shared context is to creating useful metaphors. It will not always be practical to create the shared experiences and context necessary to make metaphors that matter to the team. In our case, the core engineering team was relatively small, consisting of only a handful of developers. In addition the entire team was collocated and there were not so many metaphors that

Table I
SAMPLE OF METAPHORS WE USED THROUGHOUT THE PROJECT WITH A BRIEF EXPLANATION FOR WHY EACH WORKED OR DID NOT WORK.

Metaphor	Did it work?	Why?
“MVC in JavaScript”	No	MVC meant different things to different team members. Described whole system. No discussion.
“Coat Hangers”	No	Introduced without explanation.
“Bento Box”	Yes	Shared experience, single view, expanded on the well-known “layers” architecture style.
“Pilot Navigator”	Yes	Verb (flying) used to describe an action taking place.
“Werewolf”	Yes	Dual nature of the beast fit the dual nature of the system. Funny == very memorable.
“Eating fajitas at a restaurant”	No	Too complex, guidelines too flexible.
“Ordering furniture from Ikea”	No	Was not a shared experience, too general (bookcase? chair? special tools?)
“Client-Server”	Yes	Whole team had prior experience, well documented in literature.

we could not remember all of them. Large or geographically dispersed teams working on significantly large projects may have difficulty using an informal approach such as metaphors for understanding, communicating, and documenting a software system’s architecture.

Good metaphors can be difficult to create and require time to get right. While this list of guidelines will help you evaluate a metaphor and understand when you have found good ones, sometimes it might be easier to simply record everything in a traditional-style software architecture description. Of course, this document will need to be maintained and kept up to date, but even a five to ten page architecture description which clearly defines the design and intent of the system is worth its weight in gold. If this is too hard to capture in a set of metaphors then capture it in some way that makes sense to your team.

Metaphors are not necessarily better or worse than a traditional architecture description - every project has different needs. It is our intent that the outlined guidelines allow metaphors to become a viable tool in every software developer’s toolbox. One of the most important things about learning how to use a tool is knowing when that tool is appropriate to use. Our experience shows that metaphors work extremely well and we are confident that they can work for you too, in many cases.

VI. MAKING METAPHORS WORK FOR YOU

Great system metaphors improve communication by providing teams with a vocabulary for discussing the architecture of a system and how it satisfies specific qualities. To that end, forcing a metaphor that does not fit onto a design is contrary to this goal. Our guidelines were successful for us because they made software architecture best practices approachable to a team that had little experience with such practices. Our guidelines were equally applicable to evaluating formally defined styles or patterns as well as informally defined metaphors. If your team feels comfortable with architecture styles and patterns from existing software architecture literature, use them. Otherwise, these guidelines will help you create some great metaphors for the software you build.

When thinking about architecture, communication is king! If you are joining a team with an existing metaphor, don’t accept it blindly. Ask for an explanation before you think about the metaphor. Ask to see diagrams, documents, and code. Ask where the metaphor came from. If the metaphor is simply confusing you, voice that. A discussion *must* occur in order for a metaphor to be useful to you. In the process of discussing it, either the metaphor will finally make sense, or a new metaphor will emerge that everyone on the team can benefit from.

Likewise, if you are introducing a new member to your team, don’t immediately throw metaphors at them! Give them time to look at the various views, let them ask questions, show them how the code follows the metaphor. Once you have finally created a shared experience for your metaphor, then it can be brought out – and be open to change if the new team member has different ways of describing that part of the system. Just because you use a different metaphor to describe your architecture doesn’t mean that the architecture itself has changed.

As a community of practitioners we need to take more responsibility in promoting architectural thinking when building software using agile methods. While metaphors are a powerful communication tool, the agile community as a whole cannot continue to ignore the existing wealth of information on software architecture. It is our hope that these guidelines serve as the first step to creating a more architecture-aware agile community.

REFERENCES

[1] K. Beck Extreme, *Programming Explained: Embrace Change*. Boston: Addison-Wesley, 2000

[2] M. C. Paulk, “Extreme Programming from a CMM Perspective,” *IEEE Software*, vol. 21, no. 6, pp. 19-26, Nov. 2001.

[3] R. L. Nord, J. E. Tomayko, and R. Wojcik, “Integrating Software-Architecture-Centric Methods into Extreme Programming (XP)” CMU/SEI-2004-TN-036, 2004.

[4] J. Herbsleb, D. Root, and J. Tomayko, “The eXtreme Programming (XP) Metaphor and Software Architecture,” CMU-CS-03-167, 2003.

[5] B. Foote and J. W. Yoder, “Big ball of mud,” in *Pattern Languages of Program Design 4*, N. Harrison, H. Rohnert, and B. Foote, Ed. Morgan Kaufmann, 2000.

[6] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*, 2nd Ed. Boston: Addison-Wesley, 2010.

[7] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd Ed. Addison-Wesley, 2003

[8] Software Engineering Standard Committee of the IEEE Computer Society, “IEEE Std 1471-2000: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems,” Standard 1471-2000, 2000.

[9] P. Kruchten, “The 4+1 View Model of Architecture,” *IEEE Software*, vol. 2, no. 6, pp. 42-50, Nov. 1985.

[10] A. J. Lattanze. *Architecting Software Intensive Systems: A Practitioner’s Guide*. Boca Raton: Auerbach Publications, 2009.

[11] M. R. Barbacci, R Ellison, A. J. Lattanze, J. A. Stafford, C. B. Weinstock, and W. G. Wood, “Quality Attributes Workshops (QAWs), Third Edition” CMU/SEI-20030TR-016, 2004

[12] J. McGovern, S. Ambler, M. Stevens, J. Linn, V. Sharan, E. Jo, *A Practical Guide to Enterprise Architecture*, Prentice Hall, 2003