Improving Model Usability and Utility by Layered Diagrams

Harald Störrle
QAware GmbH
Aschauer Str. 32
82153 München, Germany

ABSTRACT

Background: Large and complex diagrams are hard to understand, yet sometimes unavoidable. In other disciplines, diagrams are routinely organized into layers to tackle complexity–why not in Software Engineering?
Objective: We want to understand whether layered diagrams are a useful addition to the software engineers repertoire. If so, how are layers best used, and what features and properties should they have?
Method: First, we explore the design space. We study how layers are used in other disciplines, discuss sample layered diagrams, and derive design decisions. We implement layered diagrams in an experimental modeling tool, and provide an initial validation, exploring the usage space in a field study.
Results: Layers are intuitive to use and benefit many usage scenarios that are poorly supported by most contemporary tools.
Conclusions: Layered diagrams complement conventional diagrams and model structuring techniques. They are easy to comprehend and apply by modelers, and allow richer models. With layered diagrams, the utility and usability of existing modeling tools can be improved at little cost.

ACM Reference format:
Harald Störrle. 2018. Improving Model Usability and Utility by Layered Diagrams. In *Proceedings of 40th International Conference on Software Engineering Companion, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18 Companion),* 8 pages.
https://doi.org/10.1145/3183440.3195025

1 INTRODUCTION

In industry as well as in academia, models are primarily used to support cognition and communication [2; 29]. Thus, the visual qualities of UML, and more generally, the usability of MDE have received more and more attention in the last years [1]. We already know that poor layout quality and large diagram size negatively affect the understandability of diagrams, independent of diagram type [27]. In fact, there is a limit to the size of diagrams beyond which modelers can not be expected to effectively understand diagrams [28]. However, models *do* get very large at times [22], and every so often the practical modeler finds herself in a situation where a large and complex model must be presented in a single large, and complex diagram. In this paper, we explore one possible avenue of handling this problem: We propose to add the construct of *layers* to the modeler's repertoire.

Def.: *A layer is a set of diagram elements grouped to express some commonality and allow collective operations. A layered diagram is an partially ordered set of layers.*

It is up to the modeler to decide what commonality a layer represents. The placement of elements on layers is not part of the model as such: Layering is a purely syntactical device. Layers may be shown or hidden independently. Thus, a diagram with n layers may replace up to $2^n - 1$ plain diagrams. Also, operations like coloring or commenting may be applied to layers independently. Finally, layers may be used as inputs to semantic model operations like union, intersection, or difference, affecting all elements of selected layers.

Layering is ubiquitous in many engineering disciplines and architecture, yet all but absent in conceptual modeling in SE. We conducted a set of qualitative interviews with eight expert modelers from architecture, graphic design, mechanical engineering, and electrical engineering to find out about the usage of layers in the respective disciplines. In graphic design, the usage of layers is omnipresent; standard tools like Adobe Photoshop have had layers for decades. Graphic designers we interviewed considered layers as natural, and completely transparent to use. To them, it was difficult to understand why layers would be absent from a UML modeling tool. As one designer put it: "*You don't have that?! But why not? How do you structure your models?*" Of course, the idea of layers predates computerized tools: for centuries, architects have used stacks of semi-transparent sheets of paper to mix and match alternative designs. Today's tool support for structural engineering uses layers to separate concerns, e.g., one layer would contain the ground plan of an edifice or storey, while other layers are reserved for electrical installations, plumbing, etc. Yet other layers are used for structural integrity computations, or construction logistics. So, layers can also help distribute and coordinate work across teams and contractors.

In software engineering, conceptual models are created with two classes of tools. On the one hand, there are *Diagraming Tools* like MS PowerPoint and Visio: regular, all-purpose drawing tools, repurposed for (visual) modeling. They allow to create (more or less) any diagram, including ones that violate the syntactic and semantic consistency rules of the language used. They allow to mix separate notations, and invent new ones on the fly. PowerPoint does not provide layers as such, though many users will use the "master slide" or sequences of overlays to a similar effect, in spite of the inherent limitations. Adobe Illustrator has complex layering features, but is hard to learn and costly to acquire. MS Visio provides a unique and distinctive kind of layers very unlike the ones we shall propose below.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3195025

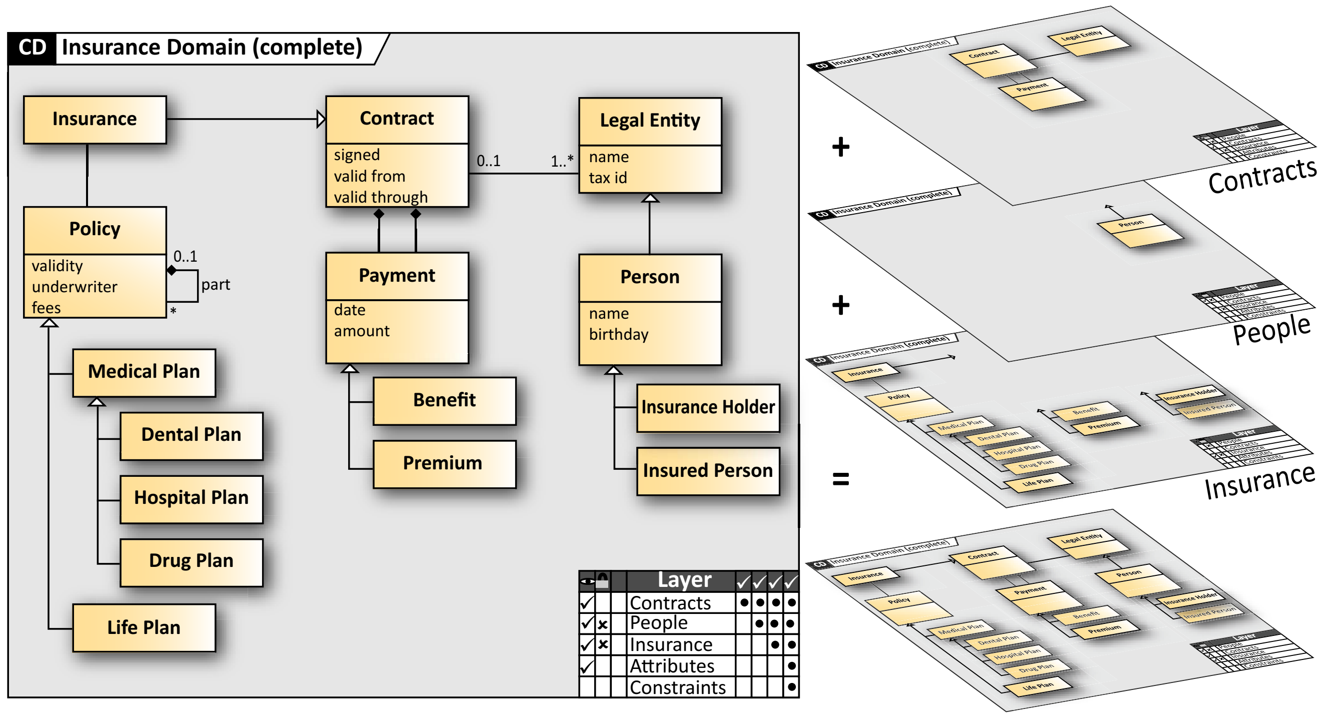

Figure 1: An initial example: a regular, unstructured class diagram (left), and the same diagram split up into orthogonal layers "Contracts", "People", and "Insurance" (right). The layers "Attributes" and "Constraints" are currently hidden. The "ticks" in the columns in the layers legend indicate which layers together should be considered as proper diagrams to be syntax checked. Here, the "Contracts" is a diagram in its own right, as well as together with layer "People", and with both "People" and "Insurance".

On the other hand, there are "proper" *Modeling Tools* like Enterprise Architect [20], Papyrus [32] or MagicDraw [15]: they implement a given language, and respect, more or less, the syntactic and semantic consistency rules of the language. Modeling tools often allow complex semantic operations, such as semantics-based folding, querying, simulation, and code generation. There are many UML modeling tools, but only one that we know of allows layering: Visual Paradigm [33]. In a personal communication, representatives of NoMagic, another leading vendor of modeling solutions, agreed that layering of diagrams had been considered in the past, as several clients had been asking for it repeatedly. Other features were prioritized higher so far, however.

Clearly, diagramming tools lack the many exciting semantic operations that the MDE community appreciates in their "proper" modeling tools. However, there is anecdotal evidence that the vast majority of conceptual models is created using drawing tools. It seems that flexibility, ease of learning, and ease of use offset the advantages of "proper" modeling tools. One possible explanation for this is that the main usage of conceptual models is to support informal usages like cognition and communication [2; 29] rather than the more advanced operations that require models of a higher degree of formality. If that is true, then a layering features as proposed in this paper has the potential to be of very high utility to practical modelers.

2 EXPLORING THE DESIGN SPACE

As a first example for a layered diagram, consider the class diagram of Fig. 1 (left). It shows a conceptual model from the insurance domain. It is fairly close to the maximum size for diagrams recommended in [28]. Despite a clean layout obeying most of the usual layout rules, it takes a while to understand this diagram. A modeler reading it will find it difficult to re-create the structure of the content that the modeler creating this diagrams may have had in his mind. Structuring the diagram into separate layers as in Fig. 1 (right) creates a narrative that makes it much easier to understand the diagram. By adding layers one by one, a reader is guided through the model. Each individual layer can be made to be a lot simpler than the overall model, and thus easier to understand. A comprehensive analysis of the usage modes of layered diagrams follows below in Section 3.

First, however, we need to explore the design space of layered diagrams, as layering is a complex feature. Depending on how the design decisions are taken, the properties of the overall feature vary greatly.

2.1 Relationships between layers

We have to decide whether layers are supposed to be completely independent of each other or have some kinds of relationships between them. Allowing arbitrary relationships implies that the

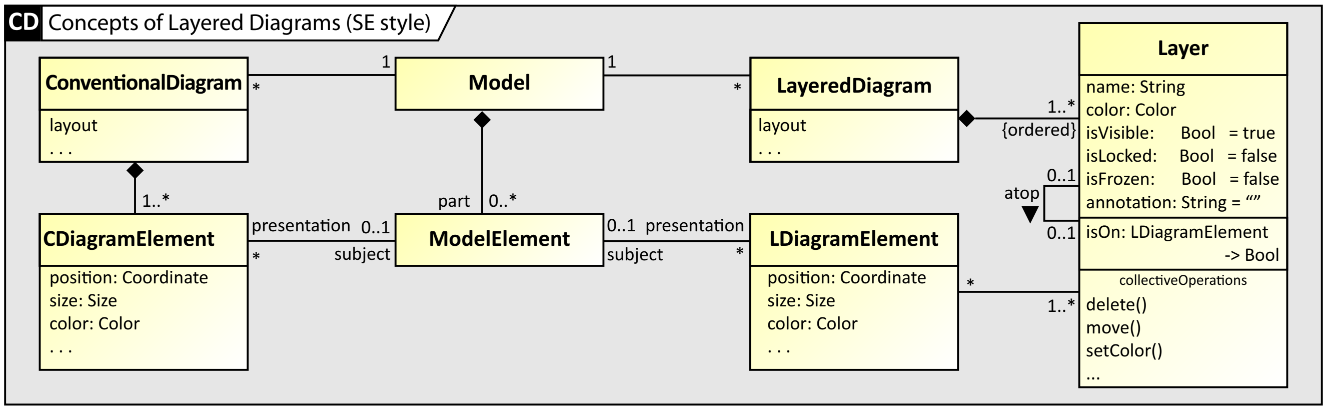

Figure 2: A conceptual model defining layers and layered diagrams in the context of SE-style conceptual modeling.

modeler has to specify and inspect existing dependencies. However, increasing the expressiveness of complex inter-layer relationships may decrease the usability of layers, and thus defeat introducing layers in the first place. On the other hand, without any kind of relationships between layers, the feature loses a lot of its expressiveness and usefulness. For instance, by its very name, layers are in an *atop*-relationship, that is, a total ordering of all layers. It implies a sequence, but not necessarily a dependency. In Fig. 1, the sequence encodes a particular narrative from the general to the specific, and contains part of the message of the model. In this particular example, there is only one reasonable sequence, but there are others where several sequences are possible. So we decide to endorse the *atop*-relationship, which is very easy to use by a legend-like table as shown in the bottom right corner of Fig. 1.

Some tools like Adobe Illustrator^{TM} provide a restricted *contains*-relationship by allowing (recursive) sub-layers. Layer elements might be considered arguments to a layer operator other than *atop*, like *XOR*. Using recursion, a model could be a sequence of layers, each of which is a group of mutually exclusive layers, which might be sequences again. While offering increased expressiveness, such features may compromise the ease-of-use and learnability that simple layers offer. The additional complexity (and reduced usability) could easily more than offset the additional expressiveness. Thus, for the time being, we restrict ourselves to plan layers without complex inter-layer relationships and recursion.

2.2 Unique assignment

We have so far silently assumed that diagram elements belong to one and only one layer. However, it is perceivable that layers may overlap; that is, diagram elements may be attached to more than one layer. Such elements are visible in a diagram if they are assigned to any of the layers that are currently visible. Similarly, an element is locked if it is assigned to a locked layer. This would allow us to model a simple form of dependency between layers by attaching required elements to all layers depending on them. That, however, makes adding (and removing) elements to layers a lot more complex and error prone. If we accidentally mis-assign an element, a modeler may have to check all available layers, for all elements in question to determine and remove offending assignments. Balancing usability and expressiveness again, we decide in favor of usability and decide to stick to unique assignment, as do most tools except Visio.

2.3 Self-containedness

Next, we have to decide whether we want all layers to be proper diagrams in themselves, that is, abiding by the syntactic rules of the respective modeling languages, without other layers being visible. Clearly, layers "People" and "Insurance" of Fig. 1 are not proper UML diagrams by themselves due to their dangling edges. So one might be tempted to demand that all layers together must be a proper diagram in the modeling language used. However, this may exclude using layers to display alternatives, such as alternative multiplicities on associations. Conversely, we are reluctant to drop syntax checks altogether. So we propose to allow users to specify sets of layers which, together, should be considered proper diagrams. This can be done by "ticking" sets of layers as shown on the right of the layering-table in Fig. 1. Observe that this, again, allows us some kinds of dependencies between layers, though this is optional, and easy to set and remove ticks.

2.4 Dynamic layers

Finally, we have so far treated layers as static entities with elements assigned to them explicitly by the user. In contrast, layers could also be defined implicitly, by a predicate "isOn" (cf. Fig. 2) that determines whether or not a given element is on a layer. For instance, particular classes of elements could be automatically assigned to a specific layer. This assignment predicate could be the element type (like the layers "Constraints" or "Attributes" in Fig. 1), or additions by a particular modeler or at a particular time or occasion, e.g., review comments, teacher's assessment in modeling teaching. There could also be more complex operations such as "all parts of a model violating a given threshold of a particular metric", or "all model elements with a particular status". Some modeling tools already offer a few features of this kind, e.g., semantic folding ("hide attributes" or "hide stereotypes"). Dynamic layers based on model semantics are contrary to the generic and lightweight addition to modeling which we intend layers to be in this paper. So, we restrict the admissible predicates to simple ones exploiting administrative properties only.

2.5 Layer operations

Apart from the static properties of layers, we also have to decide on the operations on the elements of a layer, or on the layers per se. Obviously, we need to be able to create and delete layers (and possibly sublayers), assign elements to them, identify the elements of a layer and, conversely, the layer of an object. Likewise, we must be able to show/hide layers thus showing/hiding all of the elements on the respective layer (same for locking). Of course, we have to be able to arrange the sequence of the layers, too. Using layering features in existing tools, it is easy to see that we also need a few household operations on layerst́, such as naming/renaming them, copying layers (and the presentation elements on them). There are more advanced operations, too, but those become apparent only when exploring the usage space (see Section 3).

2.6 Resulting design

To sum up our design space exploration, we design our diagram layering feature such that layers are *atop* each other, diagram elements are explicitly assigned to exactly one layer, and only "ticked" sets of layers are required to be syntactically correct diagrams. Fig. 2 contrasts the concepts used in our design of layered diagrams with the features of conventional diagrams. Obviously, the conceptual differences are small, which helps with the implementation of layered diagrams.

The layering feature we describe here is fairly similar to the one found in Adobe Illustrator and Photoshop, also in terms of user experience. Visio, on the other hand, takes a very different approach, implementing almost the exact opposite of our design. First, Visio layers have no *atop*-relationship between them. Second, elements may belong to several layers simultaneously. Pragmatically, layers in Visio are more reminiscent of named groups of elements with the purpose of applying graphic operations on them collectively. A (small) part of the function of layers is realized in Enterprise Architect's auto-coloring legends.

3 EXPLORING THE USAGE SPACE

We describe three scenarios that highlight shortcomings of existing model structuring mechanisms and point out how layers can help address the issue. All three scenarios are straightforward and common in large scale domain modeling. There are many more scenarios involving a wide variety of UML diagram types, but we do not have sufficient space in this paper to describe them all.

3.1 Orthogonal models

First, consider the case of model dimensions, that is, models that capture orthogonal angles, points of view, or dimensions, such as independent product aspects, features, or qualities. Consider a model that is being reviewed by several people, as in an inspection. Assume that the reviewers document their remarks as annotations in the model, which allows them to suggest changes or additions to the model "in place". At some point, the remarks are joined together yielding a common model with all review remarks in it. Using conventional diagrams, this amounts to a true model merge, which has several disadvantages: firstly, the resulting diagrams can become quite crowded and confusing when there are many comments on one issue. Secondly, extra effort is required to ensure that every comment can be traced to the reviewer responsible for it. Thirdly, the model may become inconsistent and difficult to understand when reviewers propose competing solutions to an issue.

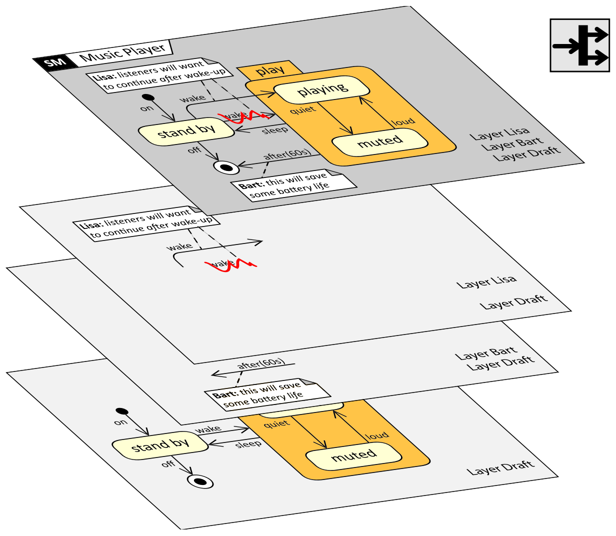

Figure 3: Layers may be used to capture orthogonal aspects

For the sake of the argument, let us assume there is a base version and three review versions. Each review version also contains different aspects, such as severity of the remark (comment, minor, major), and of course reviewers may disagree on the issue as well as on the severity. Eventually, another version is created that contains consolidated comments. Altogether, there are $(1 + 3 + 1) * 3 = 15$ diagram variants. Clearly, this is hard to handle as independent diagrams.

Using layers, however, reviewers would add their comments on separate layers, with their initials tagged to the layer name. They can spread their comments over different layers by severity. If we allow sub-layers, we get a compact tree of comments per reviewer. Such layers can easily be merged into a single diagram. In the scenario described before, we would end up with five main layers (base version, three review versions, consolidated comment). Each of the review versions (and the consolidated version) could have three sub-layers for severity. So, altogether, we have a single diagram with five layers, each of which has up to three layers. In the review session, we may selectively switch between different reviewers' viewpoints, but still can see several (or all) reviewers' comments simultaneously, if required. Similarly, we could address comments by severity. Also, it is always clear who the author of a comment is, by simply looking at the layer on which the contribution resides. Differences (and agreements) between reviewers can be seen rather easily. A similar case arises when merging concurrent versions of diagrams. Generally, layers are useful when considering orthogonal (though possibly overlapping) contributions or aspects of a diagram, see Fig. 3.

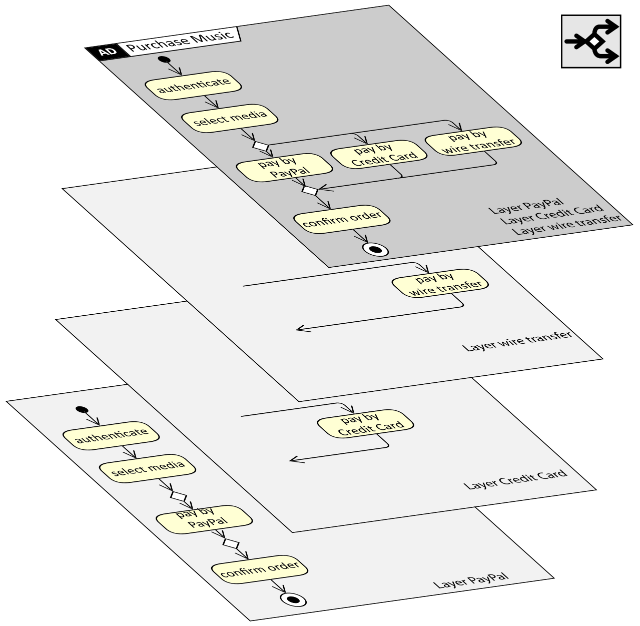

Figure 4: Layers may be used to capture alternative models

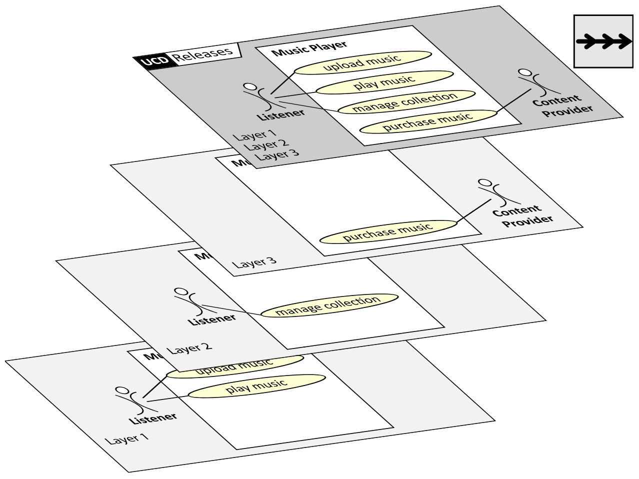

Figure 5: Layers may be used to capture sequential stages

3.2 Alternative models

Second, consider the case of modeling alternatives; that is, models that capture competing solutions to a given problem, diverging variants of designs, or branching versions of models. Consider for instance a model that contains alternative solutions to a given problem, say, different solutions to accommodate strategic choices, alternative implementations based on contingent variables, or simply variants of business processes depending on different inputs, as shown in Fig. 4. Assume that the alternatives have commonalities as well as differences, and that the model serves as documentation of implementation variants that are deployed to different customers.

Of course, one could simply join all alternatives into a single model, but then it might become quite difficult to keep an overview over the different variants, and what their respective elements are, which may lead to model clones (see [25]) when discarding unwanted alternatives. Also, only those parts where all variants overlap (i.e., the intersection of all alternatives) may be factored out, so that model parts may have to be stored redundantly, which may easily lead to inconsistencies.

Using layers, on the other hand, each alternative may use a separate layer as in Fig. 4. If an alternative is to be discarded, the layer can be deleted together with the model elements it presents. There is also no need to store overlapping model fragments redundantly, since the modeler may choose to show all layers that contribute elements to an alternative.

3.3 Consecutive models

Third, consider the case of consecutive models, i. e., sequences of stages or steps to document an evolutionary path or consecutive levels of refinement or elaboration. Consider a model that specifies and visualizes the release schedule of a project. Assume that a sequence of use case diagrams is used to represent a series of increments planned to be produced over time (see Fig. 5 for an example). Observe that each increment contains a set of actors and use cases that adds to the previous elements.

Showing all increments simultaneously in one diagram will make it less obvious to see both the progress over time and the individual contribution per increment. Using an individual diagram for each stage including all previous stages results in a high number of diagrams that are hard to maintain: changes to the i-th diagram of a sequence of n diagrams will have to be reflected in all diagrams from $i + 1$ to n, or the model's audience will be confused about changing layouts. Using layers, on the other hand, each increment can be put onto an individual layer, so that they can easily be shown in isolation, or as part of any interval of the sequence. Changing the elements of the i-th layer is immediately visible in all layers above i.

4 VALIDATION

Our research is inspired by the Design Science paradigm [9]: between 2009 and 2015, we iterated through three cycles of evaluation, design improvement, and tool enhancements.

4.1 First Iteration: laboratory testing

We implemented layers as a feature in the Advanced Interaction Design Environment (AIDE, see [23]), a tool for the model driven realization of graphical user interfaces. AIDE uses Window/Event diagrams (WEDs, see [21]), a notational variant of UML StateMachines enhanced to support group dynamic processes of GUI design. In WEDs, UML States represent GUI-elements (e. g., dialogs, buttons, and input boxes); Transitions represent events such as user actions or time-outs. Many similar notations have been proposed ("UI maps", "interaction landscapes", or "storyboards"); the key benefits of WED are (a) that it lends itself particularly well to stepwise, as-needed refinement of UI elements, and (b) that it allows formal analysis, too.

The drawback of this approach is that UI designs get very large very quickly: this became obvious in the very first practical example we modeled with AIDE, a calendar application. The underlying StateMachine consisted of 178 States nested up to 9 levels deep, and 218 Transitions, all in a single diagram. The (conventional) features AIDE offered for handling large diagrams (zooming and locator-maps) turned out to be insufficient. Inspired mainly by Adobe Illustrator, we implemented layers in AIDE, realizing the properties and features described in Section 2.

Subsequently, we used the new version of AIDE to restructure the example using layers in the controlled environment of a small modeling project by junior researchers that were unaware of the specific objective of validating layers. We logged problems, shortcomings, and incidents, finding many tool issues, and prompting many questions about modeling concepts as such, but, interestingly, not a single remark or problem related to the concept or usage of layers. All participants used layers in one form or another, though there was no mandatory requirement to do so.

From this stage we learned that layers can be used in modeling StateMachines with the same ease and simplicity as it is used in graphic design, say. In particular, navigating a large diagram and focusing on specific parts or aspects becomes very easy. Structuring the model by placing its elements on up to 20 layers provided a great help, and handling so many layers posed no significant problem to the study participants. However, different groups came up with very different systems of layers: Some used sets of layers per contributor, others used it to separate parts of the model, and yet others used layers for various purposes at the same time. So, defining the "right" layers seems to require a certain amount of care and thought. We speculate that the exact nature of layers is specific to the application domain and model purpose. In this example, we used the layers shown in Fig. 6 (a); (b) and (c) show other layer stacks useful for comparing independent contributions, and consecutive development stages, respectively.

Layer	Name
13	Scribble
12	ControlNodes
11	Review remarks
10	Comments
9	Explanaitons
8	Other Events
7	Touch Events
6	Keyboard Events
5	Mouse Events
4	Menues
3	Minor Dialogs
2	Major Dialogs
1	Dialog Groups
0	Background

(a)

Layer	Name
12	Bart (addded)
11	Bart (changed)
10	Bart (deleted)
9	Lisa (addded)
8	Lisa (changed)
7	Lisa (deleted)
6	Marge (addded)
5	Marge (changed)
4	Marge (deleted)
3	Homer (addded)
2	Homer (changed)
1	Homer (deleted)
0	Common

(b)

Layer	Name
11	Release 3, Increment 3
10	Release 3, Increment 2
9	Release 3, Increment 1
8	Release 2, Increment 3
7	Release 2, Increment 2
6	Release 2, Increment 1
5	Release 1, Increment 3
4	Release 1, Increment 2
3	Release 1, Increment 1
2	Test Scaffolding
1	Test Infrastructure
0	Generic

(c)

Figure 6: Large diagrams may require many layers: the layers we found useful in the case study (a); sample layers for comparing four different contributions (b); and sample layers from visualizing consecutive releases (c).

We observed that a full-scale view with hidden detail layers seems to combine the benefits from scaled-down views, such as the overview provided by locator-maps, with the benefits of full-scale views, such as annotation details in the main editing canvas. We hypothesize that this effect is brought about by reducing the cognitive load of the user through a reduction of display items. Obviously, controlled experiments are be needed to explore this hypothesis properly, which we will have to defer to future work.

4.2 Second Iteration: expert assessment

As a next step, we presented the improved AIDE to 22 expert modelers selected by convenience from industry and academia, and collected their feedback. Our main objective at this stage was to establish the validity of our earlier findings for a broader range of applications and modeling languages, since, up to this point, we had used layers only for StateMachines in the very specific area of GUI modeling. Thus, our main question was whether the experts were able to carry over the concepts to other types of notations and application domains, and whether they assessed the approach beneficial in their respective settings.

We were surprised to receive unanimously positive feedback: all experts immediately saw the potential benefit of layers in their respective fields and could well imagine using such a feature in previous and current projects. Some of them readily proposed more examples of usage scenarios, some of which have been incorporated in the scenarios presented in Section 3. Of course, this feedback does not constitute a scientific finding, as it lacks validity and rigor of a proper empirical study. However, the objective at this point was to engage diverse feedback and open up the design funnel rather than testing a hypothesis.

4.3 Third iteration: field testing

Encouraged by the feedback from the initial two rounds of validation and improvements, we enhanced AIDE's layering and deployed it to the classroom. As part of the course work in a requirements engineering class at the Technical University of Denmark, we asked students to create a UI storyboard. Ten teams of 4-6 students collaborated on group projects over two weeks with three plenary and supervised work sessions of 4h each, and homework assignments. There was only a very brief introduction to the tool, barely mentioning layers. In particular, using the layering feature was not a mandatory part of the assignment. Students were given complete freedom in how they organized their group work, using available tools.

Again, we received many questions concerning modeling concepts, assignment content, tool installation, and general aspects of usability, but none concerning the layering features. Not all students used layering in their models, and those that did used it in other ways than intended at times. Two particular usage modes stood out in the sense that they were unanticipated, but immediately made sense when inspected. The first usage mode was using layers *topographically*, that is, split the overall diagram into different sector-like areas, each of which was occupying one layer (or set of layers). When asked, the students explained that time pressure had forced them to split up their group assignment, work independently, and then combine their contributions as orthogonal layers. Thus, layers were used graphically and semantically at the same time. The second observation actually showed up several times independently when several groups exploited the layers in their plenary result presentations. Here, different approaches were found: some groups represented different parts of their contents on different layers while others used layers in an orthogonal way to show consecutive levels of detail.

Note, that the students had merely been informed of the availability of layers, without any instruction in their usage; the students used layers based on intuition. The course schedule prevented students from copying other groups' solutions on the fly. That is to say, several groups came up with similar usage modes independently. In both cases students immediately had clever and unanticipated ideas about how to exploit layers. However, they had to do a considerable amount of primitive copy-paste work to achieve their goals.

4.4 Lessons learned

We gather three insights from our validation. First, layers are every bit as intuitive and useful as we have expected. Layered diagrams are a real improvement with many useful applications, for all kinds of diagrams. Second, while many modelers had great ideas of how to use layers, some didn't use them quite as readily, or needed some time (more than was available in the situation) to get started with using layers effectively. This led to the idea of supporting the adoption of layers by creating stacks of appropriately named default layers. Third, we clearly saw that the tool support we had offered was insufficient to support all of these scenarios. More operations working on a higher level could be added with much benefit. Such operations should highlight the opportunities for enabling and improving collaborative work through layers. In particular, password-protecting layers against unlocking ("freeze") will allow advanced modes of collaborative modeling across time and distance.

5 RELATED WORK

It appears that the only mainstream UML modeling tool offering layers is VisualParadigm [33]. There seems to be no research published on using layers apart from our own work [23; 24; 26] and a mentioning in passing in [17]. Without layered diagrams, modelers only have three options to highlight different alternative but correlated views of models: using an independent diagram per aspect, visual or textual annotations, and animated or three-dimensional models. Creating an individual diagram per aspect is straightforward, but of course there are redundancies that have to be maintained as diagrams evolve. For Event Process Chains (EPCs), where there is exactly one diagram per model, model clones will be introduced [25]. In order to show k aspects, k diagrams are needed but only $\lceil log(k) \rceil$ layers in one diagram.

Another way of dealing with alternative but correlated views is to merge them together into a single model and add a textual or visual cue for each aspect, such as a stereotype, or color coding. This works well for small numbers of views, but is limited by the built-in limitations of the human visual apparatus. Discrimination by textual annotations relies on conscious processing and is thus cognitively inefficient [13]. Color coding, while being processed pre-attentively, is limited to between five and seven different colors in one visualization (cf. [3]). This has been confirmed in many software engineering contexts, e.g., the use of background color to highlight the implementation of alternative aspects in IDEs [6]. Thus, this approach is ineffective for modeling languages that use color as part of their syntax (e. g., EPCs in ARIS [4; 19]. UML, on the other hand, expressly avoids the use of color in the language definition, so color coding is at least possible, and has been proposed, e. g., for highlighting version differences [16]. See [13; 14; 30] for discussions of the visual properties of diagrammatic notations in software engineering.

Animated diagrams can represent sequences of models similar to consecutive layers. Thus, animations are effective for visualizing sequences, such as the structural evolution of a static model or runs of a behavioral model. Preliminary results indicate there are applications in teaching modeling and object-oriented programming [18; 31], but there is no evidence to similar benefits for industrial usages. Like (simple) layers, animated models are limited to simple paths: branching paths, multiple paths, or discontinuous sub-paths are not handled well. Complex implementations of layers that take different design decisions than those we chose in Section 2 will probably provide greater expressiveness.

Three dimensional UML diagrams have been studied since the first version of the UML [7]. Three-dimensional diagrams turn two-dimensional diagram elements into spacial elements arranged in three dimensions rather than two. Choosing the third dimension to represent time, we see that animated models are special cases of three-dimensional models. Dwyer has suggested that three-dimensional diagrams are likely to be able to convey three times as much information as their two-dimensional counterparts [5], and McIntosh and colleages found three-dimensional UML state machine diagrams effective to visualize complex state machine models in Mechatronics [10–12]. However, navigating even small three-dimensional models has proved to be a challenge for many modelers, so that McIntosh et al. experimented with replacing unrestricted navigation by "guided tours" (i.e., animation).

Layered models are less powerful than three-dimensional models in the sense that the diagram elements remain two-dimensional, and the user is restricted to a single view axis. On the other hand, three-dimensional diagrams do not offer the typical operations associated to layers, such as show/hide, lock/unlock, or merge, and it is not clear how these could be added to three-dimensional diagrams in a usable way. Conversely, layered models are strictly more powerful than animated models: like animated models, there is only one view axis in layered models. However, layers may be shown simultaneously, not consecutively as in an animation. Selecting and using any subset of layers is straightforward, while selecting sets of points or intervals on a continuous dimension involves more infrastructure and thought.

McIntosh [10] has pointed out the substantial implementation effort and the complex usage concepts and support mechanisms required to exploit the full benefit of three-dimensional diagrams. Together, they limit the adoption of three-dimensional diagrams and animation, and indeed, no widely used UML modeling tool offers three-dimensional modeling facilities today. Layering, on the other hand, is available in many tools in domains ranging from architecture, via medicine to electrical engineering. Many users have an immediate intuition about layers and apply the concept effortlessly, which makes the adoption of layered diagrams look much more promising than that of three-dimensional diagrams.

6 DISCUSSION

6.1 Summary

In this paper we propose layers as a means to enrich and structure large diagrams. We explore the design space of layered diagrams, and arrive at a set of pragmatic design decisions. We explore the usage space of this variant of layered diagrams and present three common scenarios that each illustrate advantages of layers over conventional diagrams. We have implemented layers in an experimental modeling tool, and have elaborated our approach through a case study, expert panel assessments, and a field test in the classroom, yielding specific feedback that has led to successive improvements of the concepts and their implementation.

6.2 Conclusion

We conclude that layers complement conventional diagrams by providing *dependent* views, which can drastically reduce the number of diagrams in many common place modeling scenarios. Today, such facilities are missing from most mainstream modeling tools, and many drawing tools. In contrast to existing approaches like three-dimensional and animated UML diagrams, layered diagrams offer three benefits. First, the concepts behind layered diagrams are in widespread use in many fields which makes the concept easy to grasp and apply for many modelers. Second, our validation exhibited a great number of useful application scenarios for layered diagrams, and there is no reason why their usefulness should be restricted to the UML notation. Third, layers are straightforward to implement in modeling tools, so this improvement comes at little cost.

6.3 Future Work

We have so far not considered complex inter-layer relationships and recursion, expecting an unfavorable balance of expressiveness and complexity, reducing the overall usefulness. However, it should be interesting to see how far this approach can be pushed. In particular, we see applications in novel ways of modeling (software) product lines. Also, layers may be helpful for supporting version control and more generally, contribute to collaborative modeling.

Of course, any further exploration will need to include a much more thorough investigation of user behavior using layers, which in turn requires an industry strength modeling tool where the layering feature can be fully controlled to explore novel usage concepts.

REFERENCES

[1] S. Abrahão, F. Bordeleau, B. Cheng, S. Kokaly, R. F. Paige, H. Störrle, and J. Whittle, "User Experience for Model-Driven Engineering: Challenges and Future Directions," in *ACM/IEEE 20th Intl. Conf. Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2017, pp. 229–236.

[2] S. Baltes and S. Diehl, "Sketches and Diagrams in Practice," in *Proc. 22nd ACM SIGSOFT Intl. Symp. Foundations of Software Engineering (FSE)*. ACM, 2014, pp. 530–541.

[3] J. Bertin, *Semiology of Graphics: Diagrams, Networks, Maps.* Univ. Wisconsin Press, 1983.

[4] R. Davis, *Business process Modelling with ARIS: A Practical Guide.* Springer Verlag, 2001.

[5] T. Dwyer, "Three Dimensional UML Using Force Directed Layout," in *Proc. 2001 Asia-Pacific Symp. Inform. Visualisation.* Australian Comp. Soc., 2001, pp. 77–85.

[6] J. Feigenspan, M. Schulze, M. Papendieck, C. KÃďstner, R. Dachselt, V. Koppen, and M. Frisch, "Using Background Colors to Support Program Comprehension in Software Product Lines," in *Proc. 15th Conf. Evaluation & assessment in Software Engineering (EASE 2011).* IET, 2011, pp. 66–75.

[7] M. Gogolla, O. Radfelder, and M. Richters, "Towards three-dimensional representation and animation of UML diagrams," in *Proc. 2nd Intl. Conf. Unified Modeling Language.* Springer Verlag, 1999, pp. 489–502.

[8] I. Gorton, C. Cuesta, and M. A. Babar, Eds., *Proc. 4th Eur. Conf. Sw. Architecture (ECSA): Companion Volume.* ACM, 2010.

[9] A. Hevner, S. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[10] P. M. McIntosh, "X3D-UML: User-Centred Design, Implementation and Evaluation of 3D UML Using X3D," Ph.D. dissertation, RMIT University, 2009.

[11] P. M. McIntosh and M. Hamilton, "X3D-UML: 3D UML Mechatronic Diagrams," in *Proc. 21st Australian Software Engineering Conference.* IEEE, 2010, pp. 85–93.

[12] P. M. McIntosh, M. Hamilton, and R. Schyndel, "X3d-uml: 3D UML State Machine Diagrams," in *11th Intl. Conf. Model Driven Engineering Languages and Systems*, Czarnecki et al., K., Ed. Springer Verlag, 2008, pp. 264–279.

[13] D. L. Moody, "The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering," *IEEE Trans. Software Engineering*, vol. 35, no. 6, pp. 756–779, Nov/Dec 2009.

[14] D. L. Moody and J. Van Hillegersberg, "Evaluating the Visual Syntax of UML: An Analysis of the Cognitive EffectiveneÃ§ of the UML Family of Diagrams," in *Proc. Intl. Conf. Software Language Engineering (SLE 2008)*, ser. LNCS, no. 5452. Springer, 2009, pp. 16–34.

[15] NoMagic INC, "MagicDraw^{TM} ," available at http://magicdraw.com.

[16] D. Ohst, M. Welle, and U. Kelter, "Differences between versions of UML diagrams," in *Proc. 3rd Eur. Software Engineering Conf. 2003 (ESEC'03)*, September 2003, pp. 227–236.

[17] J. Peltonen, M. Felin, and M. Vartiala, "From a Freeform Graphics Tool to a Repository Based Modeling Tool," in *Proc. 4th Eur. Conf. Sw. Architecture (ECSA): Companion Volume*, I. Gorton, C. Cuesta, and M. A. Babar, Eds. ACM, 2010, pp. 277–284.

[18] O. Radfelder and M. Gogolla, "On better understanding UML diagrams through interactive three-dimensional visualization and animation," in *Proc. Working Conf. Advanced Visual Interfaces.* ACM, 2000, pp. 292–295.

[19] A.-W. Scheer, *ARIS-Business process Modeling.* Springer Verlag, 1999, 2nd edition.

[20] Sparx Systems International, "Enterprise Architect^{TM} ," available at https://www.sparxsystems.com.

[21] H. Störrle, "Group Exercises for the Design and Validation of Graphical User Interfaces," in *Proc. Modellierung 2002*, ser. Lecture Notes in Informatics, M. Glinz and G. Müller-Luschnat, Eds., no. P-12. Gesellschaft für Informatik, 2002, pp. 135–146.

[22] ——, "Large Scale Modeling Efforts: A Survey on Challenges and Best Practices," in *Proc. IASTED Intl. Conf. Software Engineering*, W. Hasselbring, Ed. Acta Press, 2007, pp. 382–389.

[23] ——, "Model driven development of user interface prototypes: an integrated approach," in *Proc. 4th Eur. Conf. Sw. Architecture (ECSA): Companion Volume*, I. Gorton, C. Cuesta, and M. A. Babar, Eds. ACM, 2010, pp. 261–268.

[24] ——, "Improving Modeling with Layered UML Diagrams," in *Proc. 1st Intl. Conf. Model-Driven Engineering and Software Development*, J. Filipe, R. C. das Neves, S. Hammoudi, and L. Ferreira Pires, Eds. SCITEPRESS, 2013, pp. 206–209.

[25] ——, "Towards Clone Detection in UML Domain Models," *J. Softw. Syst. Model.*, vol. 12, no. 2, pp. 307–329, 2013. [Online]. Available: http://link.springer.com/article/10.1007/s10270-011-0217-9

[26] ——, "From Pen-and-Paper Sketches to Prototypes: The Advanced Interaction Design Environment," in *Joint Proc. MODELS 2014 Poster Session and ACM Student Research Competition*, S. Sauer, M. Wimmer, M. Genero, and S. Qadeer, Eds., vol. 1258. CEUR, 2014. [Online]. Available: http://ceur-ws.org/Vol-1258

[27] ——, "On the Impact of Layout Quality to Understanding UML Diagrams: Size Matters," in *Proc. 17th Intl. Conf. Model Driven Eng. Lang. and Syst. (MoDELS)*, ser. LNCS, J. Dingel and others, Eds., no. 8767. Springer Verlag, 2014, pp. 518–534.

[28] ——, "Diagram Size vs. Layout Flaws: Understanding Quality Factors of UML Diagrams," in *Proc. 10th Intl. Conf. Empir. Softw. Eng. and Measurement (ESEM)*, M. Genero, A. Jedlitschka, and M. Jørgensen, Eds. ACM, 2016.

[29] ——, "How are Conceptual Models used in Industrial Software Development? A Descriptive Survey," in *Proc. 21st Intl. Conf. on Evaluation and Assessment in Software Engineering (EASE)*, E. Mendes, K. Petersen, and S. Counsell, Eds. ACM, 2017.

[30] H. Störrle and A. Fish, "Towards an Operationalization of the "Physics of Notations" for the Analysis of Visual Languages," in *16th Intl. Conf. Model Driven Engineering Languages and Systems (MoDELS)*, ser. LNCS, A. e. a. Moreira, Ed., no. 8107. Springer Verlag, 2013, pp. 104–120.

[31] U. Thaden and F. Steimann, "Animated UML as a 3D-Illustration for Teaching OOP," in *Proc. 7th Ws. Pedagogies and Tools for Learning Object-Oriented Concepts*, 2003.

[32] The Eclipse Foundation, "Papyrus Modeling Environment," available at https://eclipse.org/papyrus.

[33] Visual Paradigm International, "Visual Paradigm^{TM} ," available at https://www.visual-paradigm.com.