CEL – Touching Software Modeling in Essence

Remo Lemma, Michele Lanza, Andrea Mocci
REVEAL @ Faculty of Informatics - University of Lugano, Switzerland

***Abstract*—Understanding a problem domain is a fundamental prerequisite for good software design. In object-oriented systems design, modeling is the fundamental first phase that focuses on identifying core concepts and their relations. How to properly support modeling is still an open problem, and existing approaches and tools can be very different in nature. On the one hand, lightweight ones, such as pen & paper/whiteboard or CRC cards, are informal and support well the creative aspects of modeling, but produce artifacts that are difficult to store, process and reuse as documentation. On the other hand, more constrained and semi-formal ones, like UML, produce storable and processable structured artifacts with defined semantics, but this comes at the expense of creativity.**

We believe there exists a middle ground to investigate that maximizes the good of both worlds, that is, by supporting software modeling closer to its essence, with minimal constraints on the developer's creativity and still producing reusable structured artifacts. We also claim that modeling can be best treated by using the emerging technology of touch-based tablets. We present a novel gesture-based modeling approach based on a minimal set of constructs, and CEL, an iPad application, for rapidly creating, manipulating, and storing language agnostic object-oriented software models, which can be exported as skeleton source code in any language of choice. We assess our approach through a controlled qualitative study.

I. INTRODUCTION

Software design is essentially modeling [1], a fundamental step in the software development process [2]. Modeling consists of the identification of the core concepts of the system to be built and the relationship between them. The produced model is a simplified representation of reality, which only includes information strictly serving the purpose at hand [3]. When performed incorrectly, modeling negatively influences the quality of the resulting system [4].

Like any other software development activity, modeling needs support of methodologies and tools. Different approaches to support modeling can be significantly different in nature, for example by considering their notation or the modeling features and constraints they impose on the users. Among the possible classifications, one can identify informal and semi-formal approaches, each one with advantages and drawbacks.

Informal modeling, done on top of one's head, on paper, on a whiteboard, or using methodologies such as CRC Cards [5], [6], sets few limits to the creative process. This produces unconstrained or minimally constrained artifacts (*e.g.,* drawings, CRC cards). These artifacts do not have a precise semantics, and moreover they are difficult to process, store, share, and maintain. Maintenance of design models is crucial, since design drift –a root cause of software aging [7]– is unavoidable in real-world systems [8].

A more formalized means of modeling is instead expressed using diagrammatic visual notations such as UML, the unified modeling language [9]. This type of modeling is mostly performed with UML editors, such as ArgoUML or UML Lab[1], which support knowledge sharing and reduce maintenance problems. Compared to artifacts produced in informal modeling, UML has a heavy notation, with a highly structured and semi-formalized specification. Because of that, we claim that UML editors inhibit the creative nature of modeling: One spends more energy in reproducing the right steps for creating correct diagrams rather than in exploring possible design alternatives.

We believe that the essence of modeling lies in between these two classes. A trade-off approach is needed, one that empowers developers to rapidly craft persistent and maintainable models, whose structure is adequate to effectively support modeling, and simple enough to stay lightweight and avoid negatively affecting creativity.

Touch-based tablet computers (such as the iPad[2]) are a middle ground that can combine the best of both semi-formal and informal modeling: They stimulate a playful environment similar to pen & paper/whiteboard, their portability enables modeling in any setting and environment, and their computational power supports processing, storage, and maintenance of the produced models.

Instead of using tablet computers as mobile whiteboards, or porting existing UML editors to touch-based devices, we present CEL, an iPad application, based on a minimal matrix-based visual language, to create and manipulate object-oriented models. CEL uses the available touch-and-gesture-based means to design software models, which can be stored, exported, and used to generate skeleton source code in any language of choice. The contributions are:

1) A novel object-oriented modeling methodology, based on a minimal set of constructs, to capture the essence of modeling.
2) CEL, a gesture-based iPad application to create, store and manipulate language-agnostic models, exportable as skeleton code. CEL uses a custom visual metaphor designed to support the creative modeling process.
3) A controlled qualitative study of our modeling methodology and of our tool.

Structure of the paper. In Section II we present our approach with its core elements, we illustrate CEL's matrix-based visual metaphor and its semantic zoomable interface, and we explain CEL's gesture-based interactions. In Section III we assess CEL through a controlled qualitative study. In Section IV, we discuss the closely related work and the projects that inspired our work. In Section V we take a critical stance towards our own work. Finally, in Section VI we summarize our work and reflect on the future of our research.

[1] http://argouml.tigris.org/ http://www.uml-lab.com/
[2] http://www.apple.com/ipad/

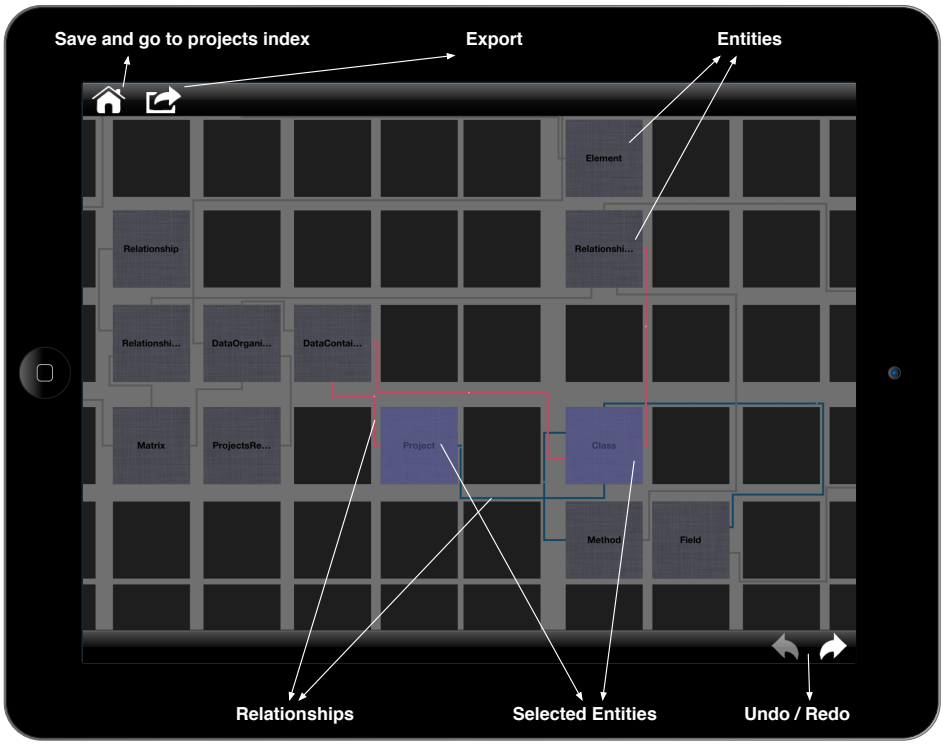


Fig. 1: CEL modeled in CEL on the iPad.

II. MODELING WITH CEL

Figure 1 depicts CEL, the iPad application [10] we developed to support our modeling methodology; this paper focuses on the conceptual design and vision of CEL. It is freely available at `http://cel.inf.usi.ch`. CEL supports the essential means to model systems, providing a visual vocabulary to create and manipulate models composed of interconnected entities, which in turn can include other entities. CEL lacks language-specific features and produces language agnostic models. Interactions are performed using gestures, minimizing keyboard- and button-based interactions. The intensive use of gestures introduces a learning phase, after which the user becomes proficient in interacting with the model under construction.

Figure 1 shows CEL modeled in CEL itself. The main view of CEL depicts the model as a set of interconnected matrix cells. The model elements are laid out automatically. To mitigate the problem of the small screen size and of the limited amount of information that can be displayed, CEL supports semantic zooming. Users can act on the matrix by instantiating new entities or relationships, manipulating the existing ones, and create a selection to focus on a group of elements to act on it.

Intermezzo. The rest of this section will explain the philosophy and core elements of CEL (Section II-A), its zoomable interface (Section II-B), the user interaction (Section II-C), and skeleton code generation (Section II-D).

Before detailing CEL, we suggest the reader to look at a video we have put together to illustrate it in action. We argue that by first watching the video all subsequent arguments will be easier to understand and assimilate. The video is located at `http://youtu.be/R838PtRuGts`.

A. CEL *Philosophy and Core Elements*

CEL is a novel methodology that tries to capture the *essence* of object-oriented modeling: CEL aims at an equilibrium between the offered modeling features and the expressiveness implicitly required by modeling tasks. In this sense, CEL offers a minimal class of entities and relationships between them, structured in a matrix-based layout.

Matrix. While modeling with a tool, users are often forced to perform operations like layout adjustments that are not related to the design process. To mitigate this problem we developed a dedicated matrix-based visual metaphor. As depicted in Figure 1, model entities are organized in a matrix separated by a grid: Its rationale is to create a simple, minimalistic interface. It introduces a constrained layout, as opposed to UML editors and visual IDEs which use free layouts, allowing users to place entities wherever they like. Free layouts do come with a price: Without positioning and size constraints, the user has to personally care about the layout, or the application has to integrate a layout engine. While automatic layouts, or techniques like snap-to-grid, seem the most practical solution, they may require the use of non-trivial algorithms such as corner stitching [11], with often unsatisfactory results [12]. The matrix-based approach constrains the user's freedom, but users can still freely place entities inside any of the cells, providing an intrinsic order and a uniform visual appearance. Moreover, adding/deleting/moving elements does not affect the positioning and the visual appearance of the other entities and does not introduce additional visual noise.

Entities. CEL's main elements are classes. Each named cell of the main matrix is a class, which can contain behavioral

entities (*i.e.,* methods) and state entities (*i.e.,* fields), represented as embedded cells.[3] We believe these few entities suffice to model the essence of a system. Other entities (*e.g.,* abstract classes, interfaces, packages, *etc.*) are useful in more advanced stages of system design. We do not deem them significant for the first stages of conceptual modeling, where one wants to deal only with a minimal set of basic concepts, and where more specialized entities would simply overcomplicate the model. Figure 2 shows how CEL depicts each entity type.

(a) Class (b) Method (c) Field

Fig. 2: Entities in CEL. Bottom-right: selected entities.

The color indicates the kind of entity being manipulated. We opted for simplicity: a mildly patterned square with a label. As claimed by Mullet *et al.*, elegant solutions can be achieved with an absolute minimum of components [15]. Figure 2 also shows how selected entities look. The blue overlay mitigates the visual difference among entities, but exploits color similarity, which overcomes the effect of proximity from a cognitive point of view [16] and makes selection appear as one single element.

Typing information for methods and fields are optional and can be used to refine important entities and for which additional details should be specified. When creating methods, (optionally typed) parameters can be specified in the signature using a JAVA-like notation (*e.g.,* `addElement(Element element)`). When adding an entity to a class, CEL checks whether it has parameters or `()` at the end of its name and, if so, it creates a method, otherwise a field. Parameters are encapsulated inside the method and can be managed in a separate, list-based, visualization.

Relationships. CEL relationships can be instantiated at the same level of abstraction (*i.e.,* classes with classes, methods and fields with methods and fields). All relationships are directed and can represent either inheritance or a generic connection.

Inheritance enables modeling of hierarchies that, when used in conjunction with subtyping, can give a better understanding of a software system domain, and is also valuable to refine entities. CEL supports multiple inheritance, often excluded in modern object-oriented languages due to the conceptual difficulties it introduces [17]. However, this feature can be used to model multiple roles of the same entity (*e.g.,* a PhD student is a university employee and a student). Inheritance relationships are antisymmetric (*i.e.,* if class A inherits from B, B cannot inherit from A) and they cannot create loops. The direction indicates that the source of the relationship is a subclass of the target. Since CEL supports only a single class type, it does not differentiate between different kinds of inheritance, as in UML realization and generalization.

UML also offers other types of relationships (*e.g.,* association, composition). This distinction often indicates implementation constraints, or specialized entity relationships. We argue it unnecessarily complicates the early stages of modeling: Thus, we treat all relationships other than inheritance as generic. The modeling role of a generic relationship is to express a strong link between two entities (*e.g.,* a class calling a method of another class), or to signal a relevant conceptual relationship (*i.e.,* two classes are related, but the designer has not yet decided how). Therefore, as opposed to inheritance relationships, generic relationships are unrestricted. Constructing models with generic relationships leaves more freedom in the implementation phase, reducing the chances of source code drifting apart from the model. To display relationships and to avoid edges crossing entities, we exploit the matrix grid (Figure 3). Relationships are always visible. The difference between inheritance and generic relationships is shown only for selected entities (Figure 3).

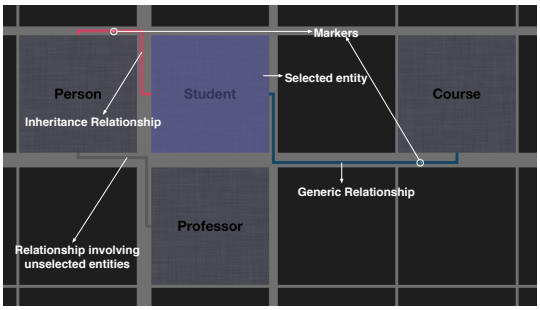


Fig. 3: CEL depicting relationships for a selected entity.

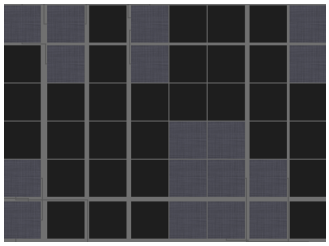
When related entities are not selected, links are displayed with a uniform light color. When selected, inheritance relationships are colored in pink and generic relationships in blue. Displaying details only for the selected entities reduces visual clutter, presenting the details on the entities in focus. The direction of relationships is visible when the user focuses on entities: Once they are selected, a marker repeatedly moves along their lines, from source to target, to show the direction. Paths are computed using Dijkstra's algorithm [18]. Figure 4 shows how grid channels grow according to how many relationships pass through the channel.

(a) 2 relationships (b) 3 relationships

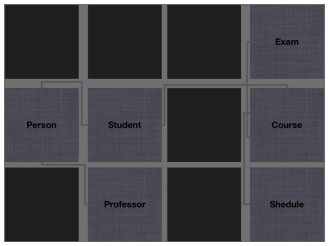
Fig. 4: Channel size depends on the number of relationships.

Enlarging the grid channels has two main benefits: (1) the paths of all relationships are always intelligible, and (2) users can spot the entities involved in many relationships, as they have larger channels around them. Since enlarging the channels might introduce visual noise when a user is not interested in relationships, CEL adapts the channel width only to the relationships that are currently displayed. This avoids situations where large channels are rendered, but no relationship is actually displayed. We tuned the weight calculation of the Dijkstra algorithm to limit the growth of the channels: The trade-off is between channel width and short paths.

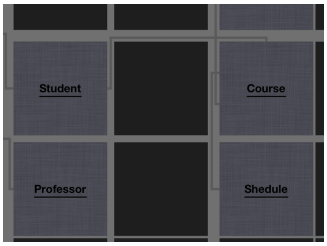
[3] Another option is to treat fields and methods as a single entity, as in Self [13]. We chose to adopt the original idea of OOP, as described by Riel [14].



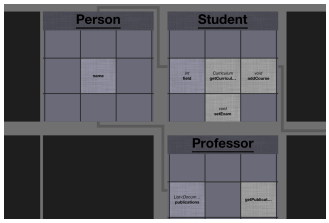
(a) Birds-Eye



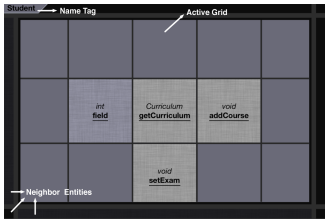
(b) Overview



(c) Edit



(d) Details



(e) Internals

Fig. 5: The five semantic levels supported by CEL.

B. The Zoomable Interface

Applications for tablets must deal with a small screen size. CEL overcomes this issue by treating the matrix as a *semantic zoomable interface*. When users zoom in, they progressively unveil details about entities, as well as new operations and actions. We opted for a semantic zoom because we argue that when users zoom out considerably, they focus on obtaining the "big picture" of the system, not the details. Furthermore, if we would allow users to perform complex operations on small entities, this might pose the risk to induce errors, as the entities would be difficult to distinguish and manipulate. Overall, we follow the principle that the quantity of information displayed has to be proportional to the available space, avoiding the introduction of visual clutter, one of the most common defects affecting modern user interfaces [15].

CEL supports five types of semantic levels (see Figure 5):

(a) *Birds-Eye:* Entities appear like in Figure 5a, where the names are not displayed. The aim is to provide users with the "big picture" of the model. Through the grid channels, users deduce which entities are involved in more relationships and see how the elements have been organized in the matrix. At this level, users can only pan and zoom the matrix.

(b) *Overview:* CEL shows entities names and provides the following new operations: context menus, creation/deletion of entities and relationships, selection creation and manipulation and the possibility to jump into the *internals* semantic level.

(c) *Edit:* This level, which is denoted by the visual cue of underlining the label, enables complex gesture-based interactions that need a considerable precision. For example, it is possible to directly edit the entity name using gestures.

(d) *Details:* At this zoom level an entity reveals its composite structure: The embedded content (*i.e.,* methods and fields for classes, and parameters for methods) becomes visible. Figure 5d shows an example in which the internals of three classes are displayed. This view shows a close overview of what is inside an entity without having to zoom in completely (thus losing the context). The internal content is read-only, and users can only inspect the internals by panning and zooming into the inner view.

(e) *Internals:* This level is accessed when the user explicitly zooms into an entity by double tapping instead of using the standard zoom gesture. The top left corner features a menu tag displaying the name of the zoomed entity. Also the outer neighbors are visualized to preserve the context. The concept of the zoomable interface is reapplied, and the five semantic zoom levels can be again exploited.

C. Interaction

Users should be able to focus only on the current task without any major interruption or disturbance [19]. To follow Raskin's guideline, we adopted two main design rules:

1. No confirmation dialogs. These messages plague users with tedious questions, eliciting often the same answers. This undermines their utility: The notification that an irreversible operation is going to be performed. According to Raskin, "Any confirmation step that elicits a fixed response soon becomes useless" [19]. To create a confirmation-free UI, we introduced a full undo/redo system that captures any relevant event.

2. No waiting times. Waiting time is wasted time, introducing frustration and damaging users' attention. Waiting times are related to costly operations (in terms of time and/or resources) or to locked resources (*i.e.,* resources which cannot be concurrently accessed by others). We designed CEL to lock as few resources as possible, allowing users to continue to interact with the entire application except for locked resources (*e.g.,* a project being saved), and all the costly operations are also deferred to separate threads to cut waiting periods.

Touch-based tablet computers do not come with pointing devices such as the mouse in standard desktop computers (unless one attaches an external device). Moreover, the digital keyboard needs training to be used proficiently. Widget-based UIs (*e.g.,* toolbars and buttons) also have significant drawbacks (*e.g.,* toolbars often have small icons, thus degrading usability) and it is hard to create fully interactive applications using only such means. We decided to minimize widget- and keyboard-based interaction and base CEL's interactions on the typical input technology of touch-based devices: *Gestures*.

Gestures are movements/actions captured on a (multi-)touch sensing surface. One can create complex gestures, or rely on simple standard gestures provided by the operating systems running on touch-based devices (*e.g.,* tap, pinch, *etc.*). Gestures are a powerful mean to create interactive applications, but the more gestures are employed, the more the usability of an application depends on them, especially on the ability of users

in utilizing them. Even simple gestures can become problematic for people not used to touch-based interfaces. This interaction technique introduces a learning phase, necessary to become familiar with these new devices, to get used to the sensitivity of the gestures, and to acquire a sufficient experience level.

In mapping gestures and operations we minimized the use of modes [19]. Modes arise from the use of the same gesture for different purposes, depending on the state of the system. However, the interaction capabilities of touch-based devices are limited, and gesture reuse is unavoidable. We adopted Norman's rule that if modes have to be used, errors can be limited with clear feedback on the state of the system [20].

Furthermore, to enhance usability, we assigned gestures involving more than 2 fingers to actions implying rapid and simple movements. CEL's gestures are summarized in Figure 6.

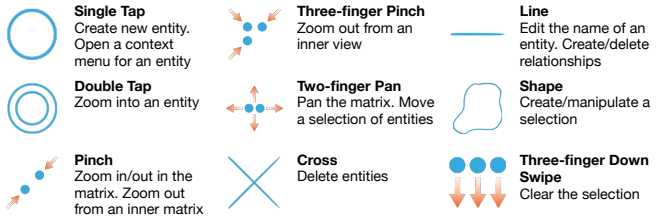


Fig. 6: Summary of the gestures used in CEL

Single Tap: This gesture indicates that the user wants to *do something*. We have assigned it the fundamental role of instantiating the creation of a new entity when an empty cell in the matrix is tapped. If instead, one taps on an existing entity, its context menu opens up, which contains actions which involve non-standard gestures (*e.g.,* delete or rename action); it is intended for users unfamiliar with gesture-based interaction.

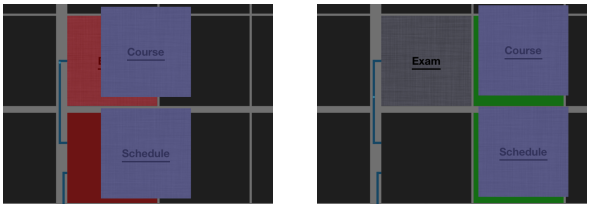
Double Tap: We keep this gesture consistent with its behavior in iOS: bound to zoom-in/zoom-out. The double tap let users zoom into an entity, showing the *internals* zoom level.

Pinch: The pinch gesture is well-known to most touch-based device users. It is used to perform zoom operations and we have reused the same concept also in CEL . Additionally, we have added a secondary feature to this gesture, allowing users to zoom out from the internals of an entity when the minimum zoom level is surpassed.

Three-finger Pinch: This custom gesture allows to rapidly zoom out from entities without the need of zooming out long enough to surpass the minimum zoom factor.

Two-finger Pan: Panning is associated with the concept of movement. We adopted a two-finger gesture to move our view, discarding the usual simpler one-finger pan, since it would conflict with other gestures (*i.e.,* shape, cross and line gestures).

If the user starts a two-finger pan gesture without entities selected, the entire matrix is panned, otherwise the user moves the selected entities. This can be considered a mode, yet the selected entities are clearly highlighted (*i.e.,* we provide visual feedback) and the overall action associated to the gesture, *i.e.,* moving, remains consistent. When panning a selection, as shown in Figure 7, we add an overlay as a visual cue to signal whether the movement is legal (green) or illegal (red).



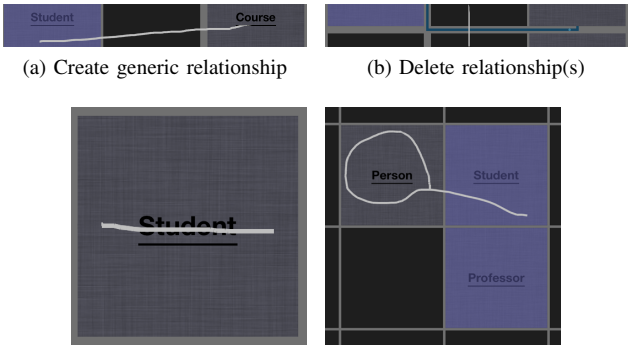
(a) Invalid (b) Valid

Fig. 7: Move of a selection of entities.

For legal movements, entities are placed at the new position and all the connected relationships are automatically rewired. For illegal movements, entities return to their original site.

The matrix also features auto-pan, that can be exploited while panning the matrix or while repositioning selected entities. To activate auto-pan, users put their fingers at the borders of the screen, and the application will automatically start to pan the matrix in the desired direction. Multiple directions can be combined at once (*e.g.,* up and left, down and right, *etc.*).

Cross: This is a custom gesture to delete entities. A cross drawn over multiple entities deletes them all. In iOS devices the deletion is usually accomplished with a (slow) combination of gestures and buttons. We claim that a cross gesture can be naturally mapped to the concept of erasing content.



(a) Create generic relationship (b) Delete relationship(s)

(c) Create inheritance (d) Edit entity name

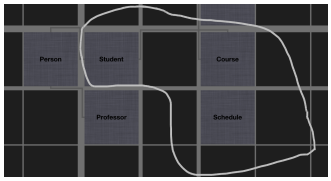
Fig. 8: Line gesture operations.

Line: The line gesture accomplishes different tasks (summarized in Figure 8): handle relationships (*i.e.,* creation and deletion) and allow the renaming of entities.

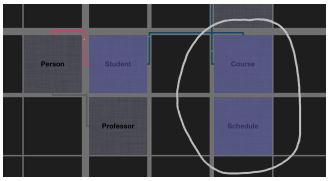
To create a relationship the user draws a line from a selection (*i.e.,* the source elements) to a target entity. Each element in the selection instantiates a relationship to the target element. To distinguish between the creation of inheritance and generic relationships, we designed two ad-hoc line-based gestures, shown in Figure 8a and Figure 8d. The inheritance creation gesture is a straight line with a closed shape at the end (*i.e.,* like a lasso) which encapsulates the center of the targeted entity. The generic relationships creation gesture is a simple line. They are similar enough to recall the same concept (*i.e.,* relationship instantiation), yet they are still easily distinguishable.

To delete relationships, we adopt a similar gesture to the one used to delete entities (*i.e.,* the cross). The selected approach is illustrated in Figure 8b. Since relationships are drawn as polygonal chains of straight lines, the line drawn with the gesture intersects the visualization forming a cross, as desired. Since details on relationships are available only for selected entities, to delete a relationship one of the involved elements has to be selected. To avoid conflicts we give priority to relationship creation over deletion if both gestures activate at the same time.

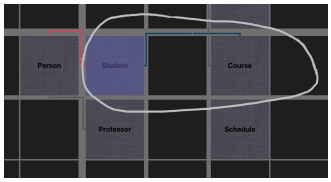
The last interaction type that involves the line gesture is the renaming action. As depicted in Figure 8c, by striking the label of an entity, one can then edit its name. Because the scope of this gesture is limited to the rectangle of an entity, it does not conflict with the other line gesture usages.



(a) Add entries to the selection



(b) Deselect entities of the selection



(c) Update the selection

Fig. 9: Shape gesture usage.

Shape: The shape gesture is probably the most powerful gesture we created. The possibility to draw any kind of closed shape to activate this gesture mitigates problems related to proficiency and ability in drawing: No extreme precision is needed. This gesture is the last one to be activated: If no other gesture is compatible with the movement, the shape is automatically closed, making this gesture even faster to perform (*i.e.,* users do not have to manually close the shape).

Each shape gesture acts on the global selection that is assigned to each matrix view. The two trivial cases are shown in Figure 9a and Figure 9b. In the former, the user can add deselected entities to the selection. In the latter, the user deselects the selected entities that are contained in the shape.

Figure 9c shows a more complicated and controversial case, in which the shape drawn by the user surrounds both selected and unselected entities. We had two options to handle this case: Implementing the shape as a toggle (*i.e.,* invert the selection) or treating the entities contained in the shape as a single group. We opted for the second strategy. Therefore, as we treat the elements included in the shape as a single group, if the group is all selected it gets deselected, otherwise all the entities are selected. In the example presented in Figure 9c, the *Course* entity is added to the selection already including *Student*.

Three-finger Down Swipe: This gesture clears the entire selection (*i.e.,* deselects all the entities). This gesture can be performed rapidly and metaphorically replicates a *sweep*.

D. Skeleton Code Generation

To exploit models after they are produced, CEL features a code generation engine that can be employed as a first guideline to implement the actual system. To export the skeleton code of a CEL model, we devised language templates (to be filled with language-specific constructs), which define the rules to export the skeleton source in any language of choice. CEL natively supports three languages: C++, Java and Objective-C. Listing 1 shows an example of a language template.

Listing 1: Java method export definition

```
"method": {
  "export": "\n\t{RETURN_TYPE} {NAME} ({PARAMETER_LIST}) { }",
  "defaultReturnType": "void",
  "parameterList": {
    "*": "{PARAMETER_TYPE} {PARAMETER_NAME}",
    "separator": ", ",      "defaultType": "Object"    } }
```

Each entity has unique tags (encapsulated between {}), which can be used to refer to entity parameters, like the return type, or the name and parameter list of the method. Moreover, for each entity there is a set of configurable parameters to specify language-specific aspects, like the default type for a method with no return type, or the formal parameter separator. Templates can also contain stylistic components (*e.g.,* \n, \t), which can be used to create readable output.

III. Evaluation: Controlled Qualitative Study

Given the nature of CEL, a comparative evaluation with existing work is not feasible, as there is no directly related work. Also, we discarded a *quantitative* comparison between CEL and a UML editor because it would be an apples and oranges comparison, since CEL does not strive to be a UML editor. We opted for a *controlled qualitative study*, since our goal is to assess the quality of our design choices and to obtain feedback on CEL's viability as a novel modeling approach. Qualitative studies attempt to interpret a phenomenon based on the explanations that people express during the study [21]. Such evaluations can be precious while studying new products to identify relevant variables to be further investigated [22]. Thanks to the richness of the data it is possible to find answers to the questions of how and why things happen, and to find out valuable latent and non-obvious issues [23]. Our study follows the process described by Wohlin *et al.* [24].

A. Planning

Context selection. Because CEL is a new product, evaluating it by solving tasks in real-world projects would have comported too many risks. We opted for a controlled environment involving students, with tasks that mimic real-world situations to gather meaningful feedback. Although this is not a comparative study, we decided to introduce different tasks which involved the use of ARGOUML, a well-known UML editor. We did not aim at proving the superiority of CEL, but we wanted to ensure that each subject would have an idea of how different these two worlds are.

Subjects. We involved both graduate and undergraduate students, to have two distinct sets of subjects: those experienced with modeling, and beginners. Thus, we applied stratified sampling [24]. Because of nearly identical sample sizes, we fixed the same number of participants for both categories.

B. Study Design

As suggested by the literature [25], we used various methodologies to gather the data[4].

Screening Questionnaire. To collect information about the participants and their experience (*e.g.,* gesture proficiency), subjects were requested to fill in a screening questionnaire.

Tasks. Subjects were asked to model two systems (one with CEL and one with ARGOUML) and to analyze two existing software models (again each one with a different tool). Thus, we had to find four different object systems, described below. The tasks were limited to 20 minutes of duration.

Modeling Tasks (A1 and A2). We provided participants a textual description of two systems S1 and S2. The goal was to model them in a sufficiently detailed manner, including internals and relationships. The participants were free to stop before the time limit, if they were convinced that the model they produced was sufficient to represent the given description.

Comprehension Tasks (B1 and B2). These tasks required the participants to analyze the software models of two systems S3 and S4. The participants were asked to give a brief description of the overall purpose of the modeled system, identify the key entities, and report the strategy they employed to find such information. The subjects could stop whenever they finished answering. During each evaluation run, we also annotated the amount of time used by each user to solve each task.

Object Systems. To avoid to produce systems which would favor CEL over ARGOUML, we chose four design exercises created by Prof. Cesare Pautasso[5] from the University of Lugano, in the context of his Master course on Software Architecture & Design. Two simple systems have been employed for the modeling tasks and two more complex ones have been used for the comprehension tasks. These latter tasks needed existing software models: We did not create the models ourselves, but produced them based on solutions proposed during the course. The models, designed in both CEL and ARGOUML, are identical in terms of entities, positions (when possible), and relationships. The four systems can be briefly described as follows:

S1: Star Bux DJ. This system models a radio service for a coffee chain. The DJ can upload new songs and create a playlist for each day. Each store retrieves the playlist, downloads the songs and plays them.

S2: ATM System. This model mimics a distributed Automated Teller Machine (ATM). Customers can withdraw from any bank. Each bank has it own account system that has to be reused. At the end of the day the ATM sends reports to any bank involved in the logged transactions.

S3: Book A Trip. The system concerns a novel interface for a legacy system used to book flights. Clients can reserve a flight, print tickets and cancel reservations. The system should also provide a new security layer and record all the history of trips for further data analysis.

S4: Public Transport. This system models a service to support people using public transport. It exploits GPS positioning to suggest routes, integrates schedules of different transport means, and takes traffic into account.

[4]The interested reader can find the whole data in [26].

[5]http://www.pautasso.info/

Debriefing Questionnaire. After the experiment a debriefing questionnaire was filled by the participants where they were asked to give feedback on different aspects of CEL, on the differences between CEL and ARGOUML, and on the experiment.

Post-experiment Interview. A short (roughly 15 minutes), semi-structured interview conducted after the experiment, to collect further feedback about the usage of CEL and to double check the answers given in the debriefing questionnaire. Example questions that guided the interview are listed below.

1. How was the overall experience with CEL? Was it difficult to use?
2. Were the gestures difficult to perform? What would you change?
3. How did you find the core concepts in tasks B1 and B2?
4. Was the zoomable interface intuitive?
5. Which problems did you encountered in modeling with CEL?
6. Do you have any other suggestion or idea to improve CEL?

As observed by Lethbridge *et al.* [27], participants are comfortable with questionnaires and interviews, yet they tend to report only facts which are relevant to them (which are potentially not the ones researchers are looking for). To avoid this, we observed the subjects while working and annotated key events, which we investigated at the end of the interview. As we borrowed the different tasks from an external source (*i.e.,* a university course) we reduced possible biases. Moreover, to minimize the chance of having two easy tasks assigned to CEL, we designed two treatments in which the tasks employing CEL and ARGOUML were switched. We used a balanced design to have the same number of participants for each treatment.

Instrumentation. At the beginning of the evaluation a tutorial of 10 minutes on the essential features of both tools was given. A handout containing all the information on the experiment and the tasks was then distributed to the subjects, who could ask questions in case of difficulties.

C. Execution & Analysis

We did a total of six runs of our study. In qualitative evaluations the quantity of data is huge [27], a problem known as "attractive nuisance" [28]. Therefore, a handful of participants may be sufficient to gather enough data, yet the findings may be difficult to generalize, while they are precious to create a base for further qualitative or quantitative studies. Qualitative evaluations employs techniques to analyze the data which usually do not rely on precise measurements to yield to conclusions [29]. We used theory generation, extracting different statements supported in multiple ways by the data. In particular, we used the constant comparison method [30], [23], in which codes are assigned to different parts of the data, emphasizing their relevance to a particular theme of interest. In a second step the data is re-analyzed to search patterns and trends. Such groups of passages can then be condensed in propositions. This method is part of "grounded theory" methods, as the theories produced are "grounded" in the data [30]. It is important when generating such theories also to confirm them. As claimed by Seaman [31], although quantitative hypothesis testing and statistical significance are often rewarded more, confirmation of theory in qualitative research has the same value. In our study we used member checking [32], a method in which the potential findings are presented to the subjects themselves, to get their support for final conclusions.

D. Results & Discussion

Of the six subjects, three had experience with modeling and three rated themselves as beginners. By analyzing the timing information (see Table I), CEL seems not inferior when creating new models or comprehending existing ones.

TABLE I: Completion time data

	CEL (mins)				ARGOUML (mins)			
Subject	**A1**	**A2**	**B1**	**B2**	**A1**	**A2**	**B1**	**B2**
P1	20	-	7	-	-	14	-	8
P2	-	14	-	14	20	-	13	-
P3	22	-	16	-	-	23	-	18
P4	-	9	-	5	10	-	7	-
P5	20	-	13	-	-	21	-	8
P6	-	24	-	7	21	-	7	-
Tot. time	62	47	36	26	51	58	27	34
Tot. tasks	109		62		109		61	

Overall, subjects were enthusiastic about CEL. They claimed that the minimal set of modeling constructs available in CEL favors rapid creation of new models, without dealing with tedious details. This minimalism is also "good for fast understanding" (cit. P6). All subjects agreed that CEL's zoomable interface is effective in rapidly jumping from a high-level system overview to the details of a single entity. This feature is essential to rapidly analyze an unknown model and the different visual cues (*e.g.,* enlarged channels) considerably push the comprehension power of CEL. Overall, the user interface was strongly appreciated, especially the matrix: all subjects found the automatic management of all layout-related aspects remarkable, allowing users to concentrate on actual modeling. P1, P2 and P5 found our custom gestures intuitive, while P3, P4 and P6 found them less so, yet easy to learn.

Insights from the Debriefing. During the design of our study we also came up with the following research questions:

RQ1 *How do participants adapt to different modeling techniques which provide distinct sets of elements to create software models?*

RQ2 *Which strategy is commonly employed by users to identify core concepts of unknown software models and how do they adapt to different visual cues?*

RQ1 pertains to the two modeling tasks A1 and A2. With ARGOUML only three subjects employed different types of relationships, and only sporadically; everyone mostly used associations. Nobody used abstract classes nor interfaces, but this may be a direct consequence of the nature of the systems to be modeled. With CEL we observed that the subjects used all the elements it provides, especially generic relationships. All subjects were satisfied by the minimalism of CEL, stating that the limited number of entities and relationships were exactly what they needed. We noticed that people were flexible enough to switch contexts among different modeling methodologies. However, techniques such as UML offer many specialized modeling concepts, and users employ only few of them if they are not forced to. Such desire for a simple modeling technique has been confirmed by our participants: P5 stated related to CEL that *"Few relationships and entities are sufficient and help me in focusing only on modeling."*

In summary, it seems that users are able to adapt to different modeling methodologies, but they seek simpler solutions with respect to mainstream, digital, modeling methodologies. Although ARGOUML provides many more elements than CEL, users perceived both approaches as functionally equivalent, and actually preferred our CEL's simpler methodology (as suggested by Ockham's Razor design principle [33]).

RQ2 involves the strategies employed by the subjects to identify the core concepts of the systems in tasks B1 and B2. They can be summarized as follows:

- P1 mainly focused on entities internals, while P6 mainly targeted entities referencing many relationships. Nevertheless, they both used the information gathered while looking at the entities to understand the system. P6 refined this strategy by also looking at entities names. This approach was harder to adopt in CEL because names are not always visible. P1 exploited the zoomable interface of CEL to switch from a global view of the system to a more local one and finally to a visualization of entity internals.
- P2, P4 and P5 identified core components by exclusively considering the number of relationships. P2 used CEL highlighting to focus on some of the entities which were candidates to become key elements and observed the connections direction following the markers. P5 also refined comprehension while using CELby first looking at the width of the grid channels, then selecting and highlighting entities in the neighborhood of crossing points to observe the number of connections.
- P3 used a different technique for each tool. With CEL P3 focused on entities with many relationships, exploiting also the direction information. With ARGOUML, instead, the participant looked at fields and methods. P3 stated that this was due to the impossibility of rapidly understanding the direction of relationships in ARGOUML.

In object-oriented systems core concepts are implemented in key entities, represented by (key) classes. Subjects identified key entities exploiting relationship or internals. Relying on the name of entities as done by P6, instead, can be misleading, especially in badly designed models. Understanding a software system is a complex operation and the perfect method to find the key entities does not exist. We observed that users prefer to use a rapid technique, being aware that it is not perfect. They prefer to learn about the system while analyzing some entities and, eventually, refine the search.

Overall, we learned that users employ simple, rapid, reasonable, yet not perfect techniques to find key entities of an unknown software model. They adapted without major issues to the user interface they were confronted with and exploited the available visual cues. With the exception of P5 all participants used some peculiarities of the user interface (*e.g.,* the zoomable interface of CEL or the easy access to fields and methods in ARGOUML) to enhance their basic analysis methodology. Thus, a well-designed user interface can enrich model comprehension.

Reflections. Summarizing, the feedback gathered during our evaluation is precious, despite the small number of subjects. Some suboptimal aspects of our current implementation have been remarked, yet, overall, our approach was highly appreciated (especially for its novelty) and the subjects claimed that it can be considered a valid alternative to mainstream modeling.

E. Threats to Validity

To mitigate internal validity we chose subjects who were at least beginners to modeling and with basic knowledge of UML. We also decided to not design the tasks ourselves. For a fair baseline we opted for a well-known UML editor, ArgoUML. We also set the same starting conditions (*e.g.,* fresh installation, quiet setting) for each run of the experiment. We also had to cope with external validity issues. We based the choice of our subjects on their modeling knowledge. Although this may not be the best criteria for a representative sample, the estimation of other relevant factors (*e.g.,* iPad proficiency) is difficult even for the subjects themselves. Moreover, the tasks and the object systems we used for this experiment may not reflect real-world situations. However, we introduced questions (*e.g.,* what is the overall purpose of the system?) to mimic real-world situations (*i.e.,* a developer who has to understand an unknown system). The object systems we selected are complex enough to be non-trivial for experts, yet also feasible for beginners. We avoided the experimenter effect by using reference solutions to evaluate the work done by the users. To minimize the disturbance given by the presence of the experimenter we limited our interaction with the subjects, and were unobtrusive while taking notes. Finally, since we performed a *controlled qualitative study* (as opposed to a quantitative one) the results were less crucial than the actual feedback given by the participants.

IV. Related Work

Given the amount of tools for modeling systems, which we do not compete with, we discuss only closely related work.

Lightweight modeling made digital. When confronted with design problems developers often sketch potential solutions using lightweight, informal, means such as the whiteboard [34]. Electronic whiteboards have been leveraged in Calico [35], which adds the possibility to easily switch between multiple sketches, favoring the reuse of user-defined notations. Our approach is more focused on conceiving an alternative modeling methodology. CRC cards are also used to rapidly prototype design decisions. Creww [36] alleviates the weaknesses of "low-fi" CRC sessions by aiding the process using Wii-Remotes as input devices. Creww records development sessions and helps with the storage of the generated cards in a digitalized form, yet the authors advocate the usage of UML in the last step of CRC sessions. With Cel we propose a solution using a simple visual language not based on UML.

UML made portable. UML editors are the main instrument through which semi-formal modeling is performed, and many of them have been ported to devices like tablets. Astah* UML pad[6] provides the means to construct class diagrams and to export models. DrawUML[7] allows to (concurrently) create different types of diagrams. Ma *et al.* proposed a web-based UML modeling tool with touch gestures, combining them with traditional keyboard and mouse input [37]. These applications do not compete with full-fledged desktop UML editors and should be rather considered as complementary tools. With Cel we aim to create an independent, competitive alternative.

Portable programming. Novel technologies have been experimented also in the context of IDEs. For example, YinYang [38] is a visual programming language explicitly designed to allow development on tablet computers. TouchDevelop [39], instead, introduces a scripting language for Windows Phone devices. Finally, Coffee Table [40] is an IDE built around a shared interactive desk, to share elements and project the software architecture and workflow. These approaches focus on programming, while with Cel we concentrate on modeling.

Alternative metaphors. Cel does not permit to write source code, yet we took inspiration from different research which leverage programming through the use of new metaphors. In Gaucho [41] developers write programs by directly manipulating graphical objects. Code Bubbles [42] depicts code fragments as lightweight, fully editable bubbles that can be grouped together, creating concurrently visible working sets. In Cel, we use abstraction to present a concise view of the entities of a model: A simple named rectangle following a color scheme. This is comparable to how the mentioned tools abstract away from treating source code as mere text. Code Canvas [43] proposes another innovative approach, in which the user is provided with an infinite zoomable interface where editable content and project-related information coexist. In Cel, we also make use of the canvas metaphor, enabling the developer to arrange the model in an infinite, zoomable, 2D surface.

V. Advocatus Diaboli

A number of criticisms may be raised towards Cel.

Cel is nothing new. Cel is neither the first modeling tool nor the last one, but it tackles modeling from a novel perspective. We target the initial stages of modeling, where the focus is more on laying out the essence of a system. To this aim, Cel only provides a few key features, enough to create accurate models without delving into time-consuming details.

Cel is too simple. Cel is just "a bunch of rectangles with three distinct colors to visualize entities and one single type of line depicted with two different colors to denote relationships", and that is what we aimed for. Since users can achieve non-trivial results with the few concepts that they are provided with, this is not functional poverty. Citing White, "Design is not the abundance of simplicity, it is the absence of complexity" [44].

Cel does not scale. The infinite 2D space makes it possible to extend the model, but the impossibility to rapidly jump to different portions of the matrix, combined with the difficulty of managing very large matrices, may lead to scalability problems. However, Cel has been designed for rapid modeling sessions, which should not involve very large models. Moreover, we are developing a search system and investigating ways to manage large matrices, such as "portal" cells that lead to other models.

Cel is less collaborative than a whiteboard. Cel is not aimed at replacing the collaborative experience of modeling on a whiteboard, yet we discussed how Cel adresses the drawbacks of a whiteboard, especially its informal and volatile nature.

Cel is not validated. The promising results obtained in our study must be taken with a grain of salt. Other studies have to be conducted to assess the potential of Cel, for example to measure the cognitive loads of operations performed using gestures. However, the insights gathered during this first qualitative evaluation can be the ground of further, quantitative, experiments such as [45].

[6] http://astah.net/editions/pad

[7] http://itunes.apple.com/us/app/draw-uml-for-ipad/id428468147

VI. Conclusions

We presented Cel, a novel methodology for object-oriented modeling. UML-based tools lack flexibility and simplicity, while lightweight, flexible means produce volatile output. Cel mitigates the drawbacks of both worlds while keeping their advantages. It uses a matrix-based visual metaphor and a minimal set of constructs to support modeling. It exploits semantic zooming to optimize the amount of displayed information in a small screen size. We performed a controlled study to gather feedback on our approach. From this evaluation Cel seems at least as competitive as UML-based modeling, which is promising. We also took a critical stance towards Cel, clarifying the controversial aspects. We tried to not underestimate the risk of creating an exciting but useless modeling toy. We believe there is a void middle ground between lightweight modeling such as a whiteboard, and heavyweight means like UML. Cel is a leap of faith into that void.

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science Foundation through project "HI-SEA" (no. 146734).

References

[1] R. Lee and W. Tepfenhart, *Practical object-oriented development with UML and Java*, ser. An Alan R. Apt Book Series. Prentice Hall, 2002.

[2] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*, 2nd ed. Prentice Hall, 2003.

[3] J.-M. Favre, "Foundations of model (driven) (reverse) engineering: Models - episode i: Stories of the fidus papyrus and of the solarus," in *Language Engineering for Model-Driven Software Development*, 2004.

[4] H. van Vliet, *Software Engineering - Principles and Practice*, 2nd ed. Wiley, 2000.

[5] N. M. Wilkinson, *Using CRC Cards — An Informal Approach to Object-Oriented Development*. SIGS Publications, Inc., 1995.

[6] D. Bellin and S. Simone, *The CRC Card Book*. Addison Wesley, 1997.

[7] D. Parnas, "Software aging," in *Proceedings ICSE 1994 (16th ACM/IEEE Int. Conf. on Soft. Eng.)*, 1994, pp. 279–287.

[8] G. C. Murphy, D. Notkin, and K. J. Sullivan, "Software reflexion models: Bridging the gap between design and implementation," *IEEE Trans. on Soft. Eng.*, vol. 27, no. 4, pp. 364–380, 2001.

[9] M. Fowler and K. Scott, *UML distilled - a brief guide to the Standard Object Modeling Language (2. ed.)*. Addison-Wesley-Longman, 2000.

[10] R. Lemma, M. Lanza, and F. Olivero, "Cel: Modeling everywhere," in *Proc. of ICSE 2013 (35th ACM/IEEE Int. Conf. on Soft. Eng.)*, 2013, pp. 1323–1326.

[11] J. Ousterhout, "Corner stitching: A data-structuring technique for vlsi layout tools," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 3, no. 1, pp. 87–100, 1984.

[12] B. Sharif and J. I. Maletic, "The effect of layout on the comprehension of uml class diagrams: A controlled experiment," in *Proc. of the 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE Computer Society, 2009, pp. 11–18.

[13] R. B. Smith, J. Maloney, and D. Ungar, "The self-4.0 user interface: manifesting a system-wide vision of concreteness, uniformity, and flexibility," *SIGPLAN Not.*, vol. 30, no. 10, 1995.

[14] A. Riel, *Object-Oriented Design Heuristics*. Addison-Wesley, 1996.

[15] K. Mullet and D. Sano, *Designing Visual Interfaces*. Prent. Hall, 1995.

[16] G. Rush, *Visual grouping in relation to age*, ser. Archives of psychology. Columbia university, 1937.

[17] G. B. Singh, "Single versus multiple inheritance in object oriented programming." *OOPS Messenger*, vol. 6, no. 1, pp. 30–39, 1995.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[19] J. Raskin, *The Humane Interface - New Directions for Designing Interactive Systems*. Addison-Wesley, 2000.

[20] D. A. Norman, "Design rules based on analyses of human error," *Communications of ACM*, vol. 26, no. 4, pp. 254–258, 1983.

[21] N. K. Denzin and Y. S. Lincoln, *Handbook of qualitative research*. Sage Publications, 1994.

[22] F. Shull, J. Singer, and D. I. K. Sjoberg, *Guide to Advanced Empirical Software Engineering*. Springer, 2008.

[23] M. B. Miles and A. M. Huberman, *Qualitative Data Analysis: An Expanded Sourcebook*. Sage Publications, 1994.

[24] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, 2000.

[25] R. Barbour, *Introducing Qualitative Research*. Sage, 2008.

[26] R. Lemma, "Software modeling in essence," Master Thesis, Faculty of Informatics, University of Lugano, Jun. 2012.

[27] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," *Empirical Software Engineering*, vol. 10, pp. 311–341, 2005.

[28] M. B. Miles, "Qualitative data as an attractive nuisance: The problem of analysis," *Administrative Science Quarterly*, vol. 24, pp. 590+, 1979.

[29] D. I. K. Sjoberg, T. Dyba, and M. Jorgensen, "The future of empirical methods in software engineering research," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 358–378.

[30] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*, ser. Observations (Chicago, Ill.). Aldine de Gruyter, 1967.

[31] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Trans. Softw. Eng.*, vol. 25, pp. 557–572, Jul. 1999.

[32] Y. S. Lincoln and E. G. Guba, *Naturalistic Inquiry*, ser. Sage focus editions. Sage Publications, 1985.

[33] W. Lidwell, K. Holden, and J. Butler, *Universal Principles of Design*, 2nd ed. Rockport, 2010.

[34] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," in *Proc. of CHI 2007 (ACM Conf. on Hum. Fact. in Comp. Sys.)*. ACM, 2007, pp. 557–566.

[35] N. Mangano, A. Baker, M. Dempsey, E. Navarro, and A. van der Hoek, "Software design sketching with calico," in *Proc. of ASE 2010 (25th IEEE/ACM Int. Conf. on Aut. Soft. Eng.)*. ACM, 2010, pp. 23–32.

[36] F. Bott, S. Diehl, and R. Lutz, "Creww: collaborative requirements engineering with wii-remotes," in *Proc. of ICSE 2011 (33rd ACM/IEEE Int. Conf. on Soft. Eng.)*. ACM, 2011, pp. 852–855.

[37] Z. Ma, C.-Y. Yeh, H. He, and H. Chen, "A web based uml modeling tool with touch screens," in *Proc. of the ASE 2014 (29th IEEE/ACM Int. Conf. on Aut. Soft. Eng.)*, 2014, pp. 835–838.

[38] S. McDirmid, "Coding at the speed of touch," in *Proc. of Onward! 2011*, 2011, pp. 61–76.

[39] N. Tillmann, M. Moskal, J. de Halleux, and M. Fahndrich, "Touchdevelop: programming cloud-connected mobile devices via touchscreen," in *Proc. of Onward! 2011*, 2011, pp. 49–60.

[40] J. Hardy, C. Bull, G. Kotonya, and J. Whittle, "Digitally annexing desk space for software development," in *Proc. of ICSE 2011 (33rd ACM/IEEE Int. Conf. on Soft. Eng.)*, 2011, pp. 812–815.

[41] F. Olivero, M. Lanza, and M. Lungu, "Gaucho: From integrated development environments to direct manipulation environments," in *Proc. of the 1st Int. Workshop on Flexible Modeling Tools*, 2010.

[42] A. Bragdon, R. Zeleznik, S. P. Reiss, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. L. Jr., "Code bubbles: a working set-based interface for code understanding and maintenance," in *Proc. of the 28th Int. Conf. on Human factors in computing systems*. ACM, 2010, pp. 2503–2512.

[43] R. DeLine and K. Rowan, "Code canvas: zooming towards better development environments," in *Proc. of ICSE 2010 (32nd ACM/IEEE Int. Conf. on Soft. Eng.)*, 2010, pp. 207–210.

[44] A. White, *The Elements of Graphic Design*. Allworth Press, 2002.

[45] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in *Proc. of ICSE 2011 (33rd ACM/IEEE Int. Conf. on Soft. Eng.)*, 2011, pp. 551–560.