Notation and Representation in Collaborative Object-Oriented Design: An Observational Study

Uri Dekel and James D. Herbsleb
School of Computer Science, Carnegie Mellon University
{udekel,jdh}@cs.cmu.edu

Abstract

Software designers in the object-oriented paradigm can make use of modeling tools and standard notations such as UML. Nevertheless, casual observations from collocated design collaborations suggest that teams tend to use physical mediums to sketch a plethora of informal diagrams in varied representations that often diverge from UML. To better understand such collaborations and support them with tools, we need to understand the origins, roles, uses, and implications of these alternate representations. To this end, we conducted observational studies of collaborative design exercises, in which we focused on representation use.

Our primary finding is that teams intentionally improvise representations and organize design information in response to ad-hoc needs, which arise from the evolution of the design, and which are difficult to meet with fixed standard notations. This behavior incurs orientation and grounding difficulties for which teams compensate by relying on memory, other communication mediums, and contextual cues. Without this additional, information the artifacts are difficult to interpret and have limited documentation potential. Collaborative design tools and processes should therefore focus on preserving contextual information while permitting unconstrained mixing and improvising of notations.

Categories and Subject Descriptors D.2.10 [*Software Engineering*]: Design - Methodologies and Representations; H.5.3 [*Information Interfaces and Presentation*]: Group and Organization Interfaces Computer Supported Cooperative Work

General Terms Design, Documentation, Human Factors

Keywords Representation, Notation, Modeling, Collaborative Software Design, UML, OOD

OOPSLA'07, October 21–25, 2007, Montréal, Québec, Canada.

1. Introduction

Software design is a highly visual activity where diagrams are used for brainstorming, grounding, and communicating ideas and decisions [6]. This is particularly true for the object-oriented (OO) paradigm, which involves large numbers of entities and complex relations between them. Various notations have been proposed for expressing designs in this paradigm, and one of them, the *Unified Modeling Language* (UML) [23], has become a widely recognized standard. Many software packages and CASE tools enable designers to create UML models for documentation and implementation purposes.

The design of complex software systems, and in particular *object-oriented design* (OOD), is often also a highly collaborative activity, where ideas are presented, discussed, and refined by multiple stakeholders. Such interactions often take place in face-to-face design meetings, in which drawing activities also play a central role [6]. Recent efforts make UML modeling functionality available to teams using electronic whiteboards [5,8,15], and some also offer support for the systematic capture of decisions and rationale [3].

However, casual observation suggests that OOD teams do not fully utilize these standards, techniques, and technologies. Instead, they tend to follow an erratic process in which they generate a plethora of diagrams and fragments over physical mediums such as whiteboards or paper. These visual artifacts are not only less aesthetic and complete than those created electronically, but most importantly, also frequently diverge in their notation from UML, in some cases substantially.

It is easy to dismiss this everyday phenomenon as a manifestation of a "developer mentality" of cutting corners or of the need for agile design methods [1]. However, it also highlights several issues that have much relevance for the larger community of OO practitioners and researchers but that have received little attention. First, while we have powerful notations, formalisms, and tools for expressing finalized designs, we know surprisingly little about the processes and representations by which teams create these designs in the first place and whether the same tools are applicable in these early stages. Second, while we know that designers of-

ten sketch in alternate notations, the origins, semantics, and roles of these notations remain unclear. Third, we know little about the impact of sketches created in these notations, the connections between them, and whether or not they can be straightforwardly used for documentation or implementation or transformed into compliance with standard formalisms.

These issues have many significant implications. For example, it is not clear that UML, which is primarily a specification and documentation standard, can also effectively be used as a design notation in early phases and in collaborative settings. Should efforts be invested in modifying UML or in coming up with a different notation? Can UML-based design tools be augmented to effectively support early collaborative design?

Despite the abundance of literature on sketching and collaboration in general, only a few works [5, 8, 9, 28, 29] address the unique settings of collaborative OOD, and these primarily focus on the eventual automatic transformation of sketches into complete and aesthetic implementation- and archival-quality UML models. They appear to treat the creation of these finalized models as the primary motivation behind drawing activities, and tend to consider the early sketches as premature or peripheral artifacts that will evolve in due time; digressions from from UML are treated as accidental and are deprecated.

While support for transforming and completing these models into a standard notation is likely valuable for use in later development stages [6], we suspect that this approach does not "tell the entire story" about OOD representations, nor does it support all the unique needs of design teams. Rather, there may be important underlying reasons behind the representational choices teams make which have significant impact on how teams work in the OO paradigm and on how practices and tools can support them.

1.1 About this work

In this paper we seek answers to several questions: First, what kinds of representations are constructed by design teams as the design evolves, and how do they diverge from UML? Second, why are such nonstandard representations constructed and what purpose do they serve? Third, how are these representations used throughout the design process? Fourth, can a better understanding of these representations suggest new practices and forms of tool support?

To answer these questions, we conducted a series of observational studies of collaborative design exercises at the OOPSLA *DesignFest* event, focused on representation and diagram use. These settings allowed us to study the representations used by groups of experienced OO practitioners working on identical problems while curbing the impact of organization-specific practices. We discuss this choice and its potential threats to the validity of our findings in Sec. 3.

Our intention in this paper is not to comprehensively catalogue or explain all the improvised notations used in collaborative OOD or to offer quantitative information about representational choices, although these are interesting avenues for subsequent work. Instead, our presentation is focused on typical cases in which alternate notations are used, since these have little to do with the completeness of the model and are perhaps the most revealing about representational needs and choices. In addition, we focus on several related issues that have significant implications for supporting collaborative OOD, but that have received limited attention: How heterogeneous information is represented, the characteristics of designs that spread over multiple diagrams, and how teams cope with this delocalization.

1.2 Findings and contribution

Our observations suggest that in choosing representations, teams make deliberate and intentional choices to diverge from standard notations or borrow idioms from other notations. While often benefiting from lower physical and cognitive efforts, teams make these choices primarily in response to the immediate needs arising from their evolving understanding of the problem and solution. Improvised freehand notations are not only easier or faster to use but also offer a degree of freedom in selecting structure and interpretation that may better fit and evolve with the design than a fixed notation. Similarly, information is dispersed and laid out in ways that not only increase the locality of related entities but that also facilitate a more natural representation of complex heterogeneous information.

While these representational choices allow teams to focus on creativity, the resulting artifacts and the relations between them may be less intuitive, exact, and complete. Teams try to compensate for this by using additional but transient communication mediums, such as voice and gestures, and by relying on memory and contextual cues. Other stakeholders, however, will face significant difficulties in subsequently interpreting and using the resulting diagrams as documentation or implementation artifacts. In fact, knowledge of contextual and historical information of the experiences shared by the meeting's participants may be necessary even for diagrams that teams have explicitly prepared for presentation.

The importance of these findings is in improving our understanding of collaborative design meetings and in guiding the development of new tools. One contribution of our work is in demonstrating the priority given to creative activities and that some of the noncompliant artifacts, often treated as peripheral by existing tools and approaches, actually play a major role in problem-solving and in communicating about the design. Second, we show several typical ways in which collaborative design unfolds and ad-hoc needs lead away from standard notations. A third contribution is in highlighting the importance of contextual and historical information in the work of design teams and in its manifestation into the products, thus prompting efforts to preserve it. Such efforts may be more effective than attempts to support or perfect any notational standard since teams require a flexibility that may not be possible with a fixed representation.

Outline: The rest of this paper is organized as follows: Sec. 2 reviews the literature on the use of visual artifacts in OOD and the tools for supporting it. Sec. 3 discusses our choice of settings for the study and presents the methodology. We begin presenting our observations in Sec. 4, which describes the use of alternate notations in individual artifacts. This presentation continues in Sec. 5, which describes the representation of heterogeneous information, and in Sec. 6, which discusses the dependencies between diagrams. We discuss these results in Sec. 7, and their implications for tools in Sec. 8. We conclude and present our current research directions in Sec. 9.

2. Background and related work

The design and architecture of a complex software system has significant implications for its functionality, cost, and reliability; a significant portion is therefore typically done upfront. While individual developers perform many design activities, the design of common modules, larger systems, and components is typically a highly collaborative process which involves many stakeholders. Design may be the most collaborative part of the development process [16].

Design tasks generally make heavy use of external representations to support problem-solving and collaboration and to capture the current state of the design [27]. This also applies to the domain of software design, in which diagrams go through a lifecycle from transient artifacts used to understand and come up with a design to archived documents serving communication and documentation purposes [6]. However, the choice of representation depends on many factors, including intended use, individual or collaborative design, organizational background, development paradigm, and the design problem specifics.

2.1 Representing completed designs

As with other design domains, well-defined and accepted representations and notations, rendered aesthetically and with precision, are important for clearly and unambiguously expressing finalized software designs [6, 9, 29]. In the OO paradigm, the *Unified Modeling Language* [23] has gained widespread academic and industrial acceptance as a standard representation for completed designs that serve documentation or implementation planning purposes.

UML consists of 13 diagram types, which cover many of the structural, dynamic, and functional aspects of a system. These notably include *class diagrams* (CDs), *sequence diagrams* (SDs), and *use-case diagrams* (UCDs). UML offers precise notations for accurate specifications as well as extension mechanisms. This comprehensiveness enables compliant UML models to be used as early implementation artifacts, as blueprints for implementation, or as basis for automatic code generation [8]. A common criticism of UML is that it lacks fixed semantics and conventions [4, 26], potentially leading to defects due to misinterpretations. Other critics, focused on creative modeling, argue that UML is too strict and suggested that investment in creating complete models is not cost-effective [1].

2.2 Representing early designs

While much attention has focused on the representations used for documenting completed designs, little is known about the representations used in earlier design phases to come up with initial designs and if the same notations are applicable. One obvious difference about these phases is that design teams tend to sketch, often using physical mediums such as whiteboards. Sketching allows designers to effectively focus on the problem [1] and encourages experimentation with the design [5, 21, 24].

Most of our knowledge on early diagrams in industrial settings comes from Cherubini's study of diagram use at *Microsoft* [6]. He found that while many diagrams created in design collaborations were transient, some were immediately captured for subsequent use or were captured after being recreated in later collaborations. As their importance became clear, they iteratively became more organized and aesthetically pleasing and were often captured electronically. The representations used in these diagrams were a a mix of informal visual conventions based primarily on box-and-arrow notation with only limited adoption of standards such as UML. However, the studied population used a variety of development paradigms and may not have been proficient in OOD or UML, and the factors leading to the representational choices were not studied.

The few observations that do focus on collaborative OOD work are primarily reported by researchers involved in constructing sketch-based design support tools [5, 8, 9, 28, 29] and take place in specialized settings. In these observations, the end product is typically a design documented in syntactically correct UML, often in electronic form. Initially, freehand sketches are used to represent the problem domain, while limited UML diagrams, incomplete in content and syntax, are used for early designs of the solution. Damm et al. [8, 9] further argued that these diagrams are then incrementally evolved into complete and compliant UML models with noncompliant elements removed. Overall, these studies give the impression that divergence from UML is accidental or unwanted.

Departures from UML or other standard notations might appear counterproductive when the goal is to produce an implementation-ready model or documentation. However, since representation choices have significant impact on design, constraining the early stages may have unexpected effects. Clearly, there are different potential uses and outcomes for diagrams created in design collaborations, and both organizational practices and prior knowledge of the intended use of a diagram likely have an impact on the representational choices. Since our goal is to understand the needs of early design, we have chosen to conduct our study in set-

tings which afford relative freedom from such potentially confounding constraints.

2.3 Design support tools

Most of the existing tool support for collaborative OOD is similar to that provided for individual designers, consisting mainly of functionality for creating complete models in UML. One approach aimed at distributed teams is to offer distributed groupware versions of the familiar single-user desktop CASE tools [3, 22]. In collocated collaborations, however, designers need large drawing surfaces [12], which were traditionally only available in the form of whiteboards or paper. As large display technologies became available, attempts were made to provide OOD specific support over electronic whiteboards [5, 8, 15]. These tools focused on the automated conversion of sketched shapes and handwriting, which are the natural mode of interaction with a whiteboard, into notational primitives and even UML models.

Of particular interest is Damm et al.'s *Knight* tool [8] which, based upon the observations described above, offers a guided mode that facilitates the evolution of these artifacts, through several levels of restriction, to complete UML models representing implementation-ready designs. Whenever artifacts violate UML specifications due to nonstandard notation or incompleteness, their tool forces the offending notations to be changed or removed. Other tools allow a layer of uninterpreted "freehand" sketches to coexist with a layer of structured UML.

The interactive nature of electronic whiteboards raises the question of how to best support collaborative design, and what, if any, support should be provided for rough, intermediate forms. We note that although these technologies received much attention, their availability is extremely limited, and most practitioners have no supporting tools when involved in collocated design collaborations. While some research focuses on supporting distributed design, such settings are plagued by many additional problems and are outside the scope of the current work.

3. Methods

3.1 Settings

Our studies of collaborative OOD span three incarnations of the annual *DesignFest* events of the *ACM conference on Object Oriented Programming Systems Languages and Applications* (OOPSLA). In 2004 we conducted a preliminary study focused on the physical design environment, using only photographic evidence [10]. The observations presented in this paper are primarily based on extensive data collection in the 2005 session and always refer to it unless otherwise noted. In 2006, we conducted a limited validation study to verify our conclusions.

In this popular conference event, experienced designers select one of several given design problems and are randomly assigned to one of several teams that will work on that problem. They are given a short document describing the problem, constrains on the solution, and important use cases and scenarios. Depending on the session, teams then spend 3 to 6 hours coming up with an appropriate design to solve that problem. The stated goal of this event is to "learn more about design by doing it and to sharpen design skills by working on a real problem with others in the field" [11].

One noteable characteristic of the *DesignFest* event is the relative freedom given to the teams. While teams are not expected to produce working systems, they are encouraged, both in the given documentation and verbally during the session, to prepare materials for presentation to others at the social event at the end of the conference and for a possible web archive. However, they are not given explicit requirements for the representations and quality of these materials. Similarly, the given documentation merely suggests a simple process outline, stepping from introductions to planning, discussion, sketching, and resolution of disagreements. It also encourages teams to elect a moderator and a recorder, but we have rarely seen these roles used in practice.

We also note that drawing activites at *DesignFest* take place primarily over physical mediums, which offer greater freedom than electronic tools [2]. The organizers provide all participating teams with at least one flipchart and several posterboards as well as notepads, sticky-notes, tacks, and pens. Powerstrips and wireless internet were available, but the few participants who used laptops did so mostly for unrelated activities. Only two participants, from the same organization but in different teams, tried using a CASE tool to digitize their teams' sketches.

3.2 Choice of setting and threats to validity

We chose these settings for three reasons: First, participants do not have a history of working together and are collaborating outside an organizational context that prescribes design methods, notations, and processes. We believe this simplifies the interpretation of our results and enables generalization about common practices. Had we observed teams in an organizational context with a work history, it would be hard to disentangle behavior that is merely prescribed by organizational practices or shared team habits from natural behaviors that directly support the immediate task of collaborative design.

Second, since our focus is on the OO paradigm and the use of UML in particular, we wanted to minimize the risk that limited familiarity with these techniques would factor into the choice of representations. The OOPSLA conference attracts experienced designers well versed in these techniques, which adds validity to the findings. Third, since design problems in *DesignFest* are based on real-world projects, we can observe multiple teams working on the same problem and reproduce the nonproprietary designs. Although we could probably find other settings that are better in any one of these considerations, *DesignFest* seems to

score adequately on all three, and, in our opinion, deserves the attention of more researchers.

Observational studies of software development in the literature take place in the lab, in academic settings, or in industrial ones. Research in artificial settings enables the collection of data which could be difficult to gather otherwise. The *DesignFest* settings not only allowed us to study teams of experienced OOD practitioners while restricting confounding organizational practices, but it also let us study multiple teams working on the same problem from scratch, a situation which rarely occurs in the industry. Of course, these advantages involve a tradeoff, as artificial settings raise the question of whether the results could be generalized.

Clearly, additional research in industrial settings is required to establish that the observations we made here hold more generally. All settings are unique in certain respects, and replication in multiple organizations may be necessary. Nevertheless, let us briefly mention the primary ways in which *DesignFest* differs from industrial settings and estimate their potential impact on generalization.

First, design takes place in isolation from other, overlapping, development phases; there is no prior requirement gathering, though in some sessions the *DesignFest* organizers played the role of clients and spent time with each team. More importantly, the produced designs are not subsequently implemented. However, *DesignFest* problems are based on real-world specifications, and most participants appeared passionate and serious in their work. It is thus likely that the proposed designs, or at least the initial high-level sketches, would be similar to those created by industrial teams early in the design phase.

This brings us to the second problem of the extreme time pressure compared compared to industrial settings. While more research is necessary, we believe that our data reliably captures how representations are selected when teams first face the problem, an issue for which limited prior knowledge exists. The impact of the time limits is likely to be primarily on how details are fleshed out, documented, and reviewed; these issues have been studied more extensively by others.

Third, *DesignFest* participants receive no material compensation for participation and are not held accountable for their work; the motivation is primarily to learn and interact with respected peers. In that sense, it resembles open-source settings, where nonmaterial compensation is key to contribution. This similarity also applies to the diverse background of the participants, as industrial teams typically have shared prior experiences and practices. However, we explicitly tried to minimize such organizational bias.

3.3 Collected data

In the 2005 session, on which we report here, we made video recordings of the design sessions of several teams. We chose video in order to capture as much context as possible, including a trace of all interactions, annotations, and references to artifacts. Because teams tend to post and refer to materials all around them [10], we used widescreen camcorders to capture more of the design area, one of them filming in high-definition. To enable us to evaluate the value and limitations of more available means of preserving design knowledge, we also made a separate audio recording and frequently took still photographs. These additional mediums also served as a redundancy for the material in the videos.

Due to the sensitive nature of video recordings, significant measures were taken to ensure participant consent. Prior to the conference, the *DesignFest* organizers sent preregistered participants an email describing the study and asking for preliminary consent to different recording mediums. The large number of preregistered participants allowed the organizers to form initial groups based on consent, without denying anyone their choice of design problem. Since the *DesignFest* event is also open to walk-in participants, potentially affecting group composition, we also verbally introduced the study to each of the consenting groups and obtained written consent. In addition to changing all names in this paper, we avoided presenting photos of individuals when possible, and distorted identifying features in other cases.

The real-life design problems available that year were a medical information system for a pediatric theraphy clinic, a system for a web-based image management and photo printing business, and a generic simulator for third-party controllers in industrial production lines. The first two problems are representative of typical information and eCommerce systems, while the last is an example of a more exploratory project in a different domain

Since many consenting teams were working at the same time, our selection of observed teams attempted to obtain a blend of problems. We sampled only groups that had at least four participants at the beginning of the session and that were located in areas with limited acoustic interference from other teams. Depending on the session, teams worked between 3 and 6 hours excluding breaks, and we tried to film the entire session of each team. In one case, we switched from team D to team E to capture more drawing activity. Overall, we obtained over 20 hours of video footage from seven teams, summarized in Table 1. We also obtained still photographs of the work of other teams.

Group	Problem	Session	Length	Recorded
A	Simulator	Sun PM	3 hours	All
B	Image Shop	Sun AM	3 hours	All
C	Image Shop	Sun PM	3 hours	All
D	Medical System	Sun Full	6 hours	First 2 hours
E	Medical System	Sun Full	3 hours	Last hour
F	Medical System	Wed Full	5 hours	All
G	Medical System	Wed Full	5 hours	All

Table 1. Groups for which video footage was recorded

3.4 Data analysis

Before proceeding to analyze the data, all video footage was digitally transferred to a video editing software, allowing us to effectively browse hours of footage to trace the evolution of artifacts. The tapes were then fully transcribed as LaTeX documents, allowing us to create versions limited to dialogue and versions with descriptions of additional activities.

Our analysis of each team's work typically proceeded as follows. First, we studied photographs of the finished diagrams from the end of the session and tried to understand them without additional information, as potential consumers of these materials would have to. Next, we examined all the photographs taken throughout the session and used temporal cues to obtain a better understanding of the artifacts and their evolution. Only then we read through the transcripts, and eventually watched the entire tape. Throughout this process, we made notes of relevant observations. Eventually we pooled and studied the observations from all teams, identified the examples which we present in this paper, and returned to the materials to study them in depth.

4. Results: Alternate notations in individual artifacts

Our presentation of the results begins with the representations used in the collaborative creation of individual artifacts.

As could be expected from the venue, UML was utilized by all observed teams. Specifically, all teams drew class diagrams, and several drew sequence diagrams, use-case diagrams, and even one package diagram. Perhaps due to the limitations imposed by the settings, the nine other diagram types were hardly used. In conformance with our casual observations and prior works, we frequently observed departures from UML and proceeded to investigate them.

Previous researchers [9] suggested that divergences from UML are mostly accidental and that diagrams eventually comply with the standard. When we observed teams explicitly reproducing artifacts for presentation at the conference's final event, they indeed tried to create aesthetic and complete UML models. However, most of these attempts took place towards the end of the session, after the brunt of the creative activity was done, and had a relaxed and visually distinct interaction style. Teams often split up, and subgroups worked on recreating diagrams on new canvases instead of modifying existing ones.

In the creative phases, however, we often observed situations where the departures from UML appeared to be intentional rather than accidental. Specifically, we saw artifacts which convey information that could have been expressed in a straightforward manner via UML, but were created in completely different notations or which violated significant principles of the standard. These cases are particularly interesting since they yield significant clues about the behavior and needs of design teams.

To help the reader distinguish the detailed objective descriptions of activities and artifacts from our subjective interpretation and analyses, we present the former as *italicized* text. Also, the textual descriptions frequently refer to photographs of the artifacts. Because of space constraints, we could only include small images within the text. The reader is encouraged to view the electronic version of this paper, where these photos appear in full color and at a much higher resolution than that possible in print. Using the PDF reader's zoom or magnifying glass functionality will make the handwriting in many figures legible.

4.1 Adapting to evolution

Certain UML diagram types convey multiple layers of information. Class diagrams, in particular, not only list the classes of the system (or its entities in the case of domain-modeling [1]), but also specify the members of each class, and the OO (e.g., inheritance) and data cardinality (e.g., one-to-many) connections between classes. The notations of UML allow designers to add this information in any order, while still maintaining the syntactic correctness of the diagram.

Nevertheless, we observed a number of cases where diagrams ended up with all these layers of information but expressed in an entirely different form. Teams tended to select diagram types opportunistically in order to address the issue at hand, and captured knowledge as it emerged from the problem-solving process even if it did not fit the current selection. In some cases, they began constructing a diagram of one type, only to have it morph into another. On other occasions, the diagrams turned into a collection of fragments related only by their relevance to a particular design issue. We present three such examples in detail in order to convey the process by which these changes occur.

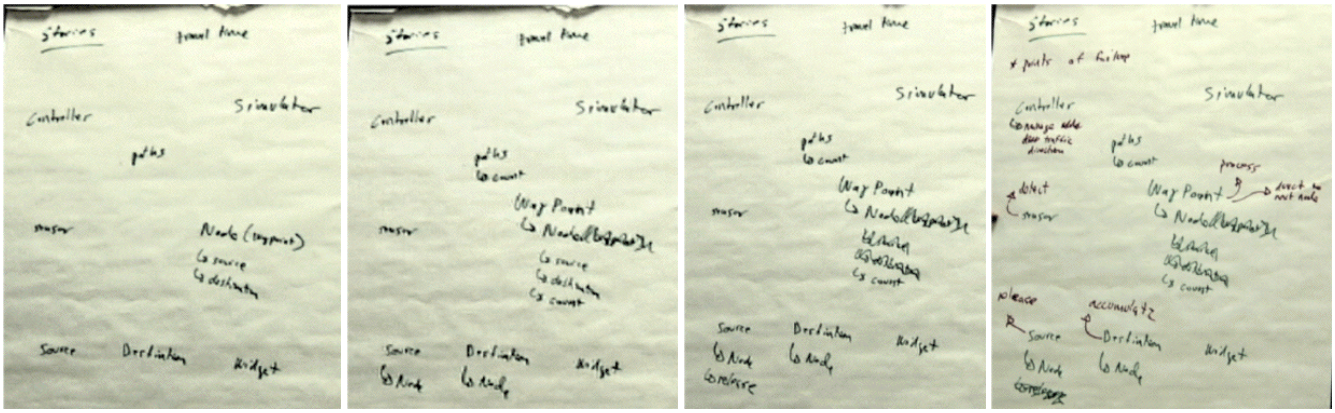
4.1.1 Example: structural domain model

Our first example, depicted in Fig. 1, comes from team A, which was working on the problem of constructing simulators for third-party controllers of production lines.

The team began its session by discussing assumptions, and then turned to exploring use-cases. Jack, standing by the flipchart, titled the blank canvas `Stories`*. Then, somebody suggested that they first cover high-level domain entities, and others agreed. The team started brainstorming ideas, which Jack scattered on the titled canvas.*

Already, we have a discrepancy that may confuse future stakeholders: based on the title one may expect to see the names of use cases, but the diagram actually contains the names of entities. On the other hand, team members will likely recall that change of direction or recognize the contents of the diagram without relying on the title.

Very soon, it turned out that the proposed entities could be related and questions were posed and discussed. Are `Node` *and* `Waypoint` *merely synonyms? Are* `Source` *and* `Destination` *unique entities, and is there more than*



(a) Additions to `Node` (b) Rearranged (c) Add `release` (d) Final

Figure 1. Steps in the evolution of the initial domain model diagram by team A

one of each? Appearing to make the decisions himself, Jack used parentheses to make `Waypoint` *an alias of* `Node`*, and he then replicated* `Source` *and* `Destination` *under* `Node`*, with a small right-angled arrow to indicate the connection (Fig. 1(a)).*

Improvising the arrow notation allowed Jack to capture the connection without disrupting his attempts to also capture the barrage of brainstormed ideas. However, the figure became inconsistent with class diagram notation and its semantics unclear. Jack did not clarify the semantics of the connection, and when another participant proposed a specialization relation, he did not hear it or simply ignored it. Nevertheless, he appears to have memorized this notation for subsequent idiomatic use.

As the team continued to brainstorm, a notion was suggested for the inventory of items at a particular location. At first, it was captured via the `Count` *property of* `Node` *and listed with the same right-angle arrow. A replica of* `Count` *with the arrow was then added under* `Path`*. Further discussion of the notion of inventory lead to the realization that* `Source`*,* `Destination`*, and* `Waypoints` *are all special kinds of* `Node`*. The representation was changed, with* `Waypoint` *listed above* `Node`*, and* `Node` *listed under the independent* `Source` *and* `Destination` *entities (Fig. 1(b)). It took a while until the redundancy of these entities under the original* `Node` *was fixed. Later on, after a* `Release` *entity was proposed for the* `Source` *and written down (Fig. 1(c)), someone suggested writing the behaviors in a different color, allowing more methods to be added at various locations around the entities (Fig. 1(d)).*

The resulting diagram, depicted in Fig. 1(d), conveys the same details as would a class diagram used for domain modeling [1]: candidate classes, properties, methods, and inheritance. However, from the point of view of "proper" UML modeling, its notations are ambiguous without the context of specific discussions and elements. For example, inheritance, which in class-diagrams appears as a direct line from a subclass to a superclass above it, appears here as an arrow from a superclass to the name of a subclass below it. The problem is not only with the direction, but also with the replication of the superclass name. In addition, the exact same notation is used to represent a property of a class and is also similar to the notation for a method.

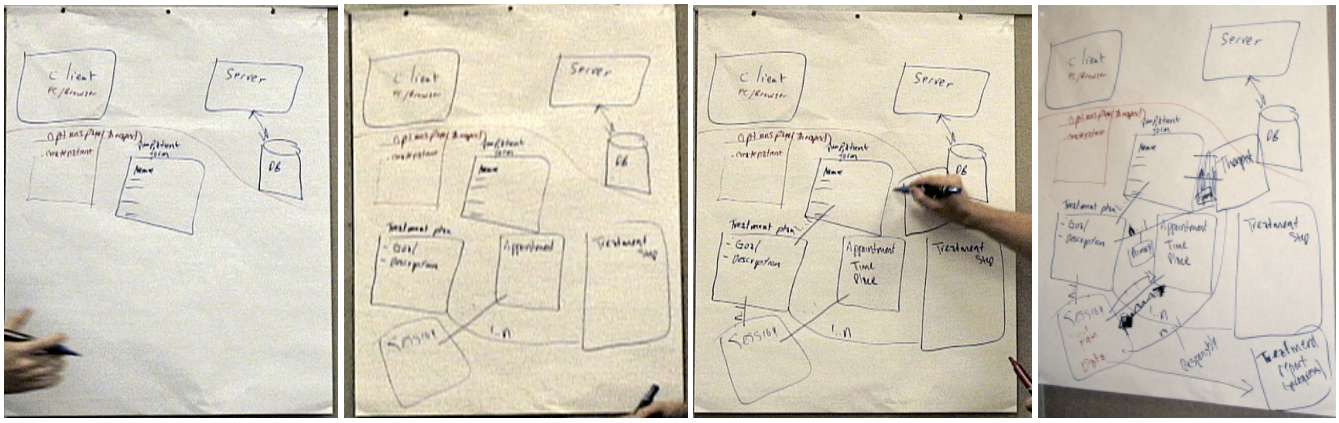
These problems may preclude this diagram from serving as a documentation artifact, at least without contextual information. For example, how would someone viewing this artifact infer that `Count` is a property of `Node` and not a superclass?

Nevertheless, this diagram appeared to facilitate brainstorming as a preliminary step to coming up with a solution. By writing all brainstormed nouns around the board, the team was able to avoid an early commitment to distinguishing classes, fields, and methods, which require different notations in class-diagrams. Furthermore, keeping the related entities for each candidate class in close proximity, sometimes at the cost of replication, allowed the preservation of the diagram as primarily a catalog of entities rather than as an attempt to fully structure the domain.

4.1.2 Example: structural solution model

In our second example, depicted in Fig. 2, team E, which was working on the medical information system, arrived indirectly at a class diagram while retaining some inconsistent notations.

The team was trying to envision the usage model of its system and started drawing a sequence diagram that started with a user making a request. Since users interact with the system and provide it with additional information via web forms, there was soon a need to represent these forms. Craig, standing by the posterboard, placed a page that contained a rudimentary architecture diagram immediately to the left of the sequence diagram, and partitioned it. In the remaining area, he started drawing rectangles for UI forms, writing the names of important actions inside them (Fig. 2(a)). More



(a) UI diagram below architecture (b) Becomes entity-relation diagram (c) More relations added (d) Final

Figure 2. Steps in the evolution of the initial solution model diagram by team E

interaction was specified in the sequence diagram, and additional web-forms were added.

By increasing the spatial proximity between the two diagrams, the team was able to follow and manipulate the representations of two facets of the system at the same time. No evidence, however, remains to alert future stakeholders of the connection between the two diagrams.

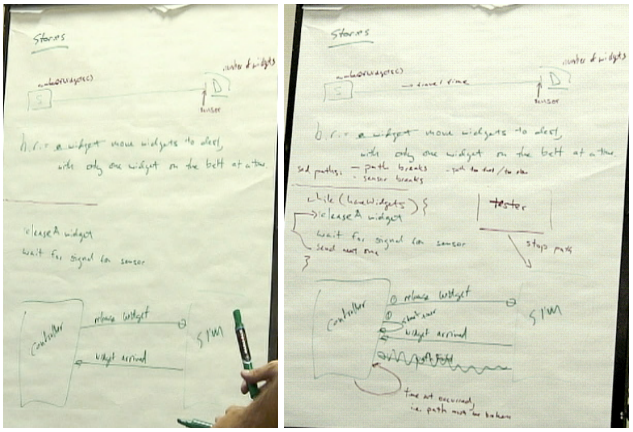
As forms for `treatment plan` *and* `treatment step` *were added, a one-to-many relationship was realized and captured (Fig. 2(b)), followed by more cardinality connections (Fig. 2(c)). These changes essentially turned the form collection into an entity-relation diagram. Then, the team realized that a* `Session` *is a special kind of* `Appointment`*, and added a parallel inheritance arrow to represent it, turning the drawing into a class diagram. In addition, a line representing control- and data-flow was added from the* `Session` *to the generated* `Treatment report`*, thus diverging from class-diagram notation (Fig. 2(d)).*

Although the final version of this diagram resembles a compliant class diagram, it contains additional layers of information such as the contents of some web-forms. In addition, certain entities represent entry points into the sequence diagram, but this contextual information is not captured in writing and is lost. By combining all this information into one diagram, the team was able to treat multiple facets of each entity without replication.

4.1.3 Example: behavioral domain model

Alternate representations evolve not only for structure but also for behavior and function.

Team A *finished the domain model of the first example, and began to explore the interaction between the controller and the simulator with a simple scenario. On a new canvas, they created a small map of the simulated production line, consisting of one source and one destination connected by a single conveyor belt. To describe the behavior, Al wrote down a business rule, requiring a widget to arrive at the*



(a) Early (b) Final

Figure 3. Functional domain model by team A

destination before the next one leaves the source (top of Fig. 3(a)).

Now, Al wanted to describe the behavior of the controller and specified it in two textual steps (middle of Fig. 3(a)). Jack approached the board and added a sequence diagram to represent the interactions between both sides of the system (bottom of Fig. 3(a)). When they realized that the controller's behavior is continuous, Al added a third step, `send next one`*, and created a flowchart arrow from it back to the first step. Someone then mentioned the need for an exit-condition, and Al turned the steps into pseudo-code by wrapping them within a* `while` *loop with curly braces (middle of Fig. 3(b)). Later, the sequence diagram was modified to represent this change, and an external testing agent was added. (bottom of Fig. 3(b)).*

We see that differences in individual perspective can lead to representation choices that differ in their ability to cope with the evolution of the design. Al was focused on the controller and described its behavior in isolation, whereas Jack

preferred a view of the entire system. Jack's choice to use a sequence diagram later helped cope with the introduction of an additional entity, the tester, into the system. However, such diagrams typically represent only a single execution path and could not be naturally extended to meet the evolving need to model iterative and conditional behavior.

The textual representation chosen by Al, on the other hand, was more malleable and was complemented by borrowing idioms from familiar representations. The series of steps seemed sufficient for sequential code, but iterative behavior demanded the representation of control flow, and the arrow notation of flow-charts was promptly borrowed. Flowcharts, however, bloat under complex control flows, and the more concise syntactic elements of a programming language were quickly used.

4.2 Customizing level of structure

Everyday experience tells us that the hand-drawn diagrams created early in the design process typically appear less aesthetic and organized than their finalized versions, especially electronic ones. In some cases the difference is purely aesthetic and the diagram can be rearranged and re-rendered without actually changing the structure [5]. Other sources [8] suggest that earlier models are simply less complete and omit content and connections outside the current focus; the missing details are incrementally added later. While we witnessed many examples of less aesthetic diagrams and of less complete ones, we also observed early diagrams which appeared visually different because of a choice to use a different level of structure.

4.2.1 Starting with unstructured representations

Notational standards such as UML typically define several diagram types, each with its own primitives. Software tools that support them often require a user to specify the diagram type in advance and only allow the use of the appropriate primitives. However, as we have seen in the previous examples, an early commitment to a diagram type is not always practical. In fact, teams may initially work at such a simplistic level that no diagram type is appropriate, since it would incur significant costs and may constrain them with irrelevant structural details.

The most typical example of this behavior occurred as teams began working on a new canvas and started brainstorming a homogenous set of entities. For example, if team A was forced to use a class diagram when constructing the domain model of Fig. 1(d), the cost of surrounding each new entity in a box might have been marginal. The real cost would have come from constraining the ability to modify and experiment as certain entities turned out not to be classes but rather properties or methods.

It appears that when teams expected their design to evolve, they tended to prefer simpler representations that imposed less constraints. Often, they chose to use text in the form of lists or scattered elements. As the design evolved, relations and annotations were gradually added. In a few cases, the teams used this opportunity to complete the diagram towards UML. For instance, team F took separate diagrams conveying a list of actors and a list of activities, placed them adjacently, and connected them to form a use-case diagram, as can be seen in Fig. 4.

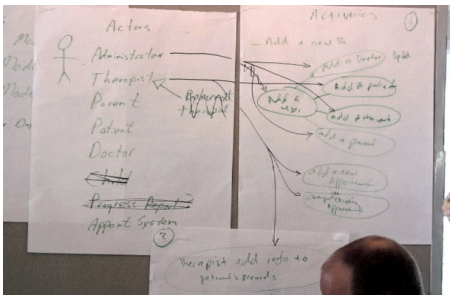
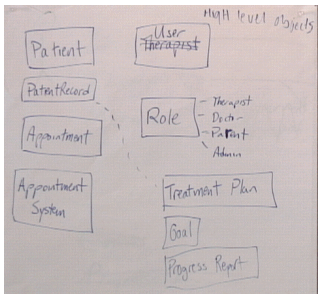
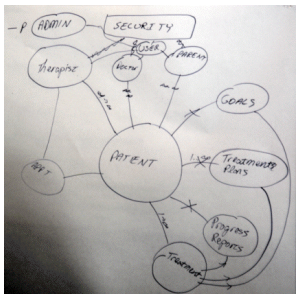


Figure 4. A use case diagram composed by team F

In most cases, however, teams continued to diverge from UML. They represented the added information by repeatedly using improvised notations, as in the case of team A, or borrowed idioms from other notational standards, such as the inheritance arrows and relation cardinality annotations from UML class diagrams. The repeated use constrained the representation and implicitly added some structure to the diagram. This structure might eventually help in the interpretation of the diagram even if the notations are unclear, as one could likely do with team A's domain model from Fig. 1(d).



(a) Team F



(b) Team G

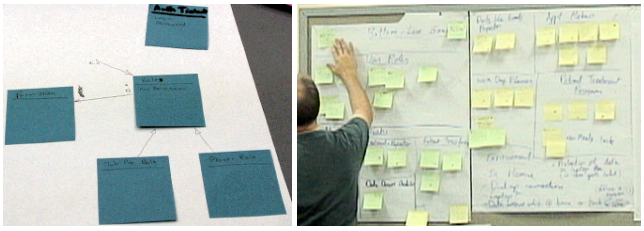
Figure 5. Connector notations in two object diagrams

When reuse was limited, and in particular when certain notations had only a single instance, we were often left with less structured and consistent representations that were more difficult to interpret. For example, Fig. 5(a) depicts a diagram by team F, which was working on the medical system. The diagram primarily lists entities, but it also conveys several unclear relations. The relations between `role` and the several descriptive entities to its immediate right use the same notation and could thus perhaps be interpreted as examples or instances of the same concept even before we consider the actual text of these entities to verify this assumption. The relationship between `patient record`

and `treatment plan`, on the other hand, is unique in this diagram and therefore cannot be interpreted with confidence without contextual knowledge from the actual session.

Even if some notations are reused, however, the benefits can be offset if the emergent structure becomes too complex. Consider Fig. 5(b), which conveys a diagram of objects from team G, which was also working on the same problem. In this diagram, it appears that the team initially brainstormed entities and scattered them about the board. When they needed to relate the entities they used generic lines, and then decorated them with borrowed cardinality notations and with an improvised arrow notation. Nevertheless, the complexity of the graphs makes it difficult to interpret the improvised notations or the undecorated lines without additional contextual information.

Finally, we note that while lists or scattered sets of entities on paper allowed teams to begin work with limited structural restrictions, even more freedom was available to those who used sticky-notes. For instance, one team in the 2006 study created entities on stickies, placed them on a canvas, and eventually connected them to form the class diagram of Fig. 6(a). Many teams, including team D in Fig. 6(b), added structure by creating hierarchies of bins on a large canvas and placed atomic elements on sticky-notes The use of sticky-notes in OOD deserves further focused study, and may benefit from specialized support in electronic whiteboards [17].



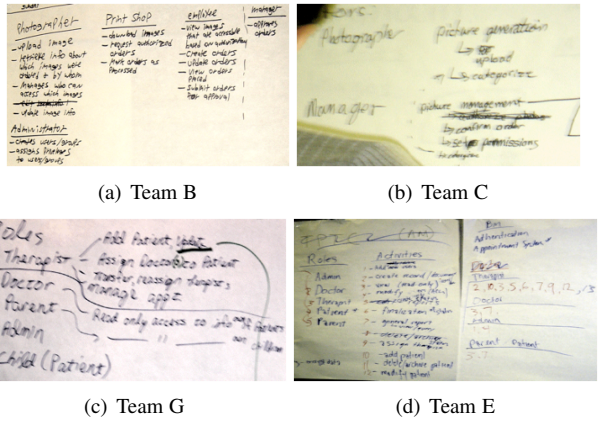
(a) Creating a class diagram (b) Creating custom hierarchies

Figure 6. Uses of sticky notes

4.2.2 Working with more structure

Based on casual observations, one may expect that freeform design sketches will always have equal or lesser structure than models adhering to standard notations. After all, these notations are often designed to accommodate several layers of complex information. While this may generally be true for UML class diagrams and sequence diagrams, use-case diagrams revealed an opposite phenomenon, where seemingly improvised non-conforming notations actually added structure rather than omitted it.

At its core, a use-case diagram (UCD) is a bipartite graph that matches a set of actors with a set of use-cases,[1] thus offering a straightforward way to represent this many-to-many relationship. However, to identify the UCs associated with particular actors, or the actors associated with particular UCs, one must trace all outgoing edges. This incurs significant cognitive effort since the UCDs for large or dense relations tend to become cluttered [10].



(a) Team B (b) Team C

(c) Team G (d) Team E

Figure 7. Alternative use case diagrams

While all but one team (A) in our study explicitly listed actors and UCs or activities, only three of them (C, F, G) drew actual UCDs. All other teams, as well as team C after it had already created a standard UCD, preferred to connect the actors and UCs using the more structured tabular, textual, and numerical forms depicted in Fig. 7.

It appears that for the two information system problems, teams tended to partition the set of use cases by an associated primary actor. This partition effectively created a one-to-many relationship that could receive little benefit from the bipartite representation and yet could become difficult to follow. Teams thus preferred to represent this partition in structured textual representations, which are more organized and carry lower cognitive demands but cannot be used for many-to-many relationships.

Use-case diagrams are particularly susceptible to replacement by more structured representations because their inherent structure is so limited. Class- and sequence- diagrams, on the other hand, are very structured, and we have thus seen them replaced by alternate notations mostly in the opposite direction, of less structure.

Nevertheless, even these diagrams types may occasionally be replaced by more structured representation. For example, flowcharts, pseudo-code and actual source code are more structured than sequence diagrams because they inherently support complex conditions, control flow, and reuse behavior that is not natural to sequences; this may explain their use in the earlier example of the behavioral model. Similarly, class diagrams are limited in describing the contents of each individual class since they simply present a list of members. We have occasionally seen teams devoting a separate diagram to the contents of a specific class, giving them

[1] Another layer of information connects use-cases to represent extensions and dependencies, but we rarely saw its use.

more freedom in specifying and manipulating its members. Some teams even created bins within the representation of a class into which members were sorted, effectively creating another level of abstraction within the class.

5. Results: Representing heterogeneous information

In the previous section we focused primarily on individual diagrams that diverge from UML notation or use alternate representations. Individual diagrams, however, tell only part of the story. As the teams' understanding of the problem and solution evolves, their knowledge and designs consists of different types of information. To fully understand how designers represent this designs, we must investigate how they deal with this heterogeneity.

5.1 Using independent diagrams

Since software designs cover multiple facets like structure and behavior, two-dimensional notational standards like UML typically consist of multiple diagram types [14]. The UML standard dictates that each diagram will use the notation of exactly one diagram type and restricts the use of foreign annotations. These restrictions are usually followed in formal design documents, and most CASE tools enforce them by providing a separate drawing canvas for each diagram, and offering only the drawing primitives of the chosen diagram type.

DesignFest teams, however, use physical mediums and are not forced to produce a formal design document. They are therefore free to violate this restriction. Nevertheless, it appears that initially they did try to adhere to the standard,

The ability to increase the spatial proximity between diagrams, which is relatively straightforward with paper, appears to aid teams in coping with the heterogeneous nature of the design. For example, in the earlier example of team E which was working on the medical system (Fig. 2), they had to consider the parts of the user interface visible to the user as well as the activities behind the scenes. This lead them to work concurrently on the sequence diagram and the UI diagram, which were placed on separate but adjacent canvases.

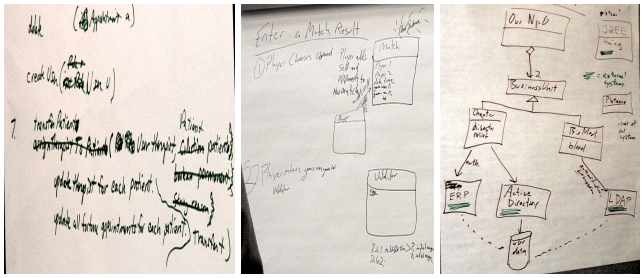
When smaller diagrams were required and the use of separate large canvases for each diagram was impractical, teams appeared to relax the restriction a little, allowing diagrams of different types to reside on the same canvas. For example, we have seen team E using the area below the architectural diagram to create the UI diagram, in order to conserve space.

Teams may also intentionally bring multiple small diagrams together onto the same canvas in order to increase locality. In the earlier example of team A's behavioral model (Fig. 3), the exploration of a scenario resulted in multiple artifacts. While each artifact serves a different role, it must be interpreted in the context of the others and kept consistent with them, thus benefiting from increased locality. A similar multiplicity, involving a sequence diagram and a more complex map of the production line, appeared when the team later explored a more elaborate scenario.

5.2 Combining diagrams

While teams generally tried to keep diagrams of different types separate, they occasionally violated UML practices by combining information from different diagram types into a single artifact. This typically involved combining behavioral and structural information.

For example, team F devoted an entire canvas to the methods of a single business object (Fig. 8(a)) in a manner similar to a class diagram, but in addition to listing the method signatures, they also specified implementation details which have no place in such diagrams according to the standard. We observed a similar phenomenon in our 2006 validation study, when a team working on a tournament management software textually elaborated the steps of a use case and added class-diagram elements to represent related entities (Fig. 8(b)). In both cases, the added information is closely coupled to the context of the primary diagram type.



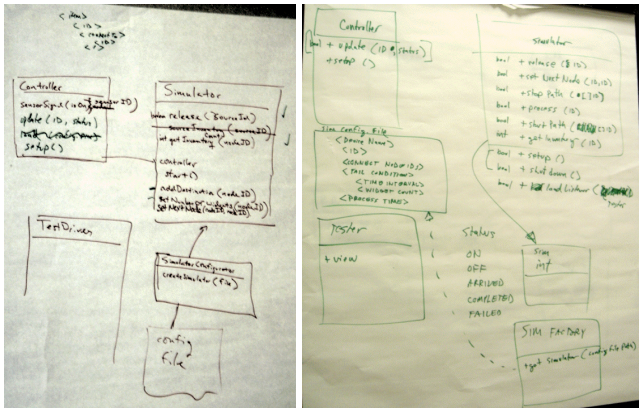
(a) Implementations steps in a class diagram (b) Class diagrams in use-case steps (c) Architecture in class diagram form

Figure 8. Combining structure and behavior

In addition, we frequently observed teams augmenting UML class diagrams with indications of control- and data- flows. For example, in the earlier example of the evolving class diagram by team E from Fig. 2(d), the team created an output entity for a `Treatment report` and added a line from the `Session` class to represent the flow of data used to create it. Similarly, the three outgoing edges from the `Treatment` object in the data model created by team G from Fig. 5(b) appear to represent some form of control or data flow.

One team in the 2005 study used class diagram notation to represent the system architecture and the interactions within it, as depicted in Fig. 8(c). Other teams introduced external systems and storage mediums into standard class diagram. For example, Team A was creating a class diagram and discussing how the `Simulator` class could be configured. They added a configuration file and a class for loading it, and then used arrows to model the flow of data from the file to the simulator via the loader class (Fig. 9(a)). This hybridization of class structure and data flow is apparently not accidental,

as it was repeated in a subsequent finalized version of the diagram (Fig. 9(b)).



(a) Initial diagram (b) Later diagram

Figure 9. External elements in class diagrams by team A

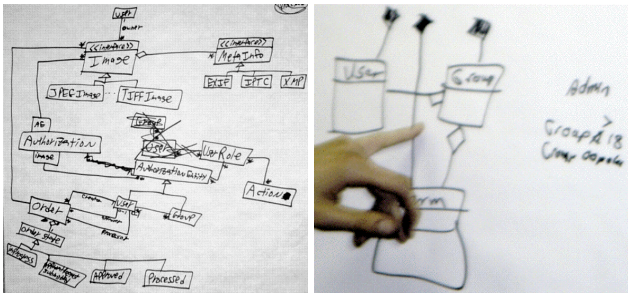
By combining diagrams or introducing foreign elements, teams increase the locality of information while reducing the clutter, effort, and redundancy that stems from the use of independent diagrams. While these are tangible benefits, this behavior appears to have a more fundamental motivation that is rooted in how designers think about the system and its components. Early design discussions typically revolved around objects rather than classes and referred to their structural and behavioral properties in ways that cannot easily be disentangled. While notations that dictate a separation of these facets offer ways to accurately model the details of each facet without interference from the other, this has limited benefit in these early stages, while the costs of forcing a disentanglement may be significant.

5.3 Introducing peripheral information into diagrams

The tendency to increase the locality of information in the diagram is also evident from the ad-hoc integration of concrete examples, details, and instantiations into the diagram, rather than placement in external documents or use of annotation standards.

The UML standard, especially as implemented in CASE tools, accommodates peripheral information only in specially marked "notes" that lie in proximity to- or in direct connection with- diagram elements; these constitute a semantically- and visually- separate annotation layer. The last example shows that in collaborative work, the boundaries are not as clear, at least visually: the configuration file with its proposed format in Fig. 9(b) looks at first like an integral part of the diagram. Similarly, as team B was working on the image shop problem, they integrated examples of images, metainfo types, and order states into the class diagram (Fig. 10(a)), even though it appears that these were not intended to eventually become classes.

Concrete examples, not always textual, were often added in an ad-hoc manner while discussing a recent idea or addition. For instance, during the initial discussion about the configuration files, a member of team A wrote a concrete example of its format on the top of the page (Fig. 9(a)). Similarly, in response to questions about security, a member of team C drew a small incomplete class diagram (Fig. 10(b)). As more questions were raised about the data relations, he added a sample record next to it.



(a) Class diagram by team B (b) Sample record by team C

Figure 10. Integrating examples into diagrams

While examples were typically placed on the same canvas as the entities they refer to, we saw limited spatial proximity or explicit connections between them within the canvas. For instance, the data format appearing at the top of Fig. 9(a) is visibly remote from the configuration file at the bottom of the figure. In fact, external observers might consider it unrelated to the diagram or relate it to the methods of the controller as both use a distinct color. It appears that explicit proximity or connection, as practiced by UML, was not necessary since the example was given in a specific context. When creating documentation-oriented artifacts, teams appeared to explicitly capture these associations, as is evident by the relocation of the format into the configuration file (Fig. 9(b)).

We note that important context-sensitive design information is not always fully captured in writing. It may sometimes be captured as more of a placeholder or reminder of other information and be difficult to interpret without the contextual information. For example, only meeting participants will recognize that the annotations on the methods of the simulator in Fig. 9(a) convey that some methods are private while others correspond to a certain scenario. Similarly, the meaning of the crosses or the arrows on the edges in the earlier example of Fig. 5(b) is unclear but likely denotes some concept discussed in the session.

Worse yet, many notions and ideas in the course of discussion are not expressed with permanent visual markings but rather created with hand movements or even sketched in the air or with a capped pen over the board. All evidence of these gestures is lost in the final diagram, although they potentially convey important annotations and examples. Designers often appeared to use this mode of communication to avoid cluttering the paper, so it is possible that interaction would have been different over a dry-erase whiteboard or an

electronic medium. Nevertheless, this behavior demonstrates that pertinent design information may not be explicitly captured in an artifact but rather expressed in its context.

6. Results: Dependencies between diagrams

The results presented so far show that even when teams are given freedom in their choice of representation, their designs are still dispersed over multiple diagrams. In this section we focus on the dependencies between these diagrams, how they change over time, and how teams cope with them.

6.1 Diagram evolution across canvases

In examining the representations of individual artifacts, we saw diagrams evolve and change their type or focus in response to ad-hoc design needs. As the diagram becomes more visually dense, however, it can no longer evolve "in place". Especially when using physical mediums, larger segments cannot be manipulated with ease, while striking out contents is detrimental to the diagram's aesthetics and may overload short-term memory if material is recreated. Thus, and as we have seen in our preliminary study [10], a single design or final diagram can sometimes evolve over several versions, each on a different canvas.

The model of continuous evolution towards complete UML described in the literature [5, 9, 28] implies a monotonically increasing shift towards completeness in content and notation. When the evolution is not in-place, this model implies that each new version should convey at least the same information as its predecessor (except for intentional revisions), thus rendering all previous versions redundant. However, reproducing the contents requires a significant menial effort, which designers are likely to try and minimize. An issue of significant concern, therefore, is whether they would perform this menial task or spread the design over multiple incomplete versions, potentially leading to a situation where the final version might not stand on its own?

Let us consider one such situation, occurring as team E continued working on the CD for its medical information system (Fig. 2(d)). As a result of the process by which the diagram evolved, it has become quite cluttered, with key entities, such as `Patient`, still represented as forms rather than as classes.

Craig described the diagram as 'a mess' and suggested that they clean it up; someone suggested summarizing and Craig agreed, saying they should also capture the relations. They proceeded to create a new diagram with a clearer and more spacious layout. It began as an entity-relation diagram, but became a class diagram as inheritance was added once again. In creating the new diagram, the team fell into a pattern, depicted in Fig. 11: Craig would turn his body or walk towards the original diagram and identify an important entity or relation. A discussion would ensue, followed by rendering a version on the new diagram, after which the pattern repeated itself.

Figure 11. Team E recreating the data model

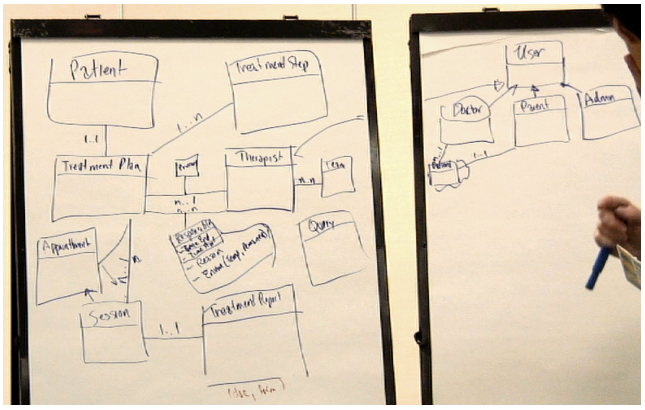


Figure 12. Revised data model by team E

The final version of the diagram, depicted on the left canvas of Fig. 12, conforms with prior research which suggests that artifacts are often recreated with equivalent content to improve aesthetics or to migrate into an electronic tool [6,8], or with less content to highlight specific details [9]. However, our preliminary study [10] suggested that the mundane activity of recreating artifacts serves not only aesthetic purposes, but also offers a chance to inspect, reconsider, and improve past decisions, resulting in different content. What, then, is the relation between the old and new diagrams in the work of team E, and to what degree are aesthetics the primary difference between them?

As cleanup progressed with elements transferred to the new diagram, certain decisions were revisited or expanded. For example, at the end of a long discussion, the one-to-many relation of `Responsibility` between `Therapist` and `Session` from the original diagram became a many-to-many relation between `Therapist` and `Treatment plan` in the new one and received several associated properties. Afterwards, the team created a relation between `Plan` and `Appointment` to indicate that a plan can consist of both appointments and sessions rather than solely of the more specific sessions. Later, the flow arrow from `Session` to `Treatment report` was replaced by a one-to-one relation. A discussion on a reporting infrastructure ensued, and though never finished, a general `Query` class was added to represent this functionality.

This example offers anecdotal evidence that designers may conserve effort by avoiding the replication of some

pertinent information, such as the fields of the `Treatment plan` class, to the new diagram. Grasping the complete design thus requires aggregating information from both versions, which is made difficult by the differences between them. It is also just one of the many situations we encountered in which substantial and often conflicting differences exist between the earlier version of the artifact and the revised one. The difference may not be noticed without careful comparison of both diagrams. Determining the final decision would require knowledge of which diagram was newer, and determining the rationale behind them involves recalling the discussions at the time of the transition.

Many diagrams evolved through more than two revisions. In general, relatively early after teams began working on a new version, they seemed to determine the magnitude of differences from the previous version. When it appeared that only minimal changes would be necessary, they tended to split into smaller groups and produce an aesthetic and complete finalized version. Otherwise, the team remained cohesive, and less attention was paid to completeness and aesthetics, perhaps in anticipation of yet another revision. Nevertheless, the early determination was not always accurate, resulting in significant decisions being made at the subgroup or individual level.

We note that the two designers using CASE tools to capture a finalized version of the work of their teams on-the-fly also made frequent individual decisions, which occasionally even conflicted with those of the team. These designers were immersed in their personal devices and thus lost touch with their group's focus. In addition, the speed at which the tools allowed them to capture decisions appeared to further draw them away from the slower and more collaborative activity of their peers. Further study is necessary to assess whether personal devices are constructive or detrimental to collocated collaborative OOD.

6.2 Noncontiguous artifact evolution

The previous example demonstrated a localized and contiguous evolution from one version of the artifact of focus to another version of the same artifact. Earlier, we have seen interweaving of work on multiple diagrams which evolve together, as when team E was creating the data model of Fig. 2(a) while working on the sequence diagram, or when team A was working on the behavioral model using multiple representations (Fig. 3). In these situations, there was still a locality in time and space and no additional artifacts were used or changed in the interim.

Such locality, however, is not always the case since, as we have seen, ad-hoc needs often divert discussions in different directions. Timelines constructed for the observed sessions show that teams tended to work for a while on one primary artifact and then shifted their attention to another. An artifact may be abandoned at some point, only to be recalled at a later point and be discussed, referenced, copied, or continued, on the same canvas or on a different one.

Team A *was working on the class diagram of Fig. 9(a), which was still located on the flipchart. In the course of one long discussion, the team's assumptions came up, and the focus changed to a list of assumptions posted earlier on a different posterboard. After adding assumptions, they came back to the* CD *for another ten minutes, after which they flipped the chart to an earlier unrelated diagram, and worked on it for a while, occasionally also working on the assumptions list. After returning to the* CD*, they began making the changes in green (controller methods, data format, and markings near simulator methods). They posted the page and turned to working on scenarios for an hour. Only towards the end of the session did they begin creating the revised version of Fig. 9(b).*

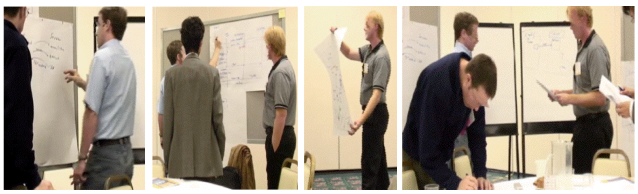
This example shows that individual diagrams do not evolve in a vacuum. Instead, each change takes place in the context of the current state of the design at that time, which could be captured in artifacts that share no obvious connections. For example, certain design decisions were made in the context of the original assumptions list, while others were made in the context of the new ones. The artifacts do not capture these temporal dependencies, potentially presenting a significant challenge for the eventual interpretation of the meeting products.

6.3 Dependencies on multiple diagrams

The network of dependencies between artifacts is further complicated by the flow of design information from multiple sources, typically earlier diagrams, into a single "sink", often the current artifact of focus.

Having finished discussing reports, the E *team turned to the relations between the entities representing actors in the system. They referred to a small diagram, created very early in the session, which described inheritance relations among* `User` *entities, including ones that did not appear in the original diagram of Fig. 2(d). These entities and relations were then copied into a separate page, and data cardinality connections were added. As can be seen in Fig. 12, this diagram was then placed adjacently to the right of the main diagram and a connection was made, effectively integrating the new part into the main diagram.*

The additions to the class diagram are compliant with UML, but much of the design rationale behind them lies in the context of the differences from the original hierarchy, and their integration into the larger diagram.



(a) (b) (c) (d)

Figure 13. Team E building an architectural model

Another example of how multiple artifacts are used, integrated, and discussed, occurs towards the end of the E team's session:

With time running out, the team decided to capture the high-level architecture of their system, which they initially abandoned very early in the session. In constructing this new diagram, they had prolonged discussions and referred to several artifacts, shown in Fig. 13, including: (a) *a simple model of client and server responsibilities,* (b) *the sequence diagram,* (c) *the architectural sketch on the same canvas as the discarded original model, and* (d) *the problem-specification document with its notes. The resulting diagram adds little beyond the original sketch, but many architectural decisions were made verbally in the context of the referenced artifacts but simply not written down as time ran out.*

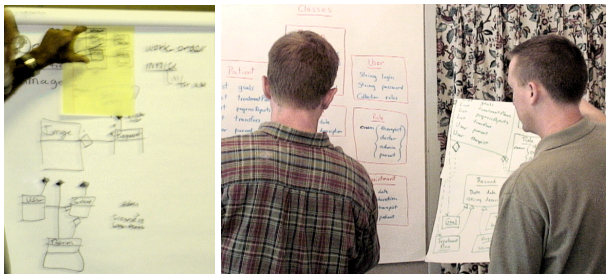
The above examples show that even though individuals maintain much design knowledge in their heads and could implicitly apply it to the current diagram, they instead explicitly reference it in existing materials. A likely explanation for this phenomenon is the need to ground the knowledge and offer context to the upcoming discussion while ensuring that that it is shared by all participants. In addition, since we frequently saw individuals examining artifacts before speaking, it is possible that peer pressure leads designers to "check their sources" before contributing.

6.4 Coping with multiple artifacts

The dependencies between design artifacts are a known challenge to all designers, but may be particularly problematic for collaborative teams working with physical mediums. The need to make artifacts visible to everyone requires the use of large shared drawing spaces and results in the creation of large scale artifacts. The team's workspace is usually limited, allowing them to keep only a limited working subset visible at all times [10]. Even the artifacts in this workspace are typically spread around the design environment, preventing designers from seeing all of them at once.

For these reasons, when teams needed to continuously focus on multiple artifacts, they tried to increase the physical locality. If the diagrams were not on the same canvas, then they were moved around and placed next to each other. For instance, we saw team E place the UI and sequence diagrams next to one another, as elements in the former were entry points into the latter. Similarly, we saw team F place lists of actors and activities next to one another and compose them into a use-case diagram (Fig. 4).

When the need to focus on multiple artifacts was transient, diagrams were often held in proximity without being physically attached. This was especially common when recreating a new version of a diagram from an earlier model or a personal note (Fig. 14), after which the original was often removed from the working set. There was limited need to affix the diagrams because the old version could always be brought back if necessary (e.g., Fig. 13(c)). There are situations, however, where diagrams were held in proximity in the course of conveying ideas. For example, when team F worked on a scenario involving user actions, a designer would often hold up a sketch of a web form and a sequence diagram, demonstrate a user action on some widget, and then continue to follow the sequence of actions in the sequence diagram until the output was produced back on the form. In these situations, no permanent evidence was left of the connection between the artifacts.



(a) Team C (b) Team F

Figure 14. Holding diagrams to increase locality

As we have seen, the content of one artifact can affect the discussion or content of another, even if they do not share mutual entities. While individuals may possess the relevant knowledge in their minds, they appear to reexamine the artifacts before speaking and later refer to them explicitly. Unfortunately, the effort and time involved in moving and arranging artifacts around the workspace prohibited the establishment of physical proximity in many of these cases and forced teams to repeatedly switch their attention between different parts of the design area. This was particularly evident for shorter references to materials posted around the work area and especially when individuals, rather than the entire team, were considering the materials or referring to their personal notes or the given documentation. The only physical manifestation of these public references were gestures and pointing, while the only clue of private references was typically a glance or a change in head position.

Nevertheless, when the entire team needed to join the focus of an individual, gestures and gaze tracing were key [10]. Gaze tracing also helped individuals remain oriented on the focus of the team. For instance, if the participant opposite the board in Fig. 15 was distracted, he could interpolate the gazes of his four peers to identify which of the three diagrams they were focused on.

7. Discussion

The collected body of data from three years of *DesignFest* events contains numerous examples of representations used in collaborative OOD, many of which significantly diverge from UML, and of dependencies between the diagrams. However, our goal was not to merely confirm the casual observations and everyday knowledge of their existence, nor was it to try to catalogue them or to elicit quantitative in-

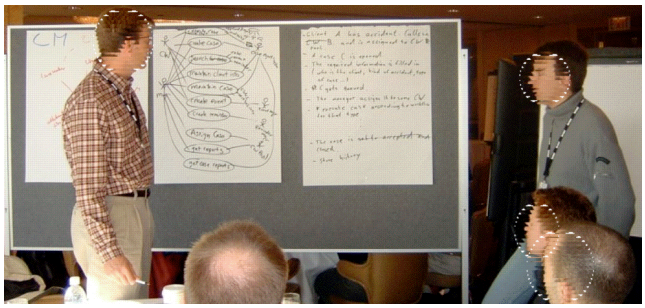


Figure 15. Gaze can be traced to determine group focus

formation, which would have little use in these restricted and unique small settings. Rather, in initiating this study, we set out to understand the representations used in collaborative OOD, the reasons for their creation, and their implications for tools. In the preceding result sections, we presented representative examples of behaviors that, we believe, shed much light on these questions.

When we set out to explore the representations, we expected artifacts to primarily be examples of incomplete diagrams with accidental freehand notations that could eventually be evolved into complete UML models, as described in the literature [9]. Perhaps because of the less restrictive settings of our study, the actual data revealed surprising results: While we did see many incomplete diagrams, the representations ended up being too varied to fall under any simple classification. Most importantly, they appeared deliberate rather than accidental.

We have seen many instances in which the divergent representation could not simply be dismissed as early forms of UML, or as independent and unrelated freehand annotations. Instead, there appear to be situations in which teams create artifacts that convey the same data as proper UML diagrams, but make use of improvised notations to quickly solve immediate problems or capture insights before they are lost. One path for the evolution of such representations occurred when teams began with a less structured representation, and then introduced additional notations and structure. A second path is when they chose a representation that could offer more structure or cover more facets of the design than a standard UML diagram.

In addition, rather than create the portfolio of visually independent diagrams freqently seen in design documents and CASE tool models, teams deliberately created a large, interdependent and seemingly disorganized array of artifacts. Some of these combined multiple diagrams, types, and annotations, while others depended on or overlapped with other artifacts. In addition, there is a complex network of dependencies, connections, and locality between the artifacts, many of them transient and context-sensitive and in some cases subtle. Some of these connections are spatial, involving the location of the artifacts at specific times, while others are temporal or contextual, involving their state or use at given times

Let us now try to explain these results and discuss their implications.

7.1 Ad-hoc choice of representation

Existing explanations for the divergence between design sketches and standards such as UML tend to follow two themes. First, early artifacts are mostly incomplete, and can evolve into conformance with the formalism with sufficient effort and guidance [8, 15]. Second, there are criticisms of the standard itself, such as its power of expression, elegance, level of restriction, or approach to organizing data. Thus, might be expected to "rebel" against it or at least adopt additional notations. We shall relate to the first theme now and discuss the second at the end of this section,

Based on repeated close studies of the videotaped evidence, of which pertinent examples were presented in this paper, we propose an explanation for the choice of representation, and the divergence in notation in particular, that is more fundamental and less dependent on the specifics of formalism and settings. Namely, we suggest that while teams are aware of the need to convey their design for future use, and will explicitly work towards that goal in later and visually distinct phases of the design, that is not their primary motivation and concern.

Rather, in the creative phases of the design, both the design process and the representations used to capture it are structured as an ad-hoc response to the team's unfolding understanding of their problem and solution. The ad-hoc approach allows teams to capture the results of their problem-solving process in whatever order that it happens to take, and tackle issues in the order they choose, often using sketches as a short-term memory.

In respect to representation, ad-hoc choice serves several purposes. One obvious benefit is in allowing teams to minimize the costs in distraction and physical effort that arise from visual activity and in particular from adhering to a complex notational standard. Like previous researchers, we saw teams filtering out unrelated content [8] and skipping aesthetic polishing [5, 24]. We also saw them avoiding the menial chore of copying all content when creating a new version of a diagram, effectively spreading the design across several versions. In addition, we have frequently seen participants using gestures and "air-pens" rather than actually drawing, to avoid the associated costs of writing and subsequently erasing materials. This was also evident when teams used small and portable canvases or sticky notes that could be temporarily attached to larger sheets and moved about [10].

A second related benefit is that it enables them to localize different types of information related to an issue, thus reducing physical clutter and memory load. Teams tended to increase the physical locality of relevant materials: they shuffled canvases around or placed related diagrams on the

same canvas, and even integrated different materials into the same diagram. However, this increase in locality is not only physical; it also extends to a reliance on individual and group memory, on the immediate context, and on alternate communication mediums like speech or gesturing. This locality acts as a substitute for the need for specific and well-defined references and notations.

A third, less obvious but more fundamental, purpose of ad-hoc choice is that information could be represented at levels of completeness, abstraction, structure, and organization that are best adapted to the team's current and anticipated needs. In some cases, the available representations could not match the level of structure at which the teams wanted to work. In other cases, by avoiding an early commitment to a particular representation, teams were able to allow their representation to evolve in ways not possible with a fixed representation.

7.2 Impact on products

As we have repeatedly seen throughout this paper, the priority given to creative design and the attendance to ad-hoc needs comes at a cost: teams create an unedited collection of artifacts with certain dependencies, inconsistencies and ambiguous notations, that may be less comprehensible and useful to outsiders, thus lowering their potential as documentation or implementation artifacts. This poses a problem because sketches as well as archival-quality artifacts from design meetings are subsequently used by developers, who need to understand decisions made in the design meeting [6]. They also frequently need to understand the rationale behind these decisions [18], which is often not documented explicitly [20]. Furthermore, problems in interpreting UML diagrams have been implicated in some software defects [19].

Even some of the explicitly-recreated and finalized diagrams that we have seen will present a challenge to external observers due to ad-hoc annotations, foreign elements, and the differences between versions. For instance, a hypothetical external observer will be challenged to understand the annotations on the methods of Fig. 9(b) or where the status codes fit. Understanding the complete solution of team E, such as its data model of Fig. 12, will require him to locate the earlier versions, such as Fig. 2(d), identify the differences, and elicit some details from the original. Interpreting all these representations would require familiarity with the context of their creation.

7.3 Order of evolution

Throughout all observed sessions, it was evident that designs and artifacts do not evolve on a single continuous path of monotonically-increasing completeness. Rather, in response to ad-hoc needs, the designs and artifacts evolve on multiple paths that are interwoven and sometimes merged. Every artifact or idea may be abandoned, only to be recalled at a later point and be discussed, referenced, copied, or continued on the same canvas or on a different one.

The interwoven and noncontiguous order of work presents several challenges. During the session, it challenges the team members' memory, individually and as a group, and they spend much effort attempting to recall and recap past discussions and decisions. Physically, it also increases clutter and confusion as the team struggles to maintain a limited working set of diagrams, requiring them to search, move, and flip diagrams. It also depreciates the value of spatial cues which are essential to locating material [10]. The impact also persists after the session has ended since the loss of temporal order can present challenges to interpreting the final diagrams.

For instance, because team A made changes to its posted list of assumptions while working on the diagram of Fig. 9(a), a reader examining earlier artifacts might mistakenly assume that all listed assumptions were made at the same time at the beginning of the session, and thus interpret those artifacts in this mistaken context.

We note that identifying the connections and interweavings between artifacts is also challenging. Access to artifacts can vary in length and impact, from longer references that involve copying, making changes, or even a return of focus and a continuation of discussion drawing, to short references, acknowledged only in glance and speech. Individual designers leverage the extreme collocation to maintain awareness of what others are doing and what they are focused on. As could be expected, we have seen coordination problems arise when teams split into subgroups [10].

The nonlinear path of the design evolution also exacerbates the problem of understanding the results, as it is difficult to interpret the resulting artifacts without understanding how and why they evolved from the originals. While the design rationale is rooted in the context of both versions of the artifact, it may only be captured in the discussion itself, and the difference between the artifacts may be the only hint that such a discussion ever took place. In addition, subtle references to artifacts and temporal clues that may have important impact are not captured.

7.4 Use of UML as an idiom

We conclude our discussion of the results by touching on the use of UML.

Although our presentation focused on divergences from UML, it is important to note that the representations of most artifacts are still based on that formalism and often comply with it. This is particularly evident in finalized versions created for presentation: UML is apparently considered appropriate for documenting and communicating designs.

Testimony to its communicative qualities is offered by the fact that some of its notations become idioms that are applied almost automatically, even outside the expected context. For example, the inheritance notation of class diagrams is so well recognized that we have frequently seen it used to indicate generalization and specialization relations among other entities, such as actors in use-case diagrams. The use of inher-

itance to convey examples in the class diagram of Fig. 10(a) is another possible example of this phenomenon. These idioms even make their way into architectural diagrams. For example, in a freeform architecture diagram created by another team working on the image shop problem (Fig. 8(c)), we can see the UML notations for inheritance and aggregation used in a diagram that primarily conveys components and the interaction between them.

In fact, after examining the plethora of diagrams in three years of *DesignFest* events, we raise the possibility that the use of UML in the earlier and more creative stages of collaborative OOD, before final diagrams are created, is not a real use of the standard or of a consistent subset. Rather, it appears to be a use of freeform notations that borrows and utilizes several well-recognized UML constructs as idioms. This hypothesis may offer some explanation for the use of only a handful of UML diagram types and of a very limited subset of their available primitives, even though many of the participants were very experienced UML modelers.

When not using the idiomatic constructs, representations tended to devolve into the box-and-arrow diagrams observed in non-OO settings [6]. This presented only a limited difficulty during the meeting, because interpretation was implied by the context of the conversation. However, it presented a problem for external readers who lack syntactic or contextual cues to interpret them.

Finally, we want to clarify that we do not consider the findings of this paper to be indicative of specific weaknesses of the UML standard compared to other potential notations. Our observations are not merely the result of studying a particularly problematic formalism. While they highlight inadequacies of UML in supporting collaborative design, UML is primarily a specification- and documentation- oriented notational standard.

The important implication of our observations is that it is not clear that any current or future fixed standard with these goals would be flexible enough. The same factors and ad-hoc needs will likely lead designers to improvise around its restrictions as well. Thus, instead of trying to improve collaborative OOD by attempting to find a perfect formalism, perhaps an investment in tools that are independent of specific notations may be more rewarding.

8. Implications for design-support tools

Our discussion so far has focused on understanding the representations used in collaborative OOD. We shall now turn to identifying implications for design-support tools.

8.1 Utilizing large displays

As described in Sec. 2.3, most existing efforts for supporting collaborative OOD focus on using electronic whiteboards as shared drawing spaces, typically using sketch recognition to identify primitives of the notation. These primarily focus on specific UML diagram types, though a recent tablet-based tool [15] supports customizable and domain-specific notations.

Although there is limited information on these tools' success in the field, our observations present two potential difficulties to real-world deployment. First, designers employ a range of improvised and generalized notations that may be difficult to distinguish and associate with the specific notations of a fixed standard with adequate confidence. Second, designers rely on contextual information and on alternate communication mediums, not available to sketch recognizers, to complement and help interpret the improvised notations.

While automatic recognition is useful, it appears oriented primarily towards the creation of archival-quality artifacts from the meeting. What other needs do teams have in order to come up with the design in the first place?

Our earlier study of the design environment [10] showed that teams struggle to maintain a working set of visible artifacts within the limited working space, and invest significant effort in finding and retrieving diagrams after they were removed from that set. The observations in this paper also demonstrated that teams tend to increase the physical locality of related information, placing related information adjacently or embedding canvases within one another. The costs associated with physical mediums prevented them from doing so in all applicable situations, especially for short references, and they often needed to switch attention between different areas of the workspace.

Large electronic sketching surfaces offer the potential for alleviating these problems by creating a virtual drawing space that is much greater and more flexible than any physical canvas [12]. With appropriate interaction techniques, they can help teams rapidly identify artifacts in this space and bring them into physical proximity within the limited physical viewport. Multiscale interfaces [13] can help overcome the inherent resolution limitations of these displays. At present, we are not aware of any whiteboard based tool that implements this functionality.

These devices can also be configured to track and log all drawing activities. This can be used to offer an undo functionality, which may lure designers away from "drawing in the air" and leave records of the removed information, typically examples. This record could also be used to provide temporal information that can help in the interpretation of diagrams.

Large displays may also eventually play a role in meeting the increasingly important challenge of globally-distributed software design. Distributed work in general, however, presents unique challenges, heavily studied in the literature, which arise from from the loss of physical affordances. Elsewhere [10], we discuss additional challenges specific to distributed OOD, based on our preliminary study, and outline additional requirements for supporting the design environment under these settings with respect to large displays.

8.2 Preserving context

While most current work and some of our recommendations focus on on the exciting potential of large displays, we must not lock ourselves to that technology. A long time may pass before such devices are ubiquitous. Even if the technology was available in specially equipped meeting rooms, many design collaborations are spontaneous, short-lived, and take place in locations such as personal offices [6]. What implications, then, does this study have for designers working in everyday settings?

Perhaps the most important contribution of our findings is in highlighting the critical role that contextual cues, memory, and alternate communication mediums play in the design process. This information, important during the session, is particularly crucial for a subsequent understanding of the designs and their rationale. At present, casual observation suggests that most teams do little to preserve this information, and at best preserve only the final diagrams after the meeting has ended. However, it is not always possible to foresee in advance what artifacts will be important in the future, and in some cases this will only be recognized after several reproductions [6].

Therefore, the "take home message" for all designers is to try and preserve as much of this information as possible. We believe that such investment may often be preferable to attempts to create a final and aesthetic version of the artifacts during the meeting The question, of course, is how to best capture this information.

In our efforts to understand the diagrams, we made use of various information sources available to us. In terms of visual information, perhaps the most useful resource, and also the one easiest to obtain, was the collection of photos taken during the session, which captured intermediate states of the diagrams and their spatial location in relation to other artifacts. These helped us understand the evolution of each diagram and of the design as a whole, as we could compare diagrams over time. In some cases, they revealed important material that was later erased. With the ubiquity of digital cameras, most designers should be able to take these photos even in spontaneous meetings.

Unfortunately, unless photos are taken frequently, important details may be lost, including concurrent work on multiple diagrams, short references, or examples that are quickly erased. In addition, photos cannot capture gestures, "drawing in the air", and the glances that establish reference. This information is particularly important if a voice recording or transcript of the meeting is captured, since many verbal references rely on a specific visual context, such as what the designer has just looked at.

In our study, only the video stream captured enough of this evidence, due to its continuous nature. At present, video may be the most comprehensive means of capturing contextual details from collocated collaboration and may therefore be the safest approach for teams seeking to minimize the risk of losing critical information. However, although its effectiveness for requirements engineering has recently been demonstrated [7], the use of video in software design remains controversial.

One obvious concern is the feasibility of recording all collaborative situations. However, since the goal of the video is primarily to capture gestures and timing information and offer context to verbal interaction, even low-fidelity streams are sufficient. With decreasing storage costs and the availability of webcams and video-capable digital cameras, we believe this footage can be captured inexpensively even in spontaneous meetings.

A more fundamental concern regarding video is the challenge of locating information within such an unstructured stream. In our experience, the random access and rapid skimming capabilities afforded by digital video are extremely helpful in pinpointing relevant sections efficiently without watching the entire stream. Video can thus be considered as a form of "insurance" in case specific information may be sought in the future.

In the future, advances in video analysis, or perhaps correlated data from instrumented electronic whiteboards, may provide structure to this stream and offer new possibilities such as the creation of useful summaries or aggregations of context. For example, one could imagine a tool that could help supply useful context for a confusing notation by visually summarizing the set of actions that occurred immediately prior to creation of the notation, or by highlighting anything edited in close temporal proximity to the use of the notation. Based on our results, it seems that finding a rich set of techniques for capturing and displaying contextual information is a potentially very rich research area.

We realize that some teams will not be able to use technological means to capture contextual information. To these designers we can only suggest that they remain constantly aware of the future interpretability and traceability of their work, and try and preserve information in the diagrams whenever possible. This does not have to mean an explicit and costly documentation effort; it could be as simple as annotating some artifacts in the short pauses before switching to another area.

9. Conclusions and future research

In this paper we presented observations that significantly improve our understanding of the visual and physical manifestations of collocated collaborative OOD. Our findings demonstrated the tendency to choose representations and arrange visual artifacts to accommodate immediate needs, at a cost for the subsequent use of the artifacts. This highlighted the importance of supporting teams in managing artifacts and in preserving pertinent contextual information. Further study is necessary to ascertain whether our results generalize to real world design settings. The network of implicit depen-

dencies between artifacts and their impact on the design also appears to merit further detailed study.

Our current work focuses on finding means to preserve contextual information from design and development activities which take place over electronic platforms since these can be monitored with existing technologies. We are investigating the potential use of *episodic memory* [25] as a metaphor for representing accounts of these activities.

Acknowledgments

We would like to thank the organizers of the ACM *DesignFest* events at OOPSLA 2004, 2005 and 2006 for their assistance in conducting this research, and the numerous participants for allowing us to observe and record their work.

The authors gratefully acknowledge support by NSF grant IIS-0414698, as well as support from the Software Industry Center and its sponsors, particularly the Alfred P. Sloan Foundation as well as Accenture Technology Labs. The first author was supported by an IBM PhD Fellowship.

References

[1] S. W. Ambler. *The Object Primer - Agile Model-Driven Development with UML 2.0*. Cambridge University Press, 2003.

[2] A. Black. Visible planning on paper and on screen: The impact of working medium on decision-making by novice graphic designer. *Behaviour and Information Technology*, 9(4):283–296, 1990.

[3] N. Boulila. Group support for distributed collaborative concurrent software modeling. In *ASE '04*, pages 422–425.

[4] L. C. Briand, Y. Labiche, M. D. Penta, and H. D. Yan-Bondoc. An experimental investigation of formality in UML-based development. *IEEE Transactions on Software Engineering*, 31(10):833–849, 2005.

[5] Q. Chen, J. Grundy, and J. Hosking. An e-whiteboard application to support early design-stage sketching of UML diagrams. In *IEEE Conference on Human-Centric Computing (HCC'03)*, 2003.

[6] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko. Let's go to the whiteboard: how and why software developers use drawings. In *CHI '07*, pages 557–566.

[7] O. Creighton, M. Ott, and B. Bruegge. Software cinema-video-based requirements engineering. In *RE '06*, pages 106–115.

[8] C. H. Damm, K. M. Hansen, and M. Thomsen. Tool support for cooperative object-oriented design: gesture based modelling on an electronic whiteboard. In *CHI '00*, pages 518–525.

[9] C. H. Damm, K. M. Hansen, M. Thomsen, and M. Tyrsted. Supporting several levels of restriction in the UML. In *UML '00*, LNCS 2844, pages 396–409. Springer, 2000.

[10] U. Dekel. Supporting distributed software design meetings: what can we learn from co-located meetings? In *Workshop on Human and Social Factors of Software Engineering (HSSE) at ICSE'05*, *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, 2005.

[11] ACM DesignFest homepage. `http://designfest.acm.org`.

[12] S. Elrod et al. Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. In *CHI '92*, pages 599–607.

[13] G. W. Furnas and B. B. Bederson. Space-scale diagrams: understanding multiscale interfaces. In *CHI '95*, pages 234–241.

[14] J. Gil and S. Kent. Three dimensional software modelling. In *ICSE '98*, pages 105–114.

[15] J. Grundy and J. Hosking. Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. In *ICSE '07*, pages 282–291.

[16] J. D. Herbsleb, H. A. Klein, G. Olson, H. Brunner, J. Olson, and J. Harding. Object-oriented analysis and design in software project teams. *Human-Computer Interaction*, 10(2):249–292, 1995.

[17] S. R. Klemmer et al. The designers' outpost: a tangible interface for collaborative web site. In *UIST '01*, pages 1–10.

[18] A. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *ICSE '07*, pages 344–353.

[19] C. F. J. Lange and M. R. V. Chaudron. Effects of defects in UML models: an experimental investigation. In *ICSE '06*, pages 401–411.

[20] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *ICSE '06*, pages 492–501.

[21] J. Lin, M. W. Newman, J. I. Hong, and J. A. Landay. Denim: finding a tighter fit between tools and practice for web site design. In *CHI '00*, pages 510–517.

[22] A. Mehra, J. Grundy, and J. Hosking. Supporting collaborative software design with a plug-in, web services-based architecture. In *Workshop on Directions in Software Engineering Environments (WoDiSEE) at ICSE '04*.

[23] Object Management Group. UML 2.0 specification.

[24] B. Plimmer and M. Apperley. Interacting with sketched interface designs: an evaluation study. In *CHI '04*, pages 1337–1340.

[25] D. L. Schacter and E. Tulving. What are the memory systems of 1994? In D. L. Schacter and E. Tulving, editors, *Memory Systems*, pages 2–38. MIT Press, Cambridge, MA, 1994.

[26] S. Tilley and S. Huang. A qualitative assessment of the efficacy of UML diagrams as a form of graphical documentation in aiding program understanding. In *SIGDOC '03*, pages 184–191.

[27] W. Visser. Designing as construction of representations: A dynamic viewpoint in cognitive design research. *Human-Computer Interaction*, 21(1):103–152, 2006.

[28] J. Wu, T. Graham, and P. Smith. A study of collaboration in software design. In *2003 International Symposium on Empirical Software Engineering (ISESE '03)*. IEEE Computer Society, 2003.

[29] J. Wu and T. C. N. Graham. The software design board: A tool supporting workstyle transitions in collaborative software design. In LNCS 2844, pages 92–106. Springer, 2004.