

Reframing Software Design: Perspectives on Advancing an Elusive Discipline

Alex Baker and André van der Hoek

Department of Informatics
University of California, Irvine
Irvine, California 92697-3440, U.S.A.
+1 949 824 6326

{abaker, andre}@ics.uci.edu

ABSTRACT

Software engineering researchers and practitioners have long had an uncertain and uneasy relationship with design. It is acknowledged that software design is critical and major strides have been made in advancing the discipline, but we all are keenly aware that something “is just not quite right” and that design remains one of the least-understood aspects of software engineering. In this paper, we present our novel *Eyeglass framework* and use it to offer a series of fresh perspectives on software design, its accomplishments, and fundamental challenges ahead. The Eyeglass framework is inspired by the broader discipline of design and evaluates software design in terms of seven interrelated dimensions: ideas, representation, activities, judgment, communication, domain of use, and domain of materials. The main conclusion of our examination is that we have unnecessarily limited ourselves in our explorations of software design. While there has been some success, to further advance the discipline we must step back, reframe software design to address all seven dimensions, and engage in a deep study of these dimensions, individually and as a whole.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – computer-aided software engineering; D.2.9 [Software Engineering]: Management – life cycle; D.3.2 [Programming Languages]: Language Classifications – design languages.

General Terms

Design.

Keywords

Design, software, software design.

1. INTRODUCTION

Design has long been recognized as having a critical role in software engineering. With the ever-increasing complexity of the software systems we develop today, this role has certainly not diminished. The quality of a design can make the difference be-

tween a software system that is successfully developed, deployed, and used, and one that fails miserably somewhere along the way.

Given this critical role, one would expect the software engineering literature to be brimming with discussions leading to an extensive and ever-expanding body of knowledge on design. Some of this is indeed present. Hefty debates rage over whether UML is a design notation and, if accepted as such, whether it is an appropriate one. New design methodologies are regularly proposed, each aiming to improve upon or provide an alternative to previous methodologies and each fueling discussions on relative strengths and weaknesses. Coupling and cohesion are time-tested metrics for evaluating the structural quality of a modular design that, over time, have been refined, supplanted, and complemented by numerous other metrics. Many other examples of contributions to the field of software design and debates regarding those contributions exist.

As in any community, with each new major approach or technology typically emerges a set of strong proponents and opponents. It does not take long to find someone who swears by using UML and someone who despises it. Some methodologies so quickly establish a dedicated following, it is clear that people are jumping on a bandwagon, not critically evaluating the relative merits of the new methodology before they adopt it [9]. Find a suitable audience, and a conversation on a new metric degenerates into a heated discussion regarding the usefulness of metrics in general. Just bring up the topics of formalism, design patterns, architecture, or aspect-oriented design, and it may be that similarly charged and sometimes bitter debates emerge.

The goal of this paper is not to take sides in these debates. Rather, we wish to encourage, broaden, and enrich the debates by looking at them from the perspective of our novel *Eyeglass framework*. Based on a deep exploration of design as it is understood and practiced across a variety of disciplines, we put together the Eyeglass framework from seven closely-interrelated dimensions that are common to any form of design: (1) *ideas*, (2) *representation*, (3) *activities*, (4) *judgment*, (5) *communication*, (6) *domain of use*, and (7) *domain of materials*. In reframing software design in terms of these seven dimensions, then, the goals of our paper are:

- To provide a new theoretical framework in which we believe software design should be viewed, researched, and practiced. No longer should we treat software design as merely a phase in the software life cycle. Instead, we must treat it in light of the complex and interrelated dimensions that define design.
- To shed light on existing efforts at advancing software design and come to an understanding why some work, some do not, and others are only partially successful. Again, this does not

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE'06, May 20-28, 2006, Shanghai, China.

Copyright 2006 ACM 1-58113-000-0/00/0004...\$5.00.

mean we will settle the aforementioned debates. Instead, we provide a basis for carrying them out and moving them from what often amount to emotional arguments to discussions about these efforts' relative strengths and weaknesses in terms of the framework's dimensions.

- To bring to bear opportunities for improving software design as we know it today. The framework clearly highlights areas of research in which the community has made great strides, but also contrasts those advances with a clear indication of where our efforts have been deficient.

We believe such a reframing is long overdue. Ask a software engineer about what it takes to create a good design, and one is likely to get a description that is rather vague and primarily focuses on notations and formal verifiability. It is far less likely that you would receive an answer about strengths and weaknesses of the field, or a description of how a software designer actually goes about their task. Few other design disciplines have such trouble describing a "good design" or "a good design process", providing evidence that software design is not yet mature and does not currently have the frame of reference to reflect on itself. We hope that our contribution of the Eyeglass framework, as well as the conclusions we draw from it, will help start the discussion that is needed to move toward an established discipline.

The remainder of this paper is structured as follows. We begin, in Section 2, with a brief look at definitions. We introduce our new Eyeglass framework in Section 3, and first illustrate its use with a look at several non-software design disciplines in Section 4. We then provide a brief contextualization in terms of the framework of a variety of views on software design in Section 5. Thereafter, we place some software design research contributions in context and identify several promising and challenging research directions in Section 6. Section 7 concludes with a final word that presents our hopes for a future in which software design emerges as a mature discipline with a healthy theoretical and practical basis.

2. DEFINING DESIGN

In any profession where creative thought is needed to devise solutions to problems, some form of design decisions take place. An architect might be tasked with designing a building, or a fashion designer with the year's fall wardrobe. Beyond such obvious examples though, design, whether intentionally initiated or resulting as a side effect of a broader goal, permeates human endeavor. A department store manager must design schedules, organizing the time constraints of various employees and devising an optimal set of shifts. And the layout of such a store's merchandise must be designed to maximize purchases, and therefore profits, while also accommodating fire exits, advertisement visibility and the store's aesthetic.

But given this breadth of endeavors, what exactly is design? Many definitions have been put forward, some of which we list here:

- *"The imaginative jump from present facts to future possibilities"* (Page [25]).
- *"The optimum solution to the sum of true needs of a particular set of circumstances"* (Matchett [21]).
- *"Initiating change in man-made things"* (Jones [16]).
- *"To conceive or plan out in the mind"* (Merriam-Webster).

We will not attempt to add to these definitions, but for the pur-

pose of this paper, will simply adopt the final definition presented above, provided by Merriam-Webster. We recognize, however, the difference between the *activity of design* and the *product of design*. That is, the activity of design involves the generation of ideas and thoughts, but these must be recorded as a tangible product that can foster discussion, be evaluated for various qualities, guide implementation, and so on. We will use the term "design", therefore, in the following ways in the paper:

- *Design – the activity.* This refers to the human creative endeavor as defined by Merriam-Webster.
- *Design – the product.* This refers to the artifact that eventually results from the activity of designing.
- *Design – the discipline.* This refers to the broader notion of design as a field of study with its art, practices, understandings, conventions, etc.

Where our use is ambiguous, we will affix the appropriate term to clarify the meaning of the word "design".

3. THE EYEGGLASS FRAMEWORK

The act of designing can take on many forms. For each product that humans create, from bridges to bracelets, there are countless ways to approach that product's design. But what makes some types of products easier to design than others? What makes one approach to design more effective than another? The answers to these questions involve many factors, including, but not limited to, the type of product being designed, design tools utilized and the communication habits of the design team. These factors vary widely and must be organized in some way to enable a proper evaluation and comparison of different design approaches.

In this paper, we introduce our new *Eyeglass* framework to provide this kind of organization. To date, such frameworks have been scant. Most theoretical work focuses on meticulously defining design [16, 23] or providing philosophical descriptions of design activities as exercises of human thought [3, 31]. Practical contributions tend to either be too precise (e.g., in providing a set of "universal principles of design [20]") or not suitable as a framework (e.g., in describing our failures in properly designing a broad variety of everyday things [24]). While all of these contributions, both theoretical and practical, are very valuable, we are left with an inability to frame concrete understandings of different forms of design.

To remedy this situation, we set out to construct our Eyeglass framework as a general, yet concrete, framework that provides:

1. *a philosophical stand on what we believe are the fundamental dimensions that constitute and influence design; both as an activity and a product; and*
2. *a strong practical component in framing how to evaluate and compare design disciplines, perspectives, and technologies.*

To do so, we engaged in a deep exploration of design as it is understood and practiced across a variety of disciplines, and furthermore brought in the various leading theoretical perspectives [3, 16, 31]. The result is shown in Figure 1. At its core, the framework recognizes the definition by Merriam-Webster: design activities (the center of the right eyeglass) lead to and manipulate ideas (the center of the left eyeglass). Surrounding these two core elements of design, the framework recognizes that, first, activities

are strongly affected by the judgment that guides them and, second, ideas can be accessed and shared only after they have been put in a concrete representation. The framework further recognizes that design, in interpreting and manipulating, is a form of communication—whether with the self or with others. Finally, representation, judgment, and communication are strongly influenced by the domain of use, where lies amassed wisdom regarding the purpose for which a design solution is constructed, and the domain of materials, where lies amassed wisdom regarding the available resources from which a design solution is constructed.

Below, we discuss each of the dimensions, as well as their relationships to the other dimensions, in more detail.

3.1 Ideas and Representation

In order to understand design, we must address the nature of “design as a noun”; that is, we must examine the artifact that results from the activity of designing. While it is common to think solely in terms of a tangible result, the reality is that there is an intangible and a tangible part. At its core, a design is a set of intangible *ideas* regarding a desirable solution, as they exist in the mind of a designer. A design idea can be nearly anything and, in our framework, simply represents some degree of intellectual progress towards a design solution, consisting of decisions, preferences, rationale, facts, etc. For example, when a new villa is being designed, the decision to limit the building to one story could represent one design idea. Another design idea might be that the entire building should be constructed out of oak, to make it distinct from all neighboring villas. Although ideas are at the heart of this framework, it is not possible to dictate or predict their creation, and such is not our goal. Ideas are formed of creative processes, the shapes of which we can only hope to begin to put in context through the other dimensions of our framework.

Ideas are commonly written down, or otherwise recorded in some form, but they must not be confused with any of these tangible *representations*. Such representations are organized by our framework in a separate, albeit closely related dimension. Countless ways exist to express a given idea, whether in a sketch, textual description, formal design document, or even just in the spoken word of the designer. Because a representation is the primary form in which ideas are accessed and manipulated, the choice of representation shapes how a given designer views the ideas at hand. A well-chosen representation can make an abstract idea much easier to understand. By the same token, a poorly-chosen representation will make such understanding more difficult, and at times may even portray the image of an entirely different idea than originally intended. For example, while expressions of texture are difficult to express and easily misinterpreted with words or even images, a physical sample of an appropriately textured material makes a fine representation of such an idea. Note that representations do not have to be singular in nature, that is, they can represent multiple ideas in concert and could even portray multiple aspects of the same idea. A drawing of a newly designed clothing item, for instance, often indicates its shape, color, pattern, and sometimes even material.

3.2 Activities and Judgment

Next we will examine the nature of “design as a verb”, that is, at the *activities* in which a designer engages to generate, organize, refine, and utilize ideas. We consider each such design activity an approach to moving forward with ideas and representations,

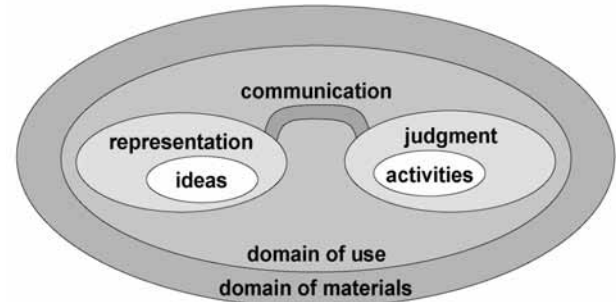


Figure 1. Eyeglass Framework.

which includes nearly any imaginable task that a designer might undertake for this purpose. For example, brainstorming, the evaluation of prototypes, quietly thinking about a design, formally proving certain qualities of a design, sketching, mentally accepting or rejecting ideas, and examining existing designs of related products are all design activities.

Just as an idea cannot be accessed without considering its representation, an activity only has meaning in the context of the *judgment* that guides it. Such judgment may take the form of broad rules, personal preferences or style, a general feeling of aesthetics, known patterns, conventional style or wisdom, and other forces that influence a person’s decision-making process. An activity, thus, determines which decisions will be considered and how they will be approached, while judgment guides the actual *deciding* inherent to those decisions. For example, in graphic design, choosing between possible images to use is an important part of the design process (the activity), and the determinations made as the result of those comparisons are based on a certain sense of beauty (the judgment).

Note that, as with ideas and representations, a dual relationship exists between activities and judgment. The type of activity being performed determines the type of judgment needed. Conversely, the judgments that one makes influence the design activities that one performs. As judgment tends to often be a mix of conscious practice and subconscious thought, it is pertinent that a designer continuously externalizes their judgment to understand how it influences their ongoing design activities.

3.3 Communication

The relationship that ties design the activity with design the product is *communication*. Communication can take on a number of forms. Perhaps one designer writes down a series of design ideas as part of a design document, and another designer reads the resulting document. In each person’s case, a design activity is being undertaken; the first person gives form to an idea and the second person uses that form to advance their own understanding of the design. We must remember that such communication need not be so linear, nor so tangible. A sketch on a napkin or whiteboard, the creation of a scale model, and even a spoken conversation are all examples of communication. In each case, the representation (i.e., the drawing, model, or spoken sound) expresses an idea or set of ideas and serves as the vehicle through which communication takes place. Furthermore, the judgment inherent to each of the various participants frames their interpretation of these ideas. In short, we are dealing with communication, the creation and sharing of information, whether with the self, with others while the

self is present, or with others while the self is not present. Each of these categories presents a unique set of challenges, and we will discuss them, in reverse order, below.

With others while the self is not present. This is the category of communication that will be most familiar to software developers. In this case, the design activity serves to record ideas so that they might be shared with others. For instance, a group of software engineers may create a design document, which will later be independently used by others, perhaps the people implementing the resulting design. Of course, the original ideas may not be transported completely or accurately from one person to the next. To understand why, it is helpful to refer to the larger structure of the Eyeglass framework. When a document's creator embarks on a design activity to write down their ideas, their judgment and the chosen representation shape the expression of those ideas. When another designer seeks to understand the ideas, their interpretation will once again be shaped by the representation as well as by their own judgment. There is a risk that the ideas will be distorted by all of these transformations. After all, the representation that is used may have been an imperfect or ambiguous expression of the initial ideas. Additionally, it may be possible that the judgment of the creator and the user of the ideas may not be 100% aligned.

With others while the self is present. One way that this risk of miscommunication can be handled is by allowing for designers to undertake design activities together. An idea that is given a permanent representation by the shared work of multiple designers, and alongside informal conversation, is more likely to be correctly interpreted when used by those designers. Similarly, if one person creates a representation of an idea and then is present when others attempt to use it, they can help to clear up misunderstandings and flesh out abstracted details.

Communication with the self. Communication also takes place when the same person creates and interprets their own ideas. This might happen when a designer doodles or sketches, simply for his or her own benefit, perhaps seeking to generate or organize ideas by recording them quickly. The advantage of such communication is that there is an increased chance that the ideas will be interpreted the way they were intended. A designer is likely to know what they meant by a given drawing, list, note, or prototype. They are also more likely to use a representation that they will find easy to interpret later. Of course, even then there is no guarantee that ideas will be communicated successfully. Consider, for example, the software engineer who returns to a design they created two years ago and cannot remember all of the details of how and why it works. As another example, consider someone trying to maintain a grasp on a complex, multi-dimensional design with a multitude of constraints, the design as a whole so large that it cannot be kept "in the mind" and can only be worked on part by part. The difficulty of communication in these cases is further compounded when we consider the fact that one's judgment may shift over time, or otherwise be applied differently at different times.

3.4 Domain of Use and Domain of Materials

Whenever a new design must be created, whether for an evening gown, vacuum cleaner, graphical advertisement, or word processor, the domain for that design brings with it a great deal of wisdom that influences the design, both as an activity and as a product. This wisdom comes in many forms, including representations expected to be used, shared assumptions, experience, proven suc-

cess strategies, folklore, and constraints of all sorts. In our framework, we represent this wisdom in two separate dimensions, the domain of use and the domain of materials, both of which encompass all of the dimensions that we have discussed so far.

The *domain of use* concerns the amassed wisdom regarding the purpose for which a design solution is being constructed. That is, it is specific to the design problem being faced and concerns such things as conventions, standards, constraints, expectations, and strategies for solving that problem. For instance, in the domain of vacuum cleaners, there is a significant body of knowledge on how such a device is supposed to look, operate, and be internally constructed. As another example, graphic designers all work almost exclusively with two-dimensional images. But depending on whether an image is to be used on the web, on a billboard, or in a print medium, different constraints and strategies will apply. Note also that different domains of use may contain or relate to one another. The domains of use of uprights, canisters, and industrial-strength vacuums, for instance, are all contained in the domain of use of vacuum cleaners in general. Furthermore, vacuum cleaners are designed with many of the same considerations as chemical carpet cleaners: both must understand the intricacies of removing dirt from carpets. But the design of vacuum cleaners also relates to the domain of use of pushed devices, wisdom shared with the design of lawn mowers or baby carriages.

The *domain of materials* concerns the amassed wisdom regarding the resources from which a design solution is being constructed. It does not need to be specific to a particular design problem, but will significantly influence the fundamental shaping of a product. A domain of materials, then, establishes a broad base of knowledge about the nature of the materials used, including the constraints that are placed on the use of those materials. A good example lies in the materials that are available to construct buildings. Using wood offers flexibility, but concrete provides the option to build very tall structures. Glass can provide an aesthetic and practical flair, but is unable to support great loads and could present a safety risk. With each choice of material, thus, come both opportunities and constraints that provide input into the design activity. The domain of materials also includes the basic understanding of the natural laws that compose these insights. An understanding of the ways that weight can be distributed through a structure, for example, is vital to the design of buildings.

Good designers are keenly aware of their domain of use and their domain of materials, as both have the potential to strongly influence the design process. However, the assumptions of these domains are not absolute, and revolutionary design ideas can result when they are challenged. Dyson disregarded commonly held wisdom from the vacuum cleaner's domain of use: the assumption that a bag be used to hold collected refuse. The Segway, meanwhile, represented a challenge to personal transport design's domain of materials. This domain included assumptions about the largely mechanical nature of such vehicles, which were flaunted in the form of a highly computerized scooter.

4. EYEGLASS APPLIED

To demonstrate how our framework can be used, we will apply it to a series of non-software design disciplines. As the framework is structured to be generic with respect to exploring and contextualizing design, we should be able to apply it to a variety of different design disciplines and gain insight into their workings. In this

section, we choose four disciplines in which design plays a significant role and apply our Eyeglass framework. While we could have chosen many different disciplines, we chose architecture, graphic design, product design, and art, because they represent a broad variety of disciplines and a broad variety of approaches to design. Clearly, in a longer paper we would treat additional disciplines and investigate them in more detail. Given that the main focus of this paper is software design, however, we necessarily limit ourselves here.

4.1 Architecture

Architecture has been a favorite for drawing design parallels to software engineering [12, 27]. Here, we focus on describing relevant characteristics of architectural design, while, for now, ignoring their parallels to software design. Figures 2a and 2b summarize the two schools of thought that we will discuss below, traditional architectural design and pattern-based architectural design. The figures use shading to indicate effectiveness in supporting design tasks: the darker a dimension in the framework, the more support that dimension provides for the creation of good design. Note that the ideas dimension is always dark, since ideas are amorphous inside the designer’s head and a discipline has no control over them.

The domain of materials and domain of use are clearly recognized in traditional architectural design, and play a strong role in the design activity. The domain of materials brings with it issues such as gravity, tensile strength, and the durability of materials. The domain of use has established its own literature, with different texts describing in detail the considerations, constraints, and approaches involved when designing buildings such as houses, schools, or casinos. Note also that both the domain of materials and the domain of use lead to a large number of implicit assumptions that are shared among designers. We all are aware of the critical role of, for instance, physics (roofs need support) or particular features needed (a house needs bedrooms, bathrooms, etc.). Such assumptions aid in the creative and communicative design tasks undertaken in architecture.

Several representations exist for different goals. Sketches are used to rapidly iterate over a divergent set of ideas, while scale models shape a few of those ideas into miniature, but faithful and tangible models that are used to review, compare, and improve candidate solutions. A broad set of formal diagrams come into play after a general solution has been settled upon, detailing much of the infrastructure that was abstracted away in the earlier phases. The specification of such details is vital, given that diagrams of this kind are often used outside of the presence of their creator.

A robust sense of judgment exists among architects, both in terms of functionality and the aesthetics of eventual solutions. While different architects may disagree over which solutions are good or bad, they at least agree on the language in which to communicate about designs and evaluate the differences that exist. Furthermore, architects have created a body of recognized architectural styles as ways to document and further converse about the differences in judgment.

Another approach to architecture emphasizes judgment to an even greater degree, while diminishing the importance of representations. Pattern-oriented architectural design proposes an alternative way of building design, one “without drawings” [2]. A pattern language of partial design solutions, which can be extended as

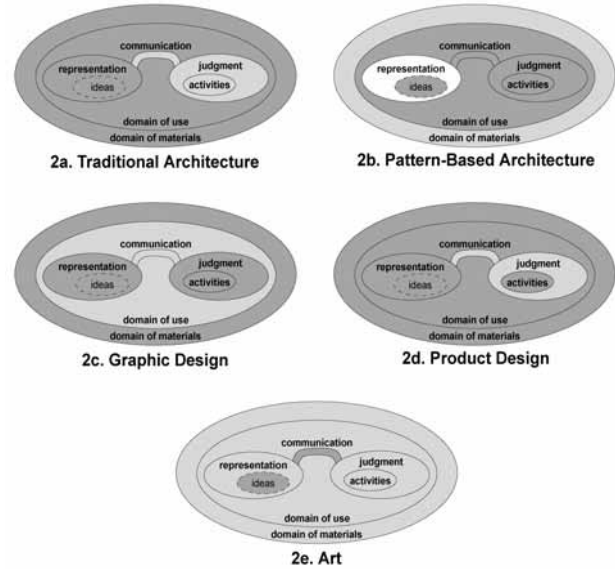


Figure 2. Eyeglass Framework Applied.

needed, is used to compose interlocking patterns into complete designs. Each pattern, such as “nine percent parking” or “windows overlooking life” consists of a name, a description of when it should be used, and advice on how it should be applied. By combining these patterns in different ways, different designs emerge, which are then implemented directly, without further specification in models or diagrams.

Note that this perspective on architecture disregards the robust representations found in traditional architecture, opting instead for simple lists of patterns. The intent is for designers and builders to work closely together and all share a common pattern language. This close-knit communication, combined with a shared sense of judgment, allows for a unified design to be created and used, even with minimal representation. The pattern-based approach strongly leverages its domain of use, as higher-level patterns integrally provide a sense of a building’s purpose. Meanwhile, the fact that the low-level details of a building’s design are not specified in advance means that these decisions can be made by the builders themselves. These builders understand the pattern language, but can finalize smaller design decisions as they see the building take shape, guided directly and precisely by the constraints of the domain of materials. This approach, then, extends design tasks out into the actual act of building: an idea that has not yet caught on in modern practice, but remains an intriguing possibility.

4.2 Graphic Design

Graphic designers are tasked with creating images, usually to be used for a commercial purpose, such as an advertisement, a CD cover, a graph for a magazine article, or a corporate logo. In the past, graphic designs were entirely paper-based creations, but nowadays such images are usually created and stored digitally. An overview of graphic design’s strengths is provided by Figure 2c.

The essence of the graphic designer’s domain of materials is made of pixels and colors, since images are intrinsically two-dimensional representations. Although file formats and kinds of paper could also be considered a part of the domain of materials,

graphic design largely creates information products, which will consist of information of a very specific type.

Various domains of use have been established within graphic design. Some are very well-developed, such as the web domain, which implies a very specific resolution and a set of preferable color palettes. Meanwhile, greeting card designers can create images of a wide variety of sizes and have the luxury of a broad selection of color palettes. Even then, this freedom is restricted by some domains of use, such as sympathy cards.

The idea representations used in graphic design take full advantage of the constraining nature of the domain of materials and the domain of use. As with architecture, sketches serve as powerful forms for exploring ideas. However, while a sketch of a building is a two-dimensional projection of a three-dimensional object, a graphic designer's sketch requires no transformation into its final product, merely a filling in of abstracted details. A sketch is easily understood, and the final state it implies easily envisioned. Moreover, available design tools such as Adobe Photoshop are entirely structured around this observation by supporting a layered model of design, allowing rapid exploration of alternatives and a staged refinement of detail [1]. Such a representation would be cumbersome in a domain of materials that allowed for aural, sensory, or three-dimensional products, but is well suited for graphic design's constrained, two-dimensional representations.

Finally, a powerful sense of judgment pervades graphic design, as numerous schools and countless books on design aesthetics and techniques can support the education of designers. But at the heart of this discipline is the fact that graphic design ideas are easily communicated thanks to the consistency of idea representations and the constraints that govern them.

4.3 Product Design

The discipline of product design spans a broad variety of artifacts, from televisions and toasters to teacups and mugs. Nonetheless, the approaches used to design these items have a great deal in common, as summarized in Figure 2d. First, we observe that three kinds of representations are ubiquitously used: sketches, models, and prototypes [18, 34]. Prototypes in particular make effective tools for evaluation; they are developed as abstractions of a potential, demonstrating many key ideas in an easily understood manner. This is especially necessary because judgment in the field of product design tends to be largely subjective. Given a particular design, one can expect a wide variety of opinions. Focus groups, trial use, and expert evaluations are therefore commonplace in absence of accepted standard criteria of beauty, effectiveness, or sales potential.

The design of a product is strongly influenced by its particular domain of use. For example, television designers are aware of modern input and output formats, as well as the high priority of high picture quality. Meanwhile, the domain of use for designing containers for hot beverages contains insight about heat insulation, handle shapes, and required durability. The domain of materials also has a strong impact on the designs that are created. The properties of specific materials are generally well-understood by designers, and can be readily tested with models and prototypes. Occasionally, a new material will develop that will fundamentally change the design of a given product, for example, the shift from analog to digital input has drastically changed the design of television sets. In general, however, these paradigm shifts are rela-

tively rare, allowing a particular product design field to mature and establish standard ways of approaching design.

A particularly interesting aspect of product design lies in its activity dimension. For some time, product design was relegated to experts in a given domain of use, but the success of such companies as IDEO in creating innovative products in a variety of fields challenges that assumption [18]. A possible reason for this success can be found in the writing of John Chris-Jones, who describes design in terms of divergent thought, which creates ideas, transformative thought, to organize ideas, and convergent thought, for arriving at a final solution [16]. The IDEO approach meets the needs of each of these activities, with a special emphasis on divergence, encouraging profuse brainstorming and the creation of ideas at every opportunity. Such an approach helps to compensate for a less developed sense of the domain of use, and demonstrates the power of divergent thought to find design solutions.

4.4 Art

While art purists may disagree, art is also a design discipline, at least in the sense described here and shown in Figure 2e. We certainly feel that most artwork is designed; a plan is laid out on what kind of art to create, what kind of materials to use, how to engage in the act of creating the art piece, etc. Consider a painter, who will usually make explicit choices about what to paint and certainly about which kind of canvas and type of paint to use. Further evidence of design in art is revealed by studies of existing paintings, which regularly reveal outlines hidden underneath the paint. Sculptors must combine careful planning with tactical decision-making while they work. By the very nature of sculpting, material can only be removed, never added, so a sculptor must ensure that each of his or her design decisions lead towards the desired result.

The artist, as designer, has few restrictions on his or her domain of materials, as nearly any object or information product might be the subject of art. Once the materials have been chosen, however, they may severely restrict the representation and thereby possibilities of expressing the artist's ideas. The domain of use also provides considerations, as different gallery spaces and audiences will be more inclined to be receptive of certain pieces of art.

Making art is traditionally seen as a solitary act, making it unique among the fields we have examined so far. Because of this, the artist is free to use whatever representations and judgment they see fit to create their work, with little concern for the need to communicate with anyone else within the design process. In many ways, this is the strength of art. While other design fields must consider the communication needs inherent to their process, an artist has the freedom to focus on the creation of the product.

Finally, while there is no need to share judgment with other designers, an artist will eventually need to consider the way that a work will be interpreted by its audience. While more commercially motivated artists might be constrained by this aspect of the judgment dimension, its influence is often minimal. In fact, one of the strengths and facilitators of art design is the fact that its appeal need not be universal, and that the flaunting of conventions is accepted, if not encouraged.

5. EYEGLASS APPLIED TO SOFTWARE

As we discussed in the introduction to this paper, software engineering researchers often have an incomplete understanding of design as it applies to their field. Some consider design to take

place in a single phase in the software process, while other approaches consider design to be “in the code”. Still other researchers have forwarded views that software development should be viewed as an engineering discipline, or as an art form. But each of these views possesses significant blind spots with regards to the essential dimensions of design, as presented in our framework. We illustrate so here, by exploring these existing perspectives and briefly discussing their contributions and shortcomings.

5.1 Software Design as a Phase

The concept of a “design phase” is extremely pervasive among software engineers. The purpose of such a phase is usually to take a requirements document and come up with a design document that will guide subsequent implementation. While this phase may be revisited a few times in an incremental or spiral approach, the view of “design as a phase” is underlying much of the modern software development process.

The problem with this concept is that the actual process of designing is not very well understood. For example, in the literature discussing the waterfall or spiral model, there is very little discussion of the specific activities that should be undertaken within the process of creating a design document, or about the ideal communication structures between designers. While some methodologies do specify ways of moving from requirements to design, this is often misleadingly billed as a purely transformational and largely mechanical activity, not as a creative process of design. And while formats such as UML exist to represent the final design document, there is very little written about how to represent ideas within the process of designing – hence our low assessment in Figure 3a.

There is some support for design in the domain of use: an understanding exists about what types of projects will benefit from specific diagram types and design structures. There is also a sense of judgment that guides work in the design phase, consisting of some basic rules, patterns [12] and heuristics [29]. But compared to other design fields, this support is still minimal, resulting in a weakly supported design process.

Nonetheless, by recognizing design as a phase, this approach must be commended for at least establishing the notion that design is an integral part of software engineering. Without this recognition, we probably would not have many of the design technologies that are in existence today.

5.2 Design in the Code

The notion of “design as a phase” was unchallenged, more or less until the recent emergence and rapid adoption of agile methods [5]. These approaches distinguish themselves by shunning design as a phase, instead preferring to program profusely, making the code itself the ultimate source for design decisions. An important offshoot of this decision is the fact that the final result of the design process in software is no longer a design document, but the program itself. We will question the wisdom of code as a design representation below, but making this choice does have some interesting implications with respect to the other dimensions of the framework.

Because coding is this perspective’s primary design task, the community’s understanding of programming, refactoring, and “bad smells” [11] can, indirectly, be leveraged as support for design activities and judgment. Similarly, research aimed at improving the coordination of work among programmers can now be

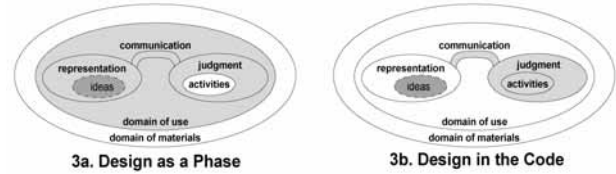


Figure 3. Eyeglass Framework Applied to Software.

viewed as enhancing our understanding of design communication. The practice of pair programming is especially effective from our framework’s perspective, allowing code to be used “with others while the self is present”. Additionally, because code is the design, many people participate as designers, which is more realistic than the “design as a phase” view in which it is assumed one, or at best, a few designers make all decisions. Programmers continuously make decisions that influence a design (as evidenced by the problem of design deviation [26]). Agile approaches recognize this and attempt to address it with a focus on frequent examination of code structure and subsequent refactoring.

That said, from the perspective of our framework and its implications for design, code is a terrible choice as a design representation. It does not support any form of rapid and creative exploration; after all, we must program and build something working before we can truly evaluate it. Furthermore, while the *functionality* of a program is clearly expressed in its code, the actual, creative design *ideas* that the program embodies are often quite obscured, leading to a needlessly difficult design process.

Figure 3b summarizes these points. Agile development represents an interesting alternative as it allows for a much more appropriate view of design’s role in software development (e.g., everybody is involved, the design activity is explicitly recognized, design communication is stressed), but unfortunately the choice of code as the medium for design negates many, if not all, of these potential benefits. Nonetheless, it provides us with an interesting lesson in that “design as a phase” is not the only possible perspective; perhaps if we can blend the better parts of several of these perspectives, a more suitable one can be found.

5.3 Software as Engineering or Art?

Beyond the debate of how we should carry out the creation of software lies a debate that is, in many ways, much larger. For some time, there has been disagreement about what the fundamental nature of software engineering really is: is it an act of engineering, or an act of art?

There are appealing arguments on both sides. The incredibly complex, mathematical nature of a large program seems to cry out for a standard set of rules and practices. This view is reflected in Shaw’s call for software engineering to be more like a mature engineering field [32] and the Software Engineering Body of Knowledge’s stated “engineering-based approach” [4]. Alternately, software engineering can be seen as art, requiring the creation of novel solutions that are communicative and elegant. Brooks, for example, in describing great designers, emphasizes; “software construction is a *creative process*” [7].

Davis, however, provides us with an alternative viewpoint, observing that during a software project, elements of engineering are needed at some times and elements of artistry at others [9]. He then engages in a comparison of software development to custom home design to illustrate his mixed point of view.

Perhaps then, rather than engineering or art, the most appropriate metaphor for software development is that it is a design task. This is not incompatible with the other views presented in this section, but in many ways helps to unite them, for design contains elements of both engineering and art. An artistic side provides us with the activities that will create ideas, while a sense of engineering is needed to understand the practical implications of those ideas, and to devise representations to communicate them precisely. Both art and engineering contribute important elements of judgment; a sense of aesthetics, as well as practical decision-making support, is vital to an effective design process. By framing software as a design discipline, we can see these disparate attitudes largely reconciled.

6. ACCOMPLISHMENTS & CHALLENGES

Now that we have conceptually examined software design from a very high level, it is time to examine how the Eyeglass framework concretely helps us focus our efforts at the level of detailed research directions, approaches, and tools. To do so, we revisit each of the dimensions of the Eyeglass framework, briefly discuss contributions within that dimension, and lay out some fundamental research questions that we feel must be addressed. Finally, we will discuss the issue of design's role in the software process, a concern that crosscuts the framework's dimensions.

6.1 Ideas and Representation

The *ideas* dimension of software engineering, as with any field, is beyond improvement. We cannot change the minds of designers. We can only provide them with the context to generate good design ideas. This, of course, is the essence behind our framework so it should be no surprise it applies here as well.

The first dimension of this context is formed by the representations that can be used for expressing design ideas. This has been an area of much research. Models of software, and the languages in which to express them, have been an integral part of the research pursuit since the inception of the software engineering field. In practice, too, models and modeling languages have received a large amount of attention. The result has been a continuous flurry of notations, from early efforts such as data flow diagrams [10] and structured diagrams [15] to newer results such as UML and architectural description languages [22].

Judged from a design perspective, the dimension of representation in software design is ripe for improvement in two different ways. First, we must improve existing notations. As an example, Perry and Wolf got it right when they led a call for “elements, form, and rationale” as a basis for modeling software architecture [27]. This vision resonates strongly with our framework, but has not yet materialized. Study any existing architecture description language, and one finds a strong focus on elements, a minor focus on form (with styles), but a lack of rationale [19]. Even then, the modeling of elements resembles code, which is not likely to be conducive to a creative design exercise. Support for such exploration was actually much more prominent in earlier modeling languages, exactly because they were less formal, further removed from code, and more intuitive to understand [10, 15]. In terms of supporting creativity, we seem to have moved backwards since then.

Suggestion 1: We must create new modeling languages that support design in all of its dimensions, and do not just focus on docu-

menting a design after it has been thought out. That is, we must create design representations, not programming abstractions.

Representation also has the potential to be improved if we explore additional forms of expression. Software is extremely exact but this does not mean that our representations of it need to be. Particularly from a communication point of view, say during a brainstorming session, we cannot be expected to just draw UML diagrams on a whiteboard. Many designers have informally adopted other means of designing, sketching, drawing different concerns at different levels of abstraction concurrently, erasing and redrawing, and so on. This implies a highly tangible activity, which may be better served by a highly tangible representation. Sketching is a beginning, but greater conceptual support and tool support for activities of this kind is needed. Other disciplines have found useful and effective representations that are structurally different from the final product (e.g., the scale model, 3D visualization). Even if it means a temporary loss of precision, so must we search for ways to make software design more tangible.

Suggestion 2: We must explore tangible representations of design, moving away from the notion that the only way in which we can express a software design is in a language of some form.

6.2 Activities and Judgment

Software design as an activity has actually received relatively little attention. While one certainly cannot expect a detailed process specification that, when followed, would unequivocally lead to a high-quality design, this is perhaps not what is necessary. To date, we have seen a large emphasis on design environments, tools with which we can graphically lay out a design [28]. These, however, have somehow ended up being documentation tools (powerful ones at that, we admit). There has been some progress in terms of guidance via critics and constraints [30], but these tools fail to truly support creative exploration. Compare them with Adobe Photoshop, which is used for graphic design. Its underlying conceptual base and method of interaction are both directly geared towards creative exploration. By analogy, we must develop lightweight approaches to generating, storing, and manipulating ideas, whether they are represented as text, physically, or as a carefully abstracted drawing. These approaches may come in the form of new notations and tools, but cannot be merely incremental improvements over current tools. We must re-envision them to integrally support the creative act of design. Software storming [17], heuristics [29], and sketching languages [13] are early examples.

Suggestion 3: We must create new modeling tools that support a designer in their creative activity, that is, integrally support the divergence – transformation – convergence model of design that is put forward by John Chris-Jones and inherent to exploration.

Judgment has long been a factor in software engineering and design. In fact, one could argue that metrics are a form of judgment, as exemplified by the ubiquitous understanding and use of coupling and cohesion. Software patterns [12] and architectural styles [33] are another form of judgment. Both codify standard “good” ways of organizing a design. Patterns are especially interesting, since they have penetrated the broader software engineering community, thereby establishing a sense of shared judgment, which is key to the maturity of any design discipline.

That said, our judgment could still be improved significantly. The number of metrics governing design is small and focused on structure. How a software performance person looks at a design is very

different from the way a scalability engineer does. Exactly what a designer looks for when interpreting a design and declaring it “evolvable” is unknown; we still know little compared to other design disciplines about what makes us appreciate one design and reject another outright.

Suggestion 4: We must pursue a sense of software aesthetics to provide a way of evaluating the quality of a design from different dimensions. This aesthetics must allow for different views, criteria, needs, and alternative bodies of design appreciation.

Note that this sense of aesthetics will not be born solely from the availability of lots of metrics and patterns. These form an essential part of judgment, but we also must develop an intuitive sense, as taught and learned via numerous examples and great discussion.

6.3 Communication

The careful reader will observe that all four suggestions thus far strongly relate to communication, either with the self or with others. This reflects the essential role that communication plays in our framework: without it, design simply would not happen.

The literature affirms this theoretical position with concrete observations and observes that design, as a critical medium of communication, must adhere to principles of clarity, non-ambiguity, etc. [35]. When we design new notations, then, this role should be kept in mind and understood for its implications on those notations. But when push comes to shove and we actually develop these new notations and sometimes other representations, it is rare we find someone speaking up for communication needs; it takes a backseat to issues such as meta-modeling, formality, and expressiveness.

Better representations are not the only way to improve communication. Shared judgment helps in providing a context in which some parts of an otherwise complex idea can be omitted. Metrics and patterns are examples of improvements that indirectly improve communication by helping the interpretation of design artifacts to remain consistent. But too often precision (metrics) or generality (patterns) is the focus of the creation of these artifacts; communication, again, takes a backseat.

Suggestion 5: We must make communication the primary objective when devising new software design tools, methodologies, or approaches, and examine our contributions in this light – that is, in terms of usability, communicativeness, speed of communication and interactivity.

6.4 Domain of Use and Domain of Materials

The domain of use in software design has received some attention. We know certain design languages are better for certain domains [35], we have explored Domain-Specific Software Architectures [36] (now called Product Line Architectures [6]), and we are experimenting with generative programming (e.g., aspect-oriented programming, model-driven architecture, multi-dimensional separation of concerns) [8]. In addition, certain design methods attempt to work on bridging the gap from the domain (specified in requirements) to the design, as is the case with the Problem Frames approach [14]. These are all good forays into leveraging the domain of use for purposes of design.

Compared to other disciplines, however, an interesting phenomenon has occurred. As a discipline, software engineering seems to have decided on remaining as generally applicable as possible.

We have resisted that which has strengthened other design fields: partitioning of the design approaches depending on different domains of use. We recognize the differences between designing a web application and next-generation space flight software, but thus far the literature has not started to reflect this in practical terms. We should not be afraid of partitioning into different subdomains, as it is a normal maturing of our discipline and stands to bring great benefits in terms of the design process.

Suggestion 6: We must explore a much greater role of the domain of use in the shaping of our discipline, allowing subdomains to emerge that each address specific classes of design problems with domain-specific representations, design tools, aesthetics, etc.

Our current domain of materials is interesting because of its stunning simplicity yet amazing deficiency: code is simply not a good domain of materials! In the end, code is what frames our designs, more so than we may wish to admit. Even our representations are intimately tied to code. Object-orientation is currently the dominant approach to design; yet, it is originally a code-level concept that has made its way up to the level of design. So it is with many design modeling concepts. Our design languages end up still being programming languages, with shared assumptions and limitations. This makes us dependent on an entirely different community, which does not necessarily share our vision of what design is to be. This can no longer continue. The essential material of which our programs are built is something that demands our attention and innovation. Fourth generation languages, in a way, are an example of what is possible: databases are designed on the spot with instant generation of prototypes. The key is the domain of materials, which is turned into fill-out forms. While code is underneath, it is hidden as a material. Of course, the domain of use helps a lot in this case, as it provides the constraints on the types of program for which this approach could effectively materialize.

Suggestion 7: We must take matters into our own hand with respect to the development of programming languages. We, as the software engineering community, need to get strongly involved in the definition of new programming languages and other forms of material so they suit our needs: those of a design discipline.

6.5 Crosscutting Concern – Process

A final consideration in the exploration of our framework lies in the timing of when design happens. From the discussion in Section 4, it should be clear that design cannot be confined to a single phase, nor can it be confined to a single kind of artifact. The reality is that we make design decisions in virtually every part of the software process. Despite the ideal separation of what versus how, the requirements engineer makes choices that constrain a design. A design does not fully constrain the implementation; a programmer will be making choices that are design choices in nature. Even these tasks do not involve the traditional design document; they still involve choices about how to implement functionality that later on may end up influencing the design document nonetheless. So, we must truly step out of the box and begin to understand that all of software engineering is a design process. Only then can we garner the true value of the framework we have presented in this paper: it provides the proper perspective for software as a design discipline.

Suggestion 8: We must treat the entire software process as a design process. Decisions can be made at each stage that constrain or otherwise influence the design – we should treat them as such!

7. CONCLUSIONS

It is our hope that this paper sparks a renewed interest in the topic of software design. Theoretical frameworks, views, and treatises have been relatively scant in our field to date. Our novel Eyeglass framework, in putting forward a new and different view of software design, is an attempt at: (1) organizing our understanding of software design as a design discipline, (2) highlighting where our efforts to date have been successful and where they have fallen short, and (3) identifying a set of research goals that we must address to advance software design and realize its potential as a design discipline.

Much work is to be performed, not just in addressing the concerns and potential research directions raised by the framework, but also in beginning and sustaining a deep theoretical discussion on the nature of software design. Examining other disciplines, one finds vocabularies, articulated theoretical models, shared values on the interpretation and evaluation of alternative designs, proven strategies and approaches, useful and effective tools that are explicitly geared towards the tasks at hand, and other aspects and signs of highly-engaged communities. For software design to mature to a similar level, we must dare to step away from the technical world in which we have so far preferred to engage.

We hope our framework is not the last, and in fact we anticipate it is not! It is only through involved discussion, reframing the reframed, and creating a multitude of possibly conflicting perspectives that we will be able to advance software design beyond the hurdles that hold us in place today.

8. ACKNOWLEDGMENTS

Effort partially funded by the National Science Foundation under grant numbers CCR-0093489 and IIS-0205724.

9. REFERENCES

- [1] Adobe Software Institute, <http://www.adobe.com/>.
- [2] Alexander, C. *The Timeless Way of Building*. Oxford University Press, New York, 1979.
- [3] Arnheim, R. *Visual Thinking*. University of California Press, Berkeley, CA, 1969.
- [4] Bagert, D., Hilburn, T., Hislop, G., Lutz, M., McCracken, M. and Mengel, S. *Guidelines for Software Engineering Education Version 1.0*. CMU SEI, Pittsburg, PA, 1999.
- [5] Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, MA, 1999.
- [6] Bosch, J. *Design and Use of Software Architectures*. Addison-Wesley Professional, Reading, Massachusetts, 2000.
- [7] Brooks, F. *The Mythical Man-Month*. Addison-Wesley, Reading, Massachusetts, 1995.
- [8] Czarnecki, K. and Eisenecker, U. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley Professional, Reading, Massachusetts, 2000.
- [9] Davis, A. *Great Software Debates*. John Wiley and Sons, Inc., Hoboken, 2004.
- [10] DeMarco, T. *Structured Analysis and Systems Specification*. Prentice-Hall, Englewood Cliffs, New Jersey, 1978.
- [11] Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, Reading, Massachusetts, 1999.
- [12] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison Wesley Professional, Reading, MA, 1994.
- [13] Hammond, T. and Davis, R., LADDER: A Language to Describe Drawing, Display, and Editing in Sketch Recognition. *International Joint Conference on Artificial Intelligence*, 2003.
- [14] Jackson, M. *Problem Frames*. Addison Wesley, 2001.
- [15] Jackson, M. *System Development*. Prentice Hall, Englewood Cliffs, N.J., 1983.
- [16] Jones, J.C. *Design Methods*. John Wiley and Sons, Inc, New York, 1970.
- [17] Jordan, P.W., Keller, K.S., Tucker, R.W. and Vogel, D. Software Storming: Combining Rapid Prototyping and Knowledge Engineering. *Computer*, 22 (5).
- [18] Kelley, T. *The Art of Innovation*. Doubleday, New York, 2001.
- [19] Lago, P. and Vliet, H.v., Explicit Assumptions enrich Architectural Models. in *27th International Conference on Software Engineering*, (St. Louis, Missouri, 2005), IEEE, 206-214.
- [20] Lidwell, W., Holden, K. and Butler, J. *Universal Principles of Design*. Rockport Publishers, Inc, Gloucester, 2003.
- [21] Matchett, E. Control of Thoughts in Creative Work. *The Chartered Mechanical Engineer*, 14 (4).
- [22] Medvidovic, N. and Taylor, R.N. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE TSE*, 26 (1). 70 - 93.
- [23] Meggs, P. *Type and Image : The Language of Graphic Design*. John Wiley & Sons, Inc., 1992.
- [24] Norman, D. *The Design of Everyday Things*. Basic Books, New York, 1988.
- [25] Page, J.K., Conference Report. in *Building for People*, (London, 1965), Ministry of Public Building and Works.
- [26] Parnas, D., Software Aging. in *16th International Conference on Software Engineering*, (Sorrento, Italy, 1994).
- [27] Perry, D. and Wolf, A. Foundations for the Study of Software Architecture. *ACM Software Engineering Notes*, 17 (4). 40-52.
- [28] Quatrani, T. *Visual Modeling With Rational Rose and Uml*. Addison-Wesley, 1997.
- [29] Riel, A. *Object-Oriented Design Heuristics*. Addison-Wesley, Boston, MA, 1996.
- [30] Robbins, J. and Redmiles, D. Software Architecture Critics in the Argo Design Environment. *Knowledge-Based Systems*, 11 (1). 47-60.
- [31] Schön, D. *The Reflective Practitioner*. Basic Books, 1983.
- [32] Shaw, M. Prospects for an Engineering Discipline of Software. *IEEE Software*, 7 (6). 15-24.
- [33] Shaw, M. and Garlan, D. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., Upper Saddle River, 1996.
- [34] Söderman, M. Comparing Desktop Virtual Reality with handmade sketches and real products. Exploring key aspects for end-users' understanding of proposed products. *The Journal of Design Research*, 2 (1).
- [35] Sommerville, I. *Software Engineering (7th Edition)*. Addison Wesley, Reading, MA, 2004.
- [36] Tracz, W., Coglianesi, L. and Young, P. Domain-Specific Software Architecture Engineering Process Guidelines. *ACM SIGSOFT Software Engineering Notes*, 18 (2). 40-49.

