Framing Software Design with the Design Diamond

Alex Baker and André van der Hoek
Institute for Software Research & Department of Informatics
University of California, Irvine
Irvine, California 92697-3440, U.S.A.
+1 949 824 6326
{abaker, andre}@ics.uci.edu

ISR Technical Report # UCI-ISR-06-11
July 2006

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 "just is not quite right" and that design remains one of the least-understood aspects of software engineering. This paper contributes the *DESIGN DIAMOND*, a new framework that examines software design from the novel perspective of the individual software designer. Taking this perspective turns the DESIGN DIAMOND into an unbiased instrument to view software design and allows us to compare the ways in which software design is typically positioned, evaluate the field's contributions to date in supporting the practice of software design, and lay out a comprehensive research agenda for improving the state of the art.

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, design diamond.

1. INTRODUCTION

Design has long been recognized as having a critical role in software engineering. With the ever-increasing size and complexity of the software systems we develop today, this role has certainly not diminished. The quality of a design can make the difference between a software system that is successfully developed, deployed, and used, and one that fails utterly somewhere along the way.

Given this critical role, it is no surprise that the software engineering literature is brimming with a multitude of proposals and opinions regarding design approaches, modeling notations, evaluation techniques, and other innovations, experiences, and ideas. To date, though, this otherwise healthy debate largely has taken place without a proper frame of reference. Granted, it is certainly possible to discuss modeling notations in terms of expressiveness or verifiability. Similarly, one can contrast design environments in terms of the features supported or design metrics in terms of the concerns addressed. But once one crosses such individual "pillars of knowledge", our overall understanding of software as a design problem, and consequently our ability to effectively reason about any solutions, becomes much murkier.

What are the factors at play in software design? How do these factors influence our current approaches? As a field, where are we excelling and where are our efforts falling short? What are the most promising avenues for improving software design? Are our contributions all working towards a common goal, or are there differences that render some approaches and ideas essentially incompatible? What, in fact, should be our goals?

All of these are critical questions that demand answers. Many such answers have indeed been provided, but they have typically been rooted in highly individual perspectives of an ideological, theoretical, or practical nature. The result is a melee of facts, opinions, and beliefs, some of which agree, others of which vehemently disagree, some of which offer valuable wisdom, others of which miss the boat completely. While this works to some degree in slowly advancing our understanding of software design and improving our ability to practice it, it is not a sign of a mature discipline and not the most productive way of moving forward. To promote effective discourse and attain measurable progress, the discipline of software design reach a state in which one can position, relate, and judge individual contributions to its literature within a common understanding of software design's nature.

As a first step into this direction, this paper contributes the *DESIGN DIAMOND*, a novel framework that offers a new, holistic perspective on software design. The DESIGN DIAMOND identifies seven dimensions of concern (domain of materials, domain of use, knowledge, ideas, representation, activity, goal) and four types of relationships (manipulates, informs, captures, and enhances) that, together, define the elementary forces at work in a design process. We identified these forces based on a deep exploration of design as it is practiced across a variety of disciplines consulting numerous texts from a variety of design fields (e.g. [1, 2, 13, 16, 25]). As such, the DESIGN DIAMOND distinguishes itself from previous attempts at characterizing software design as follows:

- *It is design centric, not software centric.* We want to ensure that our notion of software design is aligned with other notions of design. That is not to say that software should be designed in the same way as automobiles or teacups. Rather, it is to say that many of the forces at work when software is designed are the same forces that are at work when other products are designed. Only when we first recognize these forces can we support them according to the unique needs of software.

- *It is designer centric, not lifecycle or artifact centric.* One challenge of discussing software design is that its role is defined differently in different software lifecycle models. Each model delineates its own design tasks, its own inputs to be utilized, and its own design artifacts to be created. If one model's assumptions are adopted, the conclusions drawn may not apply to projects that utilize other models. Accordingly, we will decline allegiance to any particular lifecycle model or design artifact here, and instead attempt to consider the essential act of designing a software solution. In particular, we will seek to understand the needs of software designers themselves, and the challenges inherent to solving software problems. This lifecycle-neutral perspective will give our observations the widest possible scope, and allow us to comment on the lifecycles themselves in an objective way.

An important contribution is that the DESIGN DIAMOND embraces existing general design theories. These theories are represented as facets in the DESIGN DIAMOND, covering relevant subsets of the dimensions of concern and their relationships. Throughout the paper, these facets will be the primary vehicle through which we will study the DESIGN DIAMOND.

As with any framework, the value of the DESIGN DIAMOND lies in its application. Our model provides some degree of insight through its identification of design dimensions and their relationships, but we must also recognize the opportunity for analyzing software design in detail and deriving recommendations for further research. This paper, then, is organized in two halves. The first half introduces the DESIGN DIAMOND, its structure, and its relationship to general design theories. The second half applies it in three different ways. Particularly, we use it to: (1) examine current choices of positioning design in the software development process, (2) evaluate existing approaches' ability to support software design, and (3) identify promising research directions for advancing the discipline.

2. DEFINING DESIGN

In any profession where creative thought is needed to devise solutions to problems, design occurs. An architect may be tasked with designing a building, or a fashion designer with the year's fall wardrobe. Beyond such obvious examples though, whether consciously undertaken or resulting as a side effect of a broader goal, design permeates much of 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, 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 [17]).
- *"The optimum solution to the sum of true needs of a particular set of circumstances"* (Matchett [14]).
- *"Initiating change in man-made things"* (Jones [10]).
- *"To conceive or plan out in the mind"* (Merriam-Webster).

It is interesting to observe that, while these definitions all aim to define the same term and have similar undertones, they really are not all that similar in their meaning. Careful study of the definitions reveals that each illustrates a different aspect of design, as rooted in the perspective taken by each author. This indicates that design indeed is a multi-dimensional concept. We therefore refrain from adopting a particular definition of design in this paper, preferring to let our DESIGN DIAMOND and its explanation delineate what we consider design and what we do not consider design. This decision is not meant to diminish the value of the definitions provided above. On the contrary, each has provided important insight into an aspect of design, and has helped to guide the construction of our model.

Nonetheless, the word "design" is used throughout this paper and must exhibit a clear meaning. Our use will be threefold:

- *The activity.* "Design" may refer to the act of designing, the endeavor that designers undertake.
- *The artifact.* "Design" may refer to the tangible artifact that is the result of a design process.
- *The discipline.* "Design" may refer to the broader notion of design as a field of study with its practices, understandings, and conventions.

Where the context of our use of "design" may be ambiguous, we will affix the appropriate term to clarify its intended meaning.

3. THE DESIGN DIAMOND

To create the DESIGN DIAMOND, we began with a conceptual step back. Instead of immediately delving into software design, we spent over a year studying design literature that explicitly was not related to software. We examined general design theories, design principles in various fields, design approaches in other fields, numerous examples of specific designs and properties of those designs, and generally attempted to reach both a broad coverage of the many sides of design. Only then did we return to the software design literature at large.

Throughout, we identified essential differences among the various design domains, as well as common themes, shared concerns, applicable metaphors, and several other kinds of evidence that would help us understand the nature of design. The result was a broad set of individual observations, to be put together in one form or another. By organizing these observations, we gradually converged on the DESIGN DIAMOND that is presented in this paper. We note that, because of the process followed, the DESIGN DIAMOND is not specific to software. In fact, we believe it can be used to study any kind of design. But we will not defend or further justify the DESIGN DIAMOND'S generality here, instead focusing on its use to help study and understand software design itself.

Figure 1 illustrates the DESIGN DIAMOND in its entirety. It consists of seven dimensions of concern (domain of materials, domain of use, knowledge, ideas, representation, activity, and goal) and four types of relationships (manipulates, informs, enhances, and captures). These dimensions and relationships, together, form the basis with which we define the elementary forces at work in a design process. In the remainder of this section, we discuss first these seven dimensions of concern and then the four kinds of relationships that tie these dimensions of concern together.

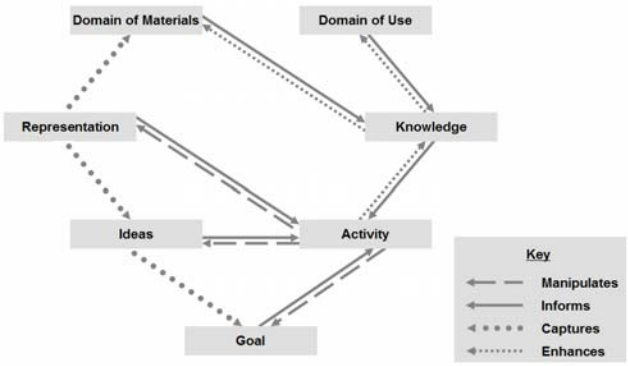

Figure 1. The DESIGN DIAMOND.

3.1 Dimensions of the DESIGN DIAMOND

The DESIGN DIAMOND considers design from the perspective of an individual software designer. By being neutral with respect to the particular development approach to be followed and the design artifacts to be created, the DESIGN DIAMOND allows us to understand how these factors influence the actual task of designing. We should also note that designer to designer communication is not addressed directly as a dimension in our model. Rather, as we discuss in Section 4, we consider communication to be a facet of our model, which involves several other basic elements of design.

Below, we precisely define each dimension of concern, provide an intuitive explanation, and include a few examples of the kinds of elements one typically will find in that dimension.

- *Goal (GL): objectives that frame a design problem.* Any design process starts with an initial goal that frames the problem to be addressed. Such a goal may be intentionally vague, incredibly precise, singular, composed of numerous subgoals, explicitly written down, hidden in unstated expectations, driven by external parties, or set by a designer themselves. The canonical example for software design is a requirements document, but other examples include a contract, an open source developer setting a goal for themselves to improve the structure of an existing piece of code, or an expectation that a newly designed and implemented web application indeed leads to increased sales.
- *Activity (ACT): acts that contribute to the crafting of a design solution to a design problem.* A design does not appear out of the air, but is the result of actions that the designer undertakes to progress towards a desired design solution. Such actions may include brainstorming, creating a UML class diagram, evaluating a prototype, starting over, sitting and pondering, generating alternatives, or performing research in the problem domain. Note that not every action directly results in a design artifact; many actions are peripheral and serve to prepare the designer for other, more concrete, tasks.
- *Ideas (ID): individual understanding of a design problem and its solution.* Throughout the design process, the designer builds up and refines an understanding of the design problem and its solution. This understanding is intangible, consisting of some set of ideas existing inside the mind of the designer. These ideas can be vague intuitions, firm decisions, relations among ideas, thoughts on possible directions, rationale, preferences, facts, recall of previously-visited alternatives, and so on. A software designer's ideas might resemble "we should use a pipe and filter architecture on this project", "that proposed solution will not scale up", "the customer said this, but from user studies I know that it is not quite true" or more abstract mental images of the program's eventual operation.
- *Representation (REP): expression of an understanding of a design problem and its solution.* A representation gives form to an understanding that a designer has of a design problem and its solution. That is, it takes the intangible ideas and expresses them in some kind of *design notation*, broadly defined to include forms such as a sketch, a memo, a diagram, a whiteboard drawing, a narrative, a sequence chart, a spoken conversation, or a high-level architectural description. The ways in which a given idea can be expressed vary greatly depending on the notation chosen, as different notations can be more or less suited to specific design situations.[1]
- *Knowledge (KN): individual wisdom about design problems and their solutions.* In addressing a particular problem, a designer draws upon and is guided by their experiences, judgments, results, and insights that they have amassed over time. As compared to ideas, which only relate to the design problem at hand, knowledge represents the broader wisdom that a designer possesses about designing in a given general context. Examples of knowledge include familiarity with some architectural style, the experience of designing multiple large databases and the intangible intuitions and insights that come with that experience, and knowing that a certain design structure is suitable for software that needs to be highly reliable.
- *Domain of Use (DOU): collective wisdom about design problems and their solutions.* An individual seeking to improve their knowledge of design can often do so by studying the collective wisdom of a design field. All software designers have knowledge that may benefit others and ideally that knowledge should be readily available for others to utilize. This collective wisdom is generally not reported in "raw" form, but typically is summarized, coalesced, and organized in books, catalogues, standards, educational materials, conferences, folklore, online resources, and so on. Examples are the OSI seven layer network model, the Design Patterns book [7], and the numerous, collected rules to keep in mind when designing user interfaces [23].
- *Domain of Materials (DOM): collective wisdom about the resources available to implement design solutions.* Designers cannot work purely in terms of abstract ideas, but always must keep in mind the materials which will be used to realize their design. A building designer uses their knowledge of materials such as steel, wood, concrete, and bamboo in designing a house. Analogously, software designers should consider the available resources for implementing a design,

[1] "Representation" sometimes is used in the literature to refer to a design notation. In this paper, we use the convention that *a representation* is a specific design expression (i.e., a UML diagram, a conversation, a sketch) in a given generic *design notation* (i.e., UML, English language, paper and pencil).

such as programming languages, web protocols and standard tools, user interface generators, and databases.

3.2 Relationships in the DESIGN DIAMOND

In constructing the DESIGN DIAMOND, it rapidly became clear to us that the individual dimensions of concern were related to each other in a variety of different ways. These relationships have been distilled down to the four described here, which together help shed light on the forces at work when one designs.

3.2.1 *The Manipulates Relationship*

Whenever design progress is made, some sequence of actions is responsible. The manipulates relationship of the DESIGN DIAMOND (shown in Figure 2a) indicates where the effects of this process are felt. The first two such relationships are straightforward: our ideas and representations are continuously manipulated throughout the entire design process to record our progress as intangible thoughts and tangible expressions, respectively. The third relationship, from Activities to Goal, corresponds to the augmentations, shifts, refinements, and even outright restatements of the initial goal that one set out to satisfy. Enacting such shifts may be difficult if the initial goal was set by another party, but such occurrences nonetheless take place.

3.2.2 The Informs Relationship

The informs relationship, shown in Figure 2b, illustrates the flow of information that helps a designer in performing their tasks. The six arrows can be partitioned into a group of four direct influences and a group of two indirect influences. The direct influences all effect the activity dimension (from Goal, Knowledge, Ideas, and Representation, respectively), representing the four sources that shape a designer;s next steps. First, the designer clearly must keep their overall goal in mind; after all, this represents the impetus for designing in the first place. Second, the designer is guided, whether consciously or unconsciously, by their amassed knowledge and experience. This permeates one's thinking process, and thoughts from the past often become interwoven with ideas regarding the current project. Third and fourth, any ideas and representations that are generated over the course of a project play a vital role in informing the activities that will be taken next.

The two indirect informs relationships represent the individual designer's process of learning from the Domain of Materials and the Domain of Use. Because of this learning, the decisions that the designer makes will be influenced, indirectly, by the community's concept of good practices and judgment. In this way, the work of the community as a whole can often achieve a level of consistency and quality, built on the strength of its shared domains.

3.2.3 The Captures Relationship

The captures relationship, shown in Figure 2c, explains where and how project-specific dimensions are obligated to one another and how these obligations create challenges within the design process. The first relationship, from Ideas to Goal, reflects that most ideas that are generated by a designer must, in one way or another, address the goals that they set out to achieve. The second relationship, from Representation to Ideas, reflects that any design expression must be faithful to the ideas in the designer's head. Together, these two relationships highlight one of the traditional difficulties in software engineering: a final representation of a design must be in accordance with its goals, but this is a *two-step* process, with ideas being generated in between. Often, this is a source of serious inconsistencies.

The third and final captures relationship is from Representation to the Domain of Materials. This relationship states that, in order for a design to eventually be successfully implemented, it must be faithful to the materials that are used in the design. If the design makes assumptions that are not accurate (for instance, that a Java interpreter will be able to execute a module at a certain speed or a module can be implemented in a certain way), the design will be of limited value to those tasked with implementing it.

The two captures relationships originating from the Representation dimension indicate another fundamental tension that underlies software design: the closer a design notation allows a designer to approximate the final implementation in its representation, the more difficult it usually is to verify whether the design adheres to its stated goals. Conversely, the closer the design is to its high-level goals, the more difficult the translation step to an actual implementation will be. Because of the large cognitive distance between the goals of a software project and its implementation, this is a significant problem, as reflected in the regular use of multiple design notations (e.g., first modeling an architecture and then refining it into UML).

3.2.4 The Enhances Relationship

The previous three types of relationships have illustrated ways in which an individual design project is influenced. Shown in Figure 2d, the enhances relationship complements these three types of relationships by explaining how experience gained during the design process filters back "up" for later use. This begins whenever a designer performs an activity, since they may learn something that has applications beyond the design problem at hand. It continues when the build-up of a designer's internal knowledge culminates in insights that are useful to the rest of the community. In such cases, a developer may take explicit action to publicize their knowledge and make it flow up to the Domain of Materials or Domain of Use. This reverse learning process is a critical process for our community and we return to the topic in Section 4.

It is worth noting that there is not a similar arrow connecting Representation to Knowledge or Ideas to Knowledge. Such learning is highly dependent on the activity undertaken. A project's UML

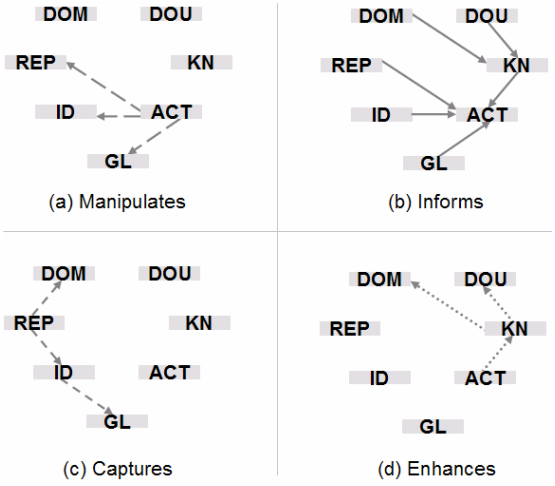

Figure 2. Four Types of Relationships.

diagrams do not directly contribute to a designer's broader knowledge, but they inform an activity such as "reading", "discussing" or "reflecting", through which the actual learning occurs..

4. DESIGN THEORY: FACETS

The broad field of design has been studied by a number of authors in the hopes of establishing general theories of design. While such theories are not abundant, there are several well-known, important generalizations that apply to software design as readily as to other sorts of design. In this section, we introduce these theories and show how they are captured by the DESIGN DIAMOND as *facets*, each of which pertains to a subset of the dimensions and relationships that define the DESIGN DIAMOND. These facets are depicted in Figure 3 below, where the dimensions and relationships that are most relevant to each facet are highlighted.

The Reflective Conversation with Materials facet and the Activity Type facet describe existing theories outright. Additional applicable theories and ideas were grouped together into three other facets: Pure Thought, Team Communication, and Community.

Below, we discuss each of the facets, beginning with the thoughts in the head of an individual designer and then moving outwards, exploring an individual's use of representations, team communication, and finally the growth of design communities. Each facet is explained generally; we defer most comments about software to Section 6, where we use the facets to evaluate software directly.

4.1 Pure Thought

The Pure Thought facet, shown in Figure 3a, describes the most primal of design activities, the purely mental generation, organization and evaluation of ideas [10]. Nearly every established design activity recommends the use of a representation, other people, and outside sources of knowledge. Underlying these activities, however, is a cycle of thought that is subconsciously fueled by a designer's own knowledge and understanding of the project's goals. This cycle, in essence, represents the creative aspects of design as it is here that new thoughts are born, existing ideas refined, and old ideas retired as needed. We note also that this process distinguishes strong designers from weak designers: the strong designer is capable of generating and understanding ideas much more quickly than a weaker colleague.

The most powerful external influence on a designer's thoughts, however, is a project's goals. If these goals are well-formed, concise, and intuitive, a designer is more likely to be able to make useful intellectual progress towards a design solution.

4.2 Reflective Conversation with Materials

A designer can rarely work out an entire design in their mind and then simply commit it to paper and call their job done. The complexity, scale, and unpredictability involved are all factors that may require a designer to record their ideas along the way. The act of recording helps the designer to internalize understanding [6] and creates a record for later consideration. This, in turn, allows them to move on to other ideas and lines of thinking, without trying to keep the entire problem and solution in their head [24].

But beyond acting as a simple repository of ideas, representations can help to generate new ideas. This facet draws its name from the work of Donald Schön, who described the way that designers can create a representation, observe it, and gain a new understanding of the design problem and solution [21]. For example, a graphic designer might create a sketch of a logo and evaluate what aspects of it "work". Based on such observation, a designer's ideas about the design problem or solution may shift, and an updated representation might be created. Schön calls this interplay a reflective conversation with materials; its place in the DESIGN DIAMOND is shown in Figure 3b.

The choice of design notation is critical to this process, in two different ways. First, a design notation must be simple and fast enough to use, so a designer can enter a state of flow, as described by Csikszentmihalyi [4]. Schön described this ideal design state as one of reflection in action, when the creation of a design and thought about it are intertwined.

While remaining easy to use, a design notation should also establish a connection to its Domain of Materials, when possible. Such a connection allows the designer to create a representation, imagine the final product that it suggests, and adjust the representation accordingly. Establishing this connection while maintaining ease of use creates a difficult tension, but is important for a design field to address. Furthermore, because the needs of a designer will change over the course of a design process, it is important that a field reach beyond a single design notation and provide an appropriate variety of notations that can be used as needed.

4.3 Team Communication

While the previous two facets dealt with a single person's journey towards a design solution, design is often a collaborative activity. The Team Communication facet, shown in Figure 3c, focuses on the sharing of ideas among designers. This sharing does not occur directly, but through intermediate representations such as spoken words, sketches, prototypes, and full-size models. Considerations for choosing a means of communication are similar to those presented in the Reflective Conversation with Materials facet. A representation must effectively reflect the ideas that the designer has in mind and it must be easy enough to create so that the ideas can be intuitively expressed. Of course, the recipient also needs to be able to understand the ideas being represented effectively, and without undue frustration.

The difficulty of achieving this understanding largely depends on the knowledge that is shared by the communicating designers. If two designers have no common background, no shared knowledge of the product, and do not know each other personally, a great

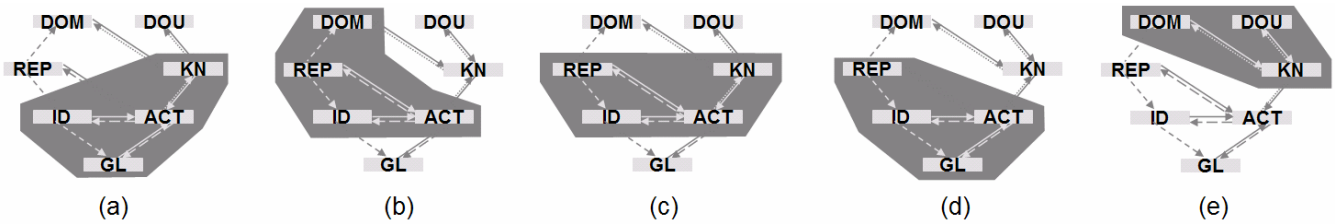

Figure 3. The Design Diamond's Facets.

deal of detail must be specified in a representation if an idea is to be correctly conveyed. But if there is extensive shared background between them, details can be abstracted away, correct assumptions made, and the design ideas much more easily communicated. The effect of such shared knowledge can be profound. For example, an email reading "ObjectManger is now a god class" may suffice in situations where a shared understanding exists, but otherwise a full and carefully-explained class diagram might be needed.

Finally, we note that the activity that guides the use of a representation can strongly affect its usefulness in communicating ideas. For example, if the creator of a diagram is present when it is being used, a discussion can serve to clarify ambiguities. Or if a representation is being created by several people at once, their shared familiarity with the intermediate forms they create can help them to understand it when it is passed around in an evolved form later.

As with the Reflective Conversation with Materials facet, the Team Communication facet introduces a tradeoff. When communicating, one must consider the recipient of those ideas as well as the nature of the ideas themselves. An appropriate representation, and thereby an appropriate design notation in which to express it, should be selected from a well-developed suite of alternatives.

4.4 Activity Type

A different way of examining design is provided by Jones, who describes design in terms of divergence, transformation, and convergence [10]. *Divergent* activities revolve around gaining an understanding of the problem, generating ideas, and exploring the solution space. Once the solution space has been explored, *transformative* design seeks to understand it. This may involve synthesizing basic ideas into higher-level principles about the design at hand. Finally, *convergence* focuses on using this understanding to make actual decisions, choose between alternatives and generally hone in on a final design. It should be noted that the ordering of these categories is not absolute. A design process might, and most often will, consist of many iterations, some with "mini" iterations within. Or the process might simply flit capriciously between divergent, transformative, and convergent tasks.

Shown in Figure 3d, the Activity Type facet illustrates the changing roles of ideas, representation, and activity as a design process progresses. During divergent tasks a creative flow of ideas is essential. Easily-created representations are desirable, even if they are not tightly bound to the final product. And when one is communicating with others during a divergent task, every detail of an idea need not be conveyed, as long as the kernel of the idea is present. Brainstorming activities, for example, place an emphasis on lightweight communication with others [11].

Transformative tasks involve bringing order to the design project, finding patterns and breaking design problems into sub-problems. This may involve the use of matrices of design considerations, the writing of specifications, or simply conversations about the design problem. In these cases, one must carefully select the design notation to be used such that the appropriate level of abstraction can be achieved and pertinent elements of the design can be emphasized. Through such approaches, the patterns that a designer is seeking can more easily emerge.

Finally, convergent tasks involve the resolution of final design decisions, helping to hone in on an optimal (or at least desired) solution. Sometimes the effects of the design decisions are very tightly interwoven, such that their combined result can be difficult to predict. In these cases, representations with a high degree of fidelity to the final product can be useful for understanding which solutions are effective, even at the cost of ease of creation. Formal proofs, systematic testing, and checklists represent examples of useful convergent tools [10].

4.5 Community

The final facet, Community, is shown in Figure 3e and represents processes that happen outside of an individual design process, but which are vital to the success of a designer nonetheless. Virtually every design process involves some element of uniqueness, but there is also a current of recurring dilemmas, common situations, and general wisdom which can help a designer in their everyday tasks. As discussed in Section 3.1, this wisdom resides in the Domain of Materials and the Domain of Use. With both domains, the challenge lies in the discovery, accumulation, and dissemination of information, which can be arranged on a scale of specificity. Very specific insight may be extremely helpful, but only in limited situations. Moreover, a sea of detailed nuggets of insight can be very difficult to navigate; even if useful advice exists, it may be impossible to find. On the other hand, broadly applicable rules can be widely used, but risk being too general to be truly useful, and are subject to exceptions or oversimplifications.

An entire paper could be dedicated to the nuances of growing community knowledge, but from the designer's perspective, it is important that the information that they need can readily be discovered and accessed. Moreover, they should have the opportunity to contribute their own knowledge to that of the community, so to not just be a beneficiary but a contributor as well.

4.6 Summary

Examining the five facets side-by-side, it becomes clear that there is tremendous pressure on the approaches and technologies that one uses to design. Representations and activities fulfill multiple, roles that vary widely, and sometimes conflict. Ideas are crucial, but are dependent on an intricate interplay of goals, knowledge, activities, and a reflective conversation with materials. The Domain of Materials and Domain of Use can be of tremendous value, but are difficult to construct. Other such tensions and forces abound. It is no surprise, then, that software design is such a challenging venture.

This does not mean, however, that we cannot make progress as a discipline. Quite the contrary, it is our belief the DESIGN DIAMOND helps us identify where we have done well, where we have fallen short, and how to approach the way forward. We do so in the next sections, first using the DESIGN DIAMOND to position current approaches to software design and then using it to highlight promising avenues of research that address some of our pertinent needs.

5. POSITIONING SOFTWARE DESIGN

One of the reasons that our field has so much difficulty discussing software design is a lack of consensus about what the phrase "design" means in a software context. Different tools, lifecycles, and philosophies imply different "design activities", different "design products" and different roles for design-oriented thought. But without a common framework, it is difficult to clearly contextualize and articulate these difficulties. In this section, we illustrate

how the DESIGN DIAMOND helps do so. Particularly, we will examine several leading perspectives on how software design fits in the overall lifecycle. This is by no means intended as an exhaustive examination of the field, or as a way of supplanting existing design perspectives. Rather, we want to show how the DESIGN DIAMOND provides a context in which different perspectives can be considered. This will give us the means to point out each perspective's focus and clarify the ways in which they agree and differ.

5.1 Waterfall Model

The Waterfall model presents a good starting point for our discussion as it directly prescribes a "design phase", and regards this design phase as containing most, if not all, of the process's design thinking. The design problem is articulated in a detailed requirements document, and the result of the design process is expected to be a plan for implementation. Purists, indeed, think of the requirements as a complete, abstract specification of "what is needed", push all consideration of "how can it be realized" into the design phase, and consider the rest "the rote task of programming out the design".

From the perspective of the DESIGN DIAMOND, the Waterfall model primarily addresses the "captures" relationships. Its recommendations serve to ensure that the large cognitive distance between the expression of a goal as a requirements document, and the realization of the goal in some set of programming languages and tools from the Domain of Materials, is bridged. One of its strengths is that it advocates using a separate design notation to express design ideas. Because it is a high-level approach, the Waterfall model does not specify many detailed design activities to be performed, but it does draw attention to the need for verification of each step along the way, explicitly calling out the role of the "captures" relationships of the DESIGN DIAMOND.

Paradoxically, the Waterfall model both recognizes and disregards the role of the more creative aspects of design. On one hand, the original concept of the waterfall model strove to reserve a phase for creatively working out a solution to a software problem, unburdened by the constraints of information gathering and implementation details. But in several ways, the waterfall model's positioning of design impedes creativity. First, a major weakness lies in the way that goals are presented. In order for the Pure Thought facet to be successful, the goal must be something that a designer can keep in mind, on a subconscious level, while making design decisions. A requirements document is often too complex from this perspective, requiring a great deal of scrutiny to understand.

A closely-related problem is that the basic, fundamental goals of a project are settled upon before designers are ever involved. The reality is that countless functionality-level design decisions reside in a requirements document. This is only exacerbated by the well-known problem of changing requirements. Because designers are not in touch with the actual needs of the user, and only receive an abstracted piece of documentation, subtle shifts can have extremely unpredictable consequences.

5.2 Agile Methodologies

Agile methodologies' perspective on design can best be summed up by the phrase "the design is in the code" [3]. Such approaches advocate the rapid creation and evolution of code, treating it not as just the material in which the final program is implemented, but also as the representation in which programmers (each of whom is also a designer) express their ideas. The main advantage of this perspective is that the gap between the chosen design notation and the Domain of Materials disappears, since the two are equal by definition. This code-centric approach is complemented by a recommendation to refactor early and often, continuously examining the structure of the code and improving it as needed. In addition, most agile efforts tend to be guided by some additional, informal design representations such as architectural sketches, manifestos, and e-mail discussions regarding critical design considerations.

From the perspective of the DESIGN DIAMOND, the *message* of the Agile methodologies resonates well with the design theories we have discussed. It is encouraged to "play with the code structure". Goals are meant to be broad understandings of user needs, not long diatribes, thereby lending freedom to the creative designer. Incremental releases fuel frequent designer-user contact to allow a goal to grow in the mind of a designer. And, finally, pair programming helps to share ideas among developers.

In *practice*, however, agile approaches do not pan out as well. As code is written, it tends to become much more rigid than intended. Although refactorings are frequently employed, major overhauls are rare, stifling the creative aspects of the design process. Even during early exploration, when little code has been written, code remains a terrible design medium as compared to sketches on a whiteboard. When coding, designers must adhere to the structural considerations of a programming language, rather than being free to express design ideas intuitively. Even though compilation and execution of code allows for feedback, the time taken to compile a program, see the results, and adjust the code is sufficient to hinder creative thought, and to prevent Schön's reflection in action. Finally, code is encumbered with so much additional detail that it is ill-suited to supporting the communication of high-level design decisions; they simply disappear.

In general, unless several favorable conditions are met (for example, a highly integrated team of expert coders, a great deal of shared understanding, and a well-understood goal), Agile methodologies pose some serious problems for design. That is not to say the methodologies are altogether "bad". Rather, it demonstrates that Agile methodologies, as the Waterfall approach, only partially meet the needs of a software designer.

5.3 Other Lifecycle Approaches

Space concerns prevent detailed analyses of additional life cycle approaches, but we do wish to highlight some interesting points with respect to a few of them.

First, the Structured Analysis and Design method [5] provides an example of an interesting tradeoff in its choice of design notation. For application domains in which information flow is a primary concern, its Data Flow Diagrams are a medium that easily spans from requirements to design to database-based implementations. The Domain of Use, i.e., information-oriented applications, and the Domain of Materials, i.e., databases, can be seen as providing uniform guidance over the course of the project.

Open Source approaches provide a different lesson, one related to the Community facet. Archives such as code repositories, mailing lists, and bug trackers are viewed as a critical part of the learning process in the open source community [9]. These sources of

knowledge are useful to those new to a project (influencing the Team Communication facet), but also to those who wish to study design structures that are used in other projects (thus forming a Domain of Use). This approach to spreading knowledge is vital to addressing some of the challenges inherent to open source.

Finally, the Rational Unified Process is a highly-modified version of the Waterfall approach in which best practice recommendations such as "visually model software", "use component-based architecture", and "develop software iteratively" complement the overall process [12]. When we consider these particular pieces of advice in terms of the DESIGN DIAMOND, we note that they emphasize easily understood and easily manipulated design representations. Even though they may not have been explicitly devised as such, they improve the practice of design in the RUP.

5.4 Summary

High-level approaches and lifecycle models are intended to structure the overall process of software development, and are not necessarily meant to provide specific design guidance. But their designation of a general role for design tasks nevertheless exerts a strong influence on software engineers' ability to design, as well as the tools needed to do so.

For example, the large cognitive gap between a project's goal and its implementation is a major design challenge. In response, the Waterfall model advocates the use of specialized design notations to be used in creating intermediate representations. If Structured Analysis and Design is employed, chances are that databases are required as a subset from the Domain of Materials, so that the transition from design to implementation is fluid. And if an iterative or Spiral model is followed, we must find an incremental representation that can easily reach back to requirements and forward to implementation; traceability is of pertinent concern. Each of these strategies of reducing the cognitive gap is quite different, but also quite natural when one understands the philosophy behind each respective lifecycle approach.

As another example, each lifecycle approach must respect the need to eventually hone in on a single design solution over the course of a project. Positioning design as an intermediate representation requires the use of validation to ensure that the representation adheres to the stated goals. Positioning design as being in the code requires refactoring to change the current design to one that is better. And positioning design within an overall iterative lifecycle requires a strong core design that is enhanced at each iteration.

We can now see how different approaches to software engineering must tackle many of the same problems. Previously their differing takes on design have prevented us from comparing their seemingly disparate solutions, but by disentangling the notion of design, the DESIGN DIAMOND helps to shed light on these and other challenges.

6. THE WAY FORWARD

Now that we have explored some of the ways that software design can be positioned, we turn our attention to how to use the DESIGN DIAMOND to help advance the field. We will do so in terms of the facets described in Section 4. For each facet, we explain to what degree the needs of software design are met by current tools and approaches, and what weaknesses remain. Based on this evaluation, we will suggest several research directions that will stand to improve software design.

6.1 Pure Thought

This is the most difficult facet for a community to support, since the actual ideas that a designer generates from their activities cannot be affected directly. But what activities the designer undertakes and how they generate and evaluate ideas are influenced by their internal knowledge and the goals they set out to achieve.

With respect to knowledge, we observe that effective designers start with a vague concept of their design, and refine it as part of their everyday design activities. But how does a designer know which refinements to make, and therefore how to proceed with improving the design? In most cases, precise metrics and formal proofs are not the answer, but rather an internal sense of aesthetics. This sense of aesthetics guides a designer on a subconscious level, causing them to be drawn to appealing design refinements, and to reject those that "seem wrong".

This sense of aesthetics is vital in fields such as architecture and fashion design, where students and professionals alike engage in the frequent viewing, discussion, and appreciation of design examples in order to develop their own sense of design. Software engineering has failed to embrace this way of thinking to date. We do know that a designer concerned with software performance values a design in a way very different from a designer who is concerned with scalability, but there are not mechanisms in place to understand, discuss, and develop these senses of aesthetics. To some extent the responsibility for improving this kind of knowledge lies with designers themselves, though we will make some suggestions on this matter when we discuss the Community facet.

Suggestion 1: We must pursue the development of a sense of software aesthetics. As with other fields, this sense of aesthetics must allow for different views and criteria, and must serve to create alternative "schools of software design appreciation".

With respect to goals and the Pure Thought facet, two points are important. First, a tome of requirements does not stimulate ideas; on the contrary, it tends to severely restrict them and interrupt the flow of creative thought because one must continuously verify if the requirements are still met. We know of no other field that goes in as much detail as software when it comes to requirements; it is customary to provide broad guidelines rather than detailed scenarios. Perhaps this is a sign that we have gone too far: should our designers not be given the freedom to bring to bear their full abilities in truly and creatively producing design solutions, rather than searching for matches to a mountain of requirements?

Should one answer a resounding "no" to this question, our second point brings to bear a slightly different argument. Specifically, we believe that if one insists on detailed requirements, then a significant portion of those should be treated as design decisions. We contend that it is preferable that a requirements document acknowledge explicitly that it has entered the realm of design decisions, rather than pretend to remain an absolute statement of needs. This point has significant depth: not every design decision is immediately recognized as such, and we must take great care to nonetheless deliver them to the designer as design choices.

Suggestion 2: It is critical to re-examine the traditional boundary between requirements and design, and to re-examine our mechanisms of specifying requirements. We must separate actual needs, broad guidelines and visions, and design decisions already made.

6.2 Reflective Conversation with Materials

At its core, the Reflective Conversation with Materials facet states that a representation must allow the designer to rapidly engage with their design in progress and receive useful feedback on the design's current state. The subtle nature of the second half of this point is perhaps best explained using an example: when a building architect first sketches out their design and looks at it, they evaluate it in terms of the final envisioned product: "is it too tall?", "is it elegant?", "does it look like it will be structurally sound?".

But it is much more difficult to look at, for example, a UML diagram, and consider it in terms of its final software product. Only an expert designer may be able to answer such questions as "is the performance of this software as expected?", "how does the software scale?", and "will it automate the company as it is meant to automate?" simply by looking at a high-level diagram. Instead, we usually evaluate the representation on its own, abstract terms, asking questions such as "are all my classes connected?", "is there good cohesion and not too much coupling?", "do I have an appropriate inheritance hierarchy?".

Software architecture provides one possible avenue for improvement. Its original vision [19] called for it to consist of elements, form, and rationale, which together begin to support the needs of reflective conversation nicely. In particular, a developed sense of form would allow a sense of aesthetics about architectural diagrams to emerge, which would help guide designers in their work. The way this vision has been realized by the community, however, has been through detailed, focusing on limited, structure-oriented languages that are geared towards formal analyzability and completeness. The designer, unfortunately, was largely ignored. Other design notations display a similar trend, tying themselves tightly to implementation at the cost of helpfulness to the designer.

Suggestion 3: We must pursue the creation of new design notations that provide feedback at the level of the software we are designing. That is, we must create design notations, not programming language abstractions.

This issue also arises in our automated design tools. On one hand, such tools are indispensable, because they provide the necessary analyses, simulations, and other mechanisms of feedback that we must use in lieu of intuitive interpretation. On the other hand, the interaction mechanisms promoted by the tools strongly focus on achieving qualities such as completeness, consistency, and precision, which do little to guide the designer. Newer design tools such as Argo/UML [20] and some software design sketching tools [8] do better in beginning to support the design experience, but we need to push further.

Suggestion 4: Our design tools must change from relatively passive tools that focus on helping a designer to precisely document a design once it has been thought out, to tools that actively help a designer think through and explore a design problem and solution; that is, tools that really help a designer design.

6.3 Team Communication

When we introduced this facet in Section 4.3, we identified three main criteria for the successful communication of design ideas in a design team: an appropriate activity structure, an effective representation, and some degree of shared understanding.

With respect to activity, designers working in a group have a natural tendency to adopt group-oriented behavior. They congregate in meeting rooms, more explicitly state their assumptions, use drawings and whiteboards to explain their point of view, etc. We also see group-oriented behavior in the open source community, where key architects of certain systems come together periodically to assess the state of affairs [15]. And when one takes the Agile view of design, pair programming is a form of design communication. Short of forcing designers to communicate (which pair programming does in a relatively friendly way), one can suggest certain behaviors of social interaction, such as letting everybody talk at a meeting or engaging in periodic reviews and brainstorming. But this requires knowing what works and what does not, something that our field does not currently possess.

Suggestion 5: We must engage in detailed empirical and analytical studies of software design teams in action to begin understand their needs, behaviors, and patterns of success and failure.

In parallel, we can improve our design notations that team members will use to communicate with one another. This is particularly important when we consider that much design takes place in meeting rooms. Considering the technology that is available, such as electronic whiteboards, tablet PCs, and a wide variety of input devices, it is now possible to display and edit software architectures, pattern compositions, informal sketches, and the like directly in the meeting room.

However, most of our current design tools are not suited towards creative group use, instead focusing on individuals documenting a design. Most group-oriented tools, such as those that track design rationale or manage reuse are more focused on communicating finalized decisions than actually supporting a group as they design. We need tools that allow for quick exploration and expression of high-level ideas, articulation of decisions and assumptions, analysis of alternatives, and simulation of designs in action.

Suggestion 6: We have to explore the development of design tools and associated design notations that are specifically geared towards interactive group use in early exploratory phases of design.

Finally, we must recognize the need for a shared understanding between communication participants. Architectural styles [22] and software design patterns [7] are powerful advances in this regard, providing a higher-level language in which designers can frame their problems and solutions. By alluding to a well-known style or pattern, a software designer can convey concepts that would take pages to describe in a rigorous document. The challenge lies in developing more of such useful concepts and disseminating them so that they become widely understood, a concept we will discuss further in Section 6.5.

6.4 Activity Type

The Activity Type facet describes design in terms of Jones' categorization of divergent, transformative, and convergent activities. We first note that, traditionally, software designers do not explicitly view or explore software design in terms of these activities. It may be that their actions exhibit some resemblance, but we have not yet turned to recognizing and utilizing this approach explicitly.

First and foremost, a radical thought is to consider these steps at the process level: what if we treated the entire software development process as a divergent, transformative, and convergent series of activities? Requirements and architecture would serve as a

divergent stage of research and exploration, lower-level design as a transformative stage, and code as a stage of converging upon a single result. This unusual perspective has far-reaching consequences. It certainly would require serious reconsideration of the boundaries and approaches set forth by the traditional Waterfall model. The foundations of Agile approaches would also be challenged, as code alone would be insufficient for meeting the needs of all three activity types. Many of our design notations would become irrelevant. But such an approach would remain true to the DESIGN DIAMOND, giving us reason to believe that this may be a plausible form of alternative lifecycle approach.

Suggestion 7: We should explore the entire software development process as a design process, with the goal of understanding the feasibility, realities, and consequences of adopting a lifecycle approach based on divergence, transformation, and convergence.

Beyond such ambition, however, we can also look at the consequences of adopting this view in the small, questioning, for example, whether current software design tools and notations support easy brainstorming (identified by Kelley as a cornerstone of idea generation [11]) and whether we have the tools available to transform these disconnected thoughts, establish evaluation criteria, and settle upon a broad solution structure. In general, the answer is "no", though there are various existing tools and technologies that show promise. Metrics can help to judge design structures, while early aspects and multi-dimensional separation of concern can serve to untangle representations. Hypermedia and other techniques can also be used to track design changes.

The problem is that these approaches have not been presented in a way that is conducive to the needs of a designer. Tools supporting divergent thought must be intuitive and easy to use, while transformative thought requires carefully chosen abstractions that can draw the designer's focus to certain aspects of the design. Convergence cannot take place simply in terms of metrics and numbers, but must illustrate the differences and tradeoffs implied by multiple possible designs. It is our hope that considering the challenges of software design in terms of this general theory will inspire new tools and technologies for software designers.

Suggestion 8: We should investigate when and where a divergence-transformation-convergence approach would be compatible with current approaches and devise tools, evaluation criteria, and design notations that take advantage of these opportunities.

6.5 Community

As discussed in Section 4.5, the Community facet involves two essential forces: the build-up of the Domain of Materials and Domain of Use and the dissemination of their collective wisdom to individual designers.

With respect to the Domain of Materials, its primary constituents are the programming languages and tools that are at our disposal to implement a design. This includes programming and scripting languages, databases, spreadsheets, generators, dynamic linkers, network protocols and associated libraries, and so on. But it is not just their availability that is important, it is also basic knowledge about them, such as portability, speed, robustness, speed of programming and adoption rate. Each of these facts has the ability to influence the designer, presenting opportunities and imposing constraints on the kinds of designs that are feasible.

The Domain of Materials for software design continues to evolve. It is interesting to observe, however, that its evolution is driven by language features for programming, not by a desire to improve our ability to design. The result is that, from the perspective of design, the Domain of Materials has not significantly improved.

Suggestion 9: We, as the software engineering community, need to become strongly involved in the definition of new programming languages and other materials so they better suit our needs as a design discipline.

Our Domain of Use is relatively well established, with knowledge of different architectural styles, design patterns, established sets of design metrics, detailed studies of how users react to and engage with different user interface paradigms, standards, and other sorts of design knowledge and amassed experience. There is still a strong need to build up our Domain of Use (for instance with the sense of aesthetics mentioned in Section 6.1), as it still lacks the quality and depth found in other disciplines. But a larger issue looms. One of the strengths of other disciplines is that their Domain of Use consists of many different subdomains, each addressing a small set of targeted designs with specialized knowledge. Software engineering has resisted doing so. We recognize the difference between designing flight software or games, but our research and literature do not yet reflect this recognition. Calls for domain-specific approaches have been made before, we add ours.

Suggestion 10: We must actively pursue the construction of a carefully-partitioned Domain of Use that provides greater guidance in each of its subdomains.

Finally, we note that much of the progress that we suggest here requires the dissemination of gathered wisdom to designers themselves, a challenge the community facet is meant to address. Much of this dissemination is handled through conferences, journals and seminars, but education plays a unique role in this regard, shaping the initial attitudes of future software designers. Unfortunately, current approaches to teaching software design are severely lacking. Students are often only exposed to design ideas as part of a generic software engineering course, and even then the focus is on notations, rather than design skills themselves. Students do not receive the exposure to all of the necessary theory, nor are they able to practice sufficiently.

Suggestion 11: We must elevate software design education to be a primary concern in curricula and find ways of effectively delivering the theoretical aspects of design and fostering appropriate practice.

6.6 Summary

In this section, we have provided a number of potential research directions that we believe will improve our ability to design software. Each of the suggestions finds its roots in the philosophy of the DESIGN DIAMOND, keeping in mind design's fundamental dimensions of concern, interrelationships, and theories.

Altogether, our suggestions are at varied levels of detail, and involve a range of activities and objectives. Together, they push us towards a vision of design in which each of the fundamental forces and tensions is addressed, or at the least understood so they can be reckoned with.

Clearly, some of our suggestions have been made before (consider Vincenti's "normal design" [26] versus our "wisdom", or Parnas' early design observations [18] versus our call for design notations instead of programming language abstractions). We consider this

an inherent strength of the DESIGN DIAMOND, since it confirms such existing intuitions by situating them in a generic framework that provides a design-oriented, explanation of their foresight.

Finally, the careful reader will note that quite a few of our suggestions seem to focus on what could be misconstrued as the "softer, non-engineering side of design". We have two responses. First, to date our field indeed has made its contributions from a more technical and analysis-focused perspective that is befitting its lineage of engineering and mathematics. It is now time to balance this focus with the more creative aspects of design. Second, addressing the theories of Pure Thought, Reflective Conversation with Materials, and Activity Type by no means implies an abandoning of technical and analysis aspect. Quite the opposite: without proper technical and analysis support, the creative exploration of software solutions would be a nearly hopeless endeavor. We therefore remain strong proponents of an integrated approach that, indeed, addresses all aspects of the DESIGN DIAMOND.

7. CONCLUSIONS

This paper makes two contributions to the discipline of software design. First, it contributes the definition of the DESIGN DIAMOND, a new framework that delineates software design from an interdisciplinary, designer-centric point of view. Second, it demonstrates how application of the DESIGN DIAMOND sheds new light on existing design perspectives, as well as on the field's overall research agenda.

The strength of the DESIGN DIAMOND lies in its neutrality; because it examines design from the perspective of its fundamental dimensions of concern and the different relationships that tie these concerns together, it is able to clearly articulate the elementary forces and tensions that underlie any kind of design process. A second strength is that the structure of the DESIGN DIAMOND is organized such as to capture established design theories in individual facets. These facets group elements of the DESIGN DIAMOND to illustrate the higher levels of concern expressed by the theories.

We raised a number of questions in the introduction. The DESIGN DIAMOND is a vehicle to answer these questions objectively, as have begun to do in this paper. While one still can express their preferences and biases by favoring some parts of the DESIGN DIAMOND over others, the fact that the DESIGN DIAMOND exists forces one to acknowledge the and make explicit these biases. Overall, our hope is then that the DESIGN DIAMOND can become a mainstay of the field and help channel discussions, debates, and future research into productive and innovative directions.

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 broad 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, and other 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 preferred to engage. The DESIGN DIAMOND is our first such step, and there are many left to be taken.

8. ACKNOWLEDGMENTS

Effort partially funded by the National Science Foundation under grant numbers CCR-0093489, DUE-0536203, and IIS-0205724.

9. REFERENCES

[1] Albrecht, D., Lupton, E. and Holt, S. *Design Culture Now*. Princeton Architectural Press, New York, 2000.

[2] Alexander, C. *The Timeless Way of Building*. Oxford University Press, New York, 1979.

[3] Beck, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, MA, 1999.

[4] Csikszentmihalyi, M. *Flow: The Psychology of Optimal Experience*. Harper Perennial, New York, New York, 1991.

[5] Demarco, T. and Plauger, P.J. *Structured Analysis and System Specification*. Prentice Hall, 1979.

[6] Eastman, C.M. New Directions in Design Cognition: Studies of Representation and Recall. in *Design Knowing and Learning: Cognition in Design Education*, Elsevier Science, Amsterdam, 2000.

[7] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison Wesley Professional, Reading, MA, 1994.

[8] Hammond, T. and Davis, R., Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams. in *AAAI Spring Symposium on Sketch Understanding*, (2002).

[9] Hemetsberger, A. and Reinhardt, C., Sharing and Creating Knowledge in Open Source Communities: The Case of KDE. in *The Fifth International Conference on Organizational Knowledge, Learning and Capabilities*, (2004).

[10] Jones, J.C. *Design Methods*. John Wiley and Sons, Inc, New York, 1970.

[11] Kelley, T. *The Art of Innovation*. Doubleday, New York, 2001.

[12] Kruchten, P. *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, Reading, MA, 2000.

[13] Lidwell, W., Holden, K. and Butler, J. *Universal Principles of Design*. Rockport Publishers, 2003.

[14] Matchett, E. Control of Thoughts in Creative Work. *The Chartered Mechanical Engineer*, *14* (4).

[15] Mockus, A., Fielding, R.T. and Herbsleb, J.D., A case study of open source software development: The apache server. in *International Conference on Software Engineering*, (2000).

[16] Norman, D.A. *The Design of Everyday Things*. Basic Books, 2002.

[17] Page, J.K. Conference Report *Building for People*, Ministry of Public Building and Works, London, 1965.

[18] Parnas, D. *Software Fundamentals: Collected Papers by David L. Parnas*. Addison-Wesley Professional, Reading, MA, 2001.

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

[20] Robbins, J., Hilbert, D. and Redmiles, D., Argo: A Design Environment for Evolving Software Architectures. in *Nineteenth International Conference on Software Engineering*, (Boston, MA, 1997), ACM Press.

[21] Schön, D.A. *The Reflective Practitioner*. Basic Books, 1982.

[22] Shaw, M. and Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline* Prentice Hall, 1996.

[23] Shneiderman, B. *Designing the User Interface*. Addison Wesley, Reading, MA, 1997.
[24] Suwa, M. and Tversky, B., External Representations Contribute to the Dynamic Construction of Ideas. in *Diagrammatic Representation and Inference: Second International Conference*, (2002), Springer
[25] Tufte, E.R. *The Visual Display of Quantitative Information.* Graphics Press, 2001.
[26] Vincenti, W.G. *What Engineers Know and How They Know It*. John Hopkins, 1990.