FlexiSketch: A Mobile Sketching Tool for Software Modeling

Dustin Wüest, Norbert Seyff, and Martin Glinz

University of Zurich, Department of Informatics, RERG,
Binzmühlestr. 14, 8050 Zurich, Switzerland
{wueest,seyff,glinz}@ifi.uzh.ch

Abstract. Although most software engineers have access to various modeling tools, they often use paper and pencil to sketch ideas and to support modeling activities. This is particularly true when they are working in the field, for example gathering requirements from stakeholders. Sketches documented on paper very often need to be re-modeled in order to allow further processing – an error-prone and time-consuming task. The aim of our work is to better integrate these early sketching and modeling activities into the overall software engineering process. We have prototyped FlexiSketch, a mobile application that supports freeform, flexible, in-situ modeling and allows software engineers to annotate their models. Apart from the application and the underlying conceptual solution we also present the results of initial experiments. Those suggest that the tool supports freeform sketching similar to paper and pencil, and that practitioners would be willing to use a tool like FlexiSketch in their daily work.

Keywords: flexible, sketch-based, modeling, software engineering.

1 Introduction

Early software engineering (SE) activities include gathering and documenting needs that stakeholders of a future software product have. Also, first design sketches of the future software system are created. These tasks often ask for creativity. Diagram-like sketches (consisting of nodes and edges) can help to depict current systems and communicate ideas in a simplified way.

Engineers and stakeholders mostly prefer to use paper and pencil, whiteboards, and flip charts for sketching in early SE phases [1,2]. This is mainly due to two reasons: First, engineers and stakeholders meet in various places. Paper and pencil, or flip charts, are available anywhere and are instantly ready for use. Second, they are easy to use. They allow for informal, unrestricted sketches. In contrast, most SE modeling tools follow a more formal modeling approach [3] and require an engineer to use a specific modeling language and syntax.

Once relevant ideas are documented on paper, the question is how to store, distribute, and make the information amenable for further processing. One option is to take photographs of sketches and later, back in the office, re-model the information manually with the help of a software modeling tool. This media break is error-prone and the re-modeling task time-consuming. Information may get lost or be interpreted in a wrong way. This is particularly a risk when another person is responsible for the re-modeling task or information is transcribed after some time has passed by.

To avoid the need for re-modeling information manually, we propose that software engineers use lightweight software tools right from the beginning of a project. Such tools must have a similar availability to that of paper and pencil-based approaches and allow for freeform sketching. In our current research, we focus on multi-touch mobile devices such as tablet computers to support these sketching activities. Today, such devices are widespread, ad-hoc available, and have sufficient computing power. Therefore, we consider such devices to be an ideal platform for supporting engineers with informal sketching and modeling tools which also allow them to stepwise refine and formalize their sketches.

In this paper we present a tool-supported approach that provides users with free-form sketching capabilities and allows them to annotate their sketches, thereby defining their notations and enabling a semi-automatic formalization of the sketches. Furthermore, we elaborate on the results of two experiments that focused on qualitative feedback regarding the usability and utility of our approach.

The remainder of this paper is structured as follows: We present our research goal and approach in Section 2. The tool prototype is described in Section 3. The two experiments and results are discussed in Sections 4 - 7. Section 8 presents related work, and Section 9 presents conclusions and future work.

2 Flexible Sketch-Based Modeling

2.1 Main Goal

The overall goal of our work is to unite the flexibility of unconstrained sketching with the power of formal modeling. In this context, we want to provide a tool-supported approach that (i) allows users to sketch any informal models, (ii) provides means for assigning syntax and semantics to sketched elements on the fly, and (iii) supports the semi-automated transformation of sketches into classic semi-formal models (e.g. a class diagram or a statechart) [4]. We envision our approach to allow for free, flexible sketching (as pen and paper does), and at the same time support a step-wise beautification and formalization of the sketches, thus avoiding the media break between early software engineering sketches and semi-formal models [5].

2.2 Key Requirements

Discussions with experts and the study of related work led to the following high-level requirements that we consider to be relevant for building a solution that fulfills end-user needs:

High flexibility: Users shall be able to sketch any kinds of diagrams. There should be no restrictions limiting users' expressiveness.

Natural sketching: The tool shall allow the use of input devices that give users a natural feeling for sketching, for example, a tablet PC with a pen or an electronic whiteboard.

Formalization capabilities: The tool shall support the transformation of diagram sketches into classic semi-formal models by, e.g. a semi-automated method, thus avoiding media breaks.

Speed: The tool shall allow fast creation and annotation of sketches. If the process is not faster than traditional sketching followed by model re-creation in a modeling tool, nobody would be motivated to use it.

The biggest challenge regarding our work is the aim to give users maximum flexibility in what they are sketching, not restricting them to any specific language or notation. Yet, a tool needs to "understand" what a user is sketching in order to perform any transformation of sketches into semi-formal models. We address this challenge by giving users the freedom to draw anything they want, and request that users assign meaning to sketches in due course with the help of annotations. These annotations will then be turned into corresponding metamodel elements.

2.3 Our Approach

We aim at a process that lets users switch arbitrarily between three work modes (Fig. 1): in the *modeling* mode, the user creates, augments or modifies sketches. In the *metamodeling* mode, the user annotates sketches and the tool creates metamodel elements based on these annotations. In the *sketch recognition* mode, the tool transforms sketched elements into semi-formal model elements by recognizing and interpreting sketches based on the information currently available in the metamodel. To make this process work, annotating must be easy and straightforward. No programming, scripting, or metamodeling skills should be required for this task. This free interleaving of modeling, metamodeling and recognition activities is the key difference between the presented approach and related work, where any form of metamodeling has to be done first. The following paragraphs depict our vision in more detail.

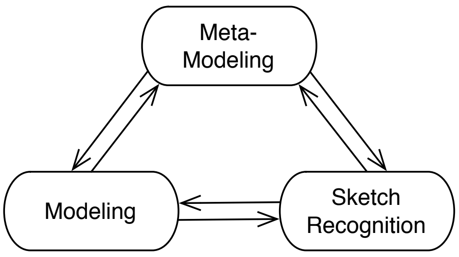

Fig. 1. The three activities leading to a semi-formal model in the end

Modeling is facilitated by two drawing modes. One mode mimics a whiteboard and allows for free sketching, while the other mode enables users to modify individual symbols (e.g. scale, move, copy). As opposed to other work, we do not define the modes as exclusive modes, i.e. the modes are not switched by pressing a dedicated button. The modes are switched implicitly. We believe that a single mode stays closer to the pen and paper metaphor. As soon as a user selects an already sketched object, she can make modifications.

Metamodeling starts when users assign a type to a previously sketched symbol. The symbol then gets added automatically to a dynamic symbol library. A symbol library consists of all defined symbols including their types, and allows to decouple types and their graphical representations. Symbol libraries are the first part of a lightweight, end-user metamodeling approach that supports users in defining a modeling language. Each symbol library contains a set of symbols belonging to a specific context, i.e. a specific diagram type. Users can define multiple contexts where the same symbol has different meanings. Because people like to mix different visual conventions during creative tasks [6], multiple symbol libraries can be active at the same time. A big challenge is to invent an easy-to-use interface for more complex metalanguage definitions, such as associations and rules for associations.

Sketch recognition happens in an interactive manner. Users are encouraged to take part in the recognition process to train the recognizer. They are also able to decide when and if recognition feedback should be given or not. These options assure that users do not get distracted from their sketching.

Our flexible process produces two kind of re-usable artifacts: the models drawn by the user, and a partial metamodel that is contained in the created (or re-used) symbol libraries. We envision that these two artifacts will make it easy to share models between different persons. Moreover, exporting the sketched models and further processing them in more formal SE tools comes within reach.

3 The FlexiSketch Prototype

We developed the FlexiSketch prototype, a mobile tool for freeform sketching with the focus on diagram sketching for SE. The prototype serves as proof of concept that the idea of letting the user switch flexible between all three activities works. In this section we present our prototype in detail, and we show how it realizes each of the three activities depicted in Figure 1.

Our tool is written in Java using the Android SDK for the Android OS, supporting version 3.0 and above[1]. We especially focused on tablet computers (with screen sizes around 10 inches) since we consider smart phone screens to be too small for effective sketching. Figure 2 shows a screenshot of our prototype.

[1] FlexiSketch is available on Google Play (formerly known as Android Market). A video demonstration can also be found on Google Play or at `http://www.youtube.com/watch?v=D06t0K50tzw`

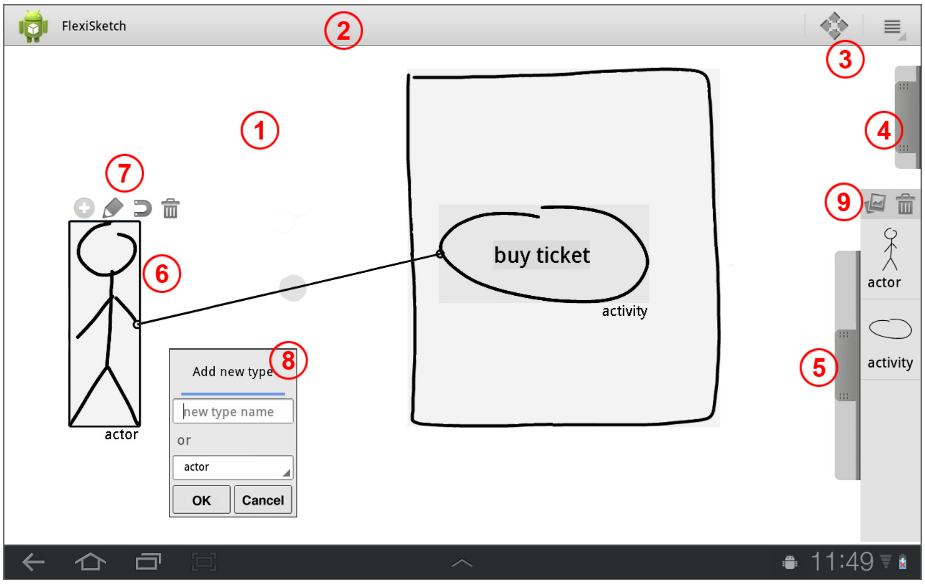

Fig. 2. Screenshot of the FlexiSketch prototype

3.1 Modeling

In order to emphasize the possibility of freeform sketching, the tool shows a white drawing canvas (1) when it is started. Additional functionality is hidden in the action bar at the top (2) and pull-out containers (4,5) on the right edge of the screen. To not distract the user from freeform sketching, we wanted to hide as much of the GUI as possible. However, we included a permanent action menu bar at the top of the screen to make some functions like scrolling quickly accessible. Scrolling of the drawing canvas is activated at the push of a button (3), and allows the user to draw sketches in the size of an A4 paper, independent of the actual screen size of the mobile device.

The user can start to draw like she would in any other paint application, either with her fingers or with a stylus. When the user stops to draw a particular item and removes the fingers from the screen for a predefined amount of time, the drawing is converted into a distinct symbol. The conversion of drawings into a symbol is not visible for the user per se, but the tool colors the background of a symbol with a light gray to give feedback that the conversion took place.

Symbols can be selected by tapping on them (touching the screen and lifting the finger without moving it). A selected symbol (6) is highlighted with a rectangular border around it and some context menu icons on top of it (7). The symbol can then be manipulated, dragged around, or deleted. The tool does not provide two distinct modes for drawing and symbol manipulation in order to allow for freeform drawing at any time. If the user touches the screen outside of the selected symbol and moves the finger, i.e. on white space or another symbol, she can just continue to draw naturally.

When the user starts sketching on top of one symbol and ends the stroke on top of another symbol, the tool recognizes the drawing as an association between the two symbols. It replaces the drawing by a straight line. In the middle of the line there is an anchor point that allows to select the line (otherwise the line would be hard to grab) and show its context menu. The line replacement is optional and can be switched off in the options menu.

An arbitrary amount of text boxes can be attached to symbols via the context menu (7). The text boxes are tied to the symbols, but can also be moved individually.

3.2 Metamodeling

The prototype provides means for defining symbols by assigning types to them. This is the basic functionality needed for the symbol library to work. After sketching a symbol, the user can choose to define its type via the context menu (7). A small popup menu provides the user with a list of all types that she already created, and an option to add a new type (8).

When the user decides to define a new type, she enters a name for it with the standard virtual keyboard. The tool creates a new entry in the symbol library with the given name, and stores the selected symbol in it. The symbol library contains entries for all defined types, and each type is graphically represented by at least one symbol (the symbol that was selected when the user defined the type). A list consisting of these graphical representations is shown in the pull-out container at the right edge of the screen (5). From there, defined symbols can be re-used with a drag-and-drop gesture.

The symbol library allows to add multiple symbols to the same type. This flexibility is needed because different users might draw different symbols to depict the same meaning. The opposite can also happen: the same symbol might have different meanings in different contexts, i.e. in different diagram types. Therefore, a symbol library is meant to be a collection of symbols belonging to a single diagram type. Symbol libraries can be stored and loaded (9) independently from the user sketches.

Theoretically, associations can have different types (similar to symbols), but in the current prototype, the number of association types is limited and bound to predefined line styles. The look can be quickly changed via a drop-down menu.

To help the user to not forget defining the symbols, the main menu (2) includes an option to visually highlight all symbols on the screen that do not yet have a type assigned.

Types assigned to symbols can be shown or hidden through the options menu. This allows the user to show types while she is taking care of type definitions. She can hide the types when she is working with text boxes, which do not belong to type definitions, but to particular instances of types. For example, in a class diagram most symbols are of the type class, but each symbol on the screen has a different class name written in a text box. To reflect the distinction between symbol types and text content of symbols, the functionality is strictly separated, accessible through two different context menu icons.

3.3 Sketch Recognition

We distinguish between *object recognition* and *sketch recognition*. We define object recognition as the automatic process of dividing user drawings into distinct symbols (and associations). The prototype does this while the user draws by using a simple timeout mechanism.

We define sketch recognition as the automatic process of comparing symbols to each other in order to identify similar symbols. For example, the user draws a stickman, and assigns the type *actor*. The sketch recognizer detects when the user draws a second stickman, so that the tool can automatically assign the type *actor* to it.

The prototype performs sketch recognition on drawn symbols. After the user draws a symbol, the sketch recognition algorithm searches the symbol library whether there are similar, already defined symbols. If the algorithm does not find any similar symbol, nothing happens. If a similar symbol is found, a pop-up on the bottom of the screen presents the symbol type for several seconds. If multiple similar symbols are found, the three closest matches are shown. The user can tap on one of the proposed types to assign this type to the newly drawn symbol. Choosing one of the proposed types is optional. The user can also decide to ignore the proposals and continue sketching. In this case the proposals will fade out again. This mechanic allows to give non-disruptive sketch recognition feedback to the user. If the user chooses one of the proposals, the new symbol is added to the respective type entry in the symbol library. In this way we realize a trainable sketch recognizer that learns from user inputs. The training is transparent to the user in order not to distract her from her actual task.

When the user draws a symbol that looks similar to an already defined symbol, an optional feature allows to replace the newly drawn symbol by the defined one. This can help to get a more uniformly looking sketch at the end. The feature can also be used to beautify symbols: A user can place one of the provided standard geometric shapes in the drawing canvas and assign a type to it. When she then draws a similar symbol by hand, the sketch recognition mechanism proposes to replace it with the geometric shape. Figure 3 shows a use case diagram where sketch recognition was used to beautify the hand drawn symbols.

For the implementation of the sketch recognition algorithm, we adapted an algorithm that is based on the Levenshtein string distance [7], which calculates the distance between two strings.

4 Evaluation

We used the developed prototype to evaluate our approach in early 2012. We performed two experiments to assess the usability and utility of our tool-supported approach. The perceived utility, or usefulness [8], affects the intention to use the approach, and therefore influences how much the approach will be adopted in practice [9].

We especially wanted to know whether our tool provides the same kind of flexible, ad-hoc sketching support that is also provided by paper and pencil-based

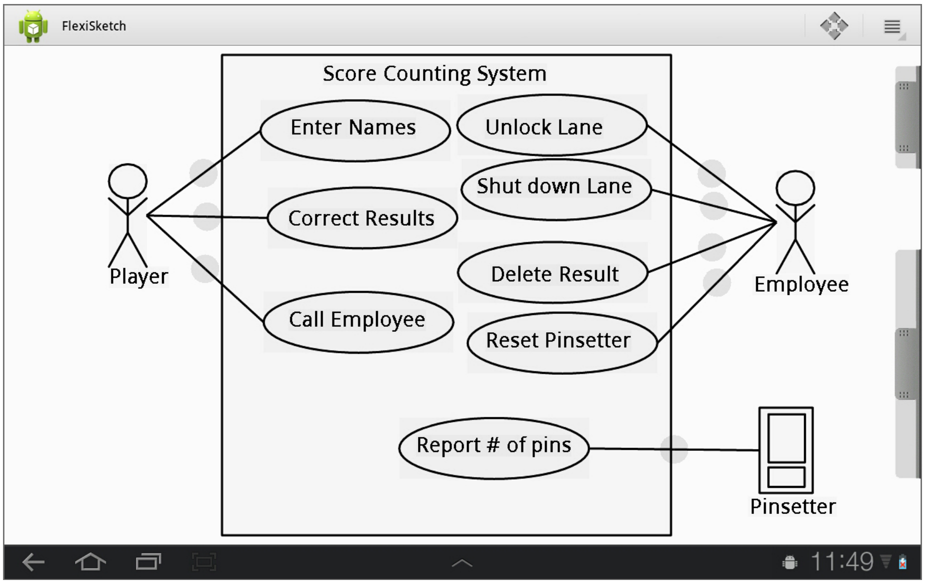

Fig. 3. Sketch recognition beautified a hand drawn use case diagram

approaches. Therefore we asked: What are the differences between FlexiSketch and paper and pencil-based approaches (*RQ 1*)? We also wanted to know whether users can in general classify the elements they draw with our tool into types (*RQ 2*). For adoption in practice, a tool has to be tailored to the needs of its users. Therefore we asked: Can they sketch diagrams with our prototype well enough such that they would consider adopting our tool in practice (*RQ 3*)? This also included to investigate what kind of diagrams software engineers sketch in practice.

The goal of the first experiment was to identify usability issues and assess whether our tool and approach allow users to sketch diagrams. In this experiment, we focused on answering RQ 1 and RQ 2 for one particular diagram type: use case diagrams. We chose this diagram type because (i) use case diagrams are widely known (and thus can serve as a common basis for our evaluation), (ii) they are typically used in early SE phases, and (iii) they are well-suited for transforming sketches into semi-formal models. This experiment did not allow to answer RQ 3.

The goal of the second experiment was to generalize from use case diagrams in order to answer RQ 1-3 for arbitrary diagrams. We wanted to investigate how the tool prototype can handle different diagram types (RQ 3). Another goal of the second experiment was to gain insights into how physical media for sketching are used in SE practice, and what kind of sketches practitioners draw.

In both experiments, participants used an Android tablet and a stylus to draw diagrams with our prototype and assign types to sketched symbols.

5 Experiment #1: Feasibility of our Approach and Tool Usability

This first experiment investigated the usability of our tool and the feasibility of our approach. The experiment was conducted in a controlled setting with 17 participants from research and industry: four undergraduate students and four PhD students in Computer Science, and nine software engineering practitioners, all having several years of experience (three persons counted as practitioners have returned from practice to academia). Practitioners were between 25 and 70 years old, and included experts in SE and HCI. 15 participants considered themselves to be novice users regarding touch interfaces (using smart phones for standard tasks, having no or little experience with tablet devices). Two persons considered themselves to be expert users (having a more in-depth understanding of mobile devices and their features, which includes tablet devices).

5.1 Method

The evaluation was conducted with each participant separately. It included a short briefing, the actual evaluation where participants had to perform three tasks using our tool, and a debriefing. In the briefing, we first asked demographic questions about the participant's knowledge in SE, touch interfaces, and working experience. Then we presented our research and gave a short introduction to the tool (about three minutes). For the first task, participants were asked to sketch a use case diagram according to a given problem description (our sample solution depicted five use cases). As second task, participants had to create a type for at least one instance of every distinct symbol As a last task, participants had to sketch a second use case diagram from a problem description of similar size, but this time using their predefined symbols whenever possible. For the tasks, we asked participants to think aloud. Interaction between us and the participants was reduced to a minimum, but participants were allowed to ask questions. We recorded our observations. In the debriefing, we asked about the usability and utility of the tool. Questions were based on the IBM Computer Usability Satisfaction Questionnaire [10].

The briefings lasted from five to ten minutes, working with the tool took between 20 and 30 minutes. Debriefings lasted from 30 minutes to one hour.

5.2 Results

Qualitative feedback from the participants and our observations revealed that they were able to draw use case diagrams with our tool. In debriefing meetings participants told us that they liked the informality provided by the tool. Furthermore, it took them a very short amount of time to learn how types are assigned to symbols and how the container with the symbol library is used (RQ 2).

When we asked about the symbol library that allows for the re-use of defined symbols, 14 out of 17 participants said that they like the feature and that they perceive it to be more or equally efficient than sketching by hand (RQ 1). Three participants argued that use case diagrams consist of simple symbols easy enough to sketch by hand each time. But most participants said that even if the re-use mechanism is slower, they prefer it over sketching for the reason that all instances of the same symbol look exactly identical.

Almost all participants stated that it is still a bit faster to sketch with pen and paper, but the additional functionality of the tool compensates for it (RQ 1). Most frequently mentioned advantages over paper and pencil were: the manipulation of symbols, the re-use of symbols, the save/load option, and the (not yet implemented) possibility to export the sketches to other programs.

The following features were also liked and mentioned frequently: (1) the possibility to draw anything, (2) sketched lines between symbols turning into straight connections, (3) the use of different colors to distinguish symbols, (4) the option to merge two drawn symbols into a single one, and (5) the text autocompletion feature provided by the OS.

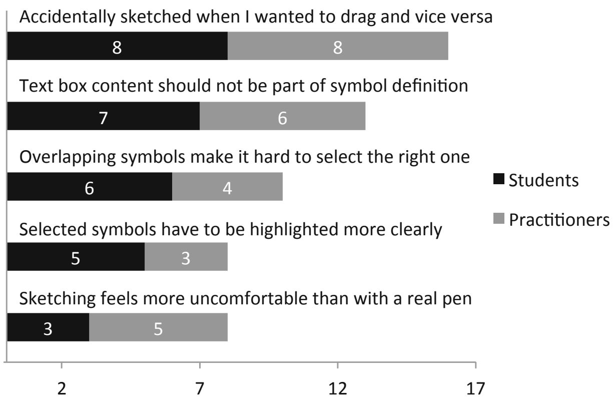

Fig. 4. The 5 most frequently occurred usability problems

In the following we discuss the most frequently mentioned usability issues (see also Fig. 4). Some of the issues are related to our design decisions and need re-thinking, while others are relatively easy to solve in future tool versions.

Sketching versus dragging: We do not provide two distinct modes for drawing and the manipulation of symbols. Participants often painted over existing symbols when they wanted to drag the symbols instead. Few said it actually makes more sense that the symbols have to be selected before they can be dragged around, but they continued to make the mistake.

Text boxes in symbol definitions: When users define symbols that contain text boxes, the text boxes - including the previously entered text - become part of the definition. This makes it possible to define, e.g. a class symbol with three text boxes. But participants said that it takes too much time to change the text

Table 1. The 8 most frequently mentioned feature wishes

	Students (8)	Prac. (9)	Total (17)
Larger screen (flip chart, whiteboard)	5	6	11
Symbol grouping for faster editing	5	5	10
Automatic beautification of symbols	5	5	10
Zoom / Scaling Function	3	7	10
Unlimited scrolling (larger canvas)	6	3	9
Resizing of drawn symbols	6	3	9
Landscape format	3	5	8
Ability to define types of associations	4	4	8

of the text boxes: copies of defined symbols should come without, or with empty text boxes.

Overlapping symbols: Multiple taps on overlapping symbols successively select each symbol in turn. But participants told us that they did not figure this out and therefore had problems selecting symbols overlapped by other symbols.

Selection highlighting: Participants reported that the visual cue for highlighted symbols was not strong enough. As implication, they accidentally manipulated the wrong symbol, especially when symbols overlapped.

Sketching does not feel natural enough: Indeed, participants mentioned that sketching does not feel the same as with paper and pencil. The tip of the used stylus is wider than a normal pen, and the tablet has a tiny, but noticeable lag before it displays the drawn strokes. This is due to hardware characteristics of the tablet.

There are several feature wishes that participants mentioned a couple of times and therefore got a high ranking (see Table 1).

Larger screen: The wish for a larger screen (in the size of a flip chart or whiteboard) made it to the top. Related to this are the following wishes (see Table 1): a *zooming or scaling function*, *unlimited scrolling*, *resizing of drawn symbols*, and the ability to turn the tablet to use it in *landscape format.*

Symbol Grouping functionality: More than 50% of the participants would like to have a grouping function, so that they can manipulate a group of symbols at the same time (e.g. to move them around or to delete them).

Beautification: 10 out of 17 participants wish to have an advanced beautification function. In particular, they were suggesting to have a feature that beautifies symbols based on their geometry. For example, a hand drawn rectangle should be displayed with straight lines, and a hand drawn oval should get smooth curves.

Different types of associations: Participants also stated the missing ability to define different types of lines, i.e. associations.

Further feature wishes include an undo feature, standalone text boxes and a distinct feature to add commentary text to the sketches.

5.3 Findings

With this experiment, we investigated whether the FlexiSketch tool can be used to draw use case diagrams, and whether it supports ad-hoc and flexible sketching.

Furthermore, we investigated whether users are able to define the individual elements of a use case diagram by assigning types to symbols (RQ 2).

All participants confirmed that they could draw the diagrams corresponding to the given problem description. They liked how the tool allows for freeform sketching. While observing the participants during the modeling task, we noticed three problems that delayed the completion of the task. These issues were also mentioned by participants.

First, in several cases participants accidentally drew a stroke when they wanted to drag a symbol. We will have to investigate whether this problem disappears once selected symbols get highlighted more clearly (which was also a mentioned critique). Therefore, we recently implemented a blue-colored background for selected symbols. We believe that users will associate the color with draggable symbols and will stop trying to drag symbols that are not colored blue. This is an assumption that yet needs to be evaluated. If the assumption does not hold, we have to rethink our design decision of not having two alternate modes for drawing and the manipulation of symbols.

Second, the screen size of the tablet computer proved to be limiting. Many participants lost time due to rearranging symbols, and they asked for a larger screen or an enhanced scrolling function. This seemed to be the biggest issue regarding adoption in practice.

Despite these issues, participants stated that drawing is easier and faster than with any other software modeling tool they know. Furthermore, we found the resulting diagrams to be well readable.

Participant feedback showed that they had no problem with defining the individual symbols (RQ 2). Some of them even started to define the symbols right away although we only asked them to draw a use case diagram as the first task. Around 50% of the participants, mainly students, asked how to make copies of symbols. Defining them is the only way in the current prototype. This might have been an additional motivation for the participants to do so.

Participants frequently mentioned the lack of certain tool features being a disadvantage, although physical media like paper and pencil do not have these features either (RQ 1). It seems that for paper and pencil, most people unconsciously accept the lack of symbol manipulation features. But they expect them to be present in a software tool. Participants stated that for them it felt a bit slower to draw with a software tool, thus a tool must compensate this by providing additional features. They mentioned that, once the formalization or export function is available, the whole process will be faster with our tool compared to paper and pencil.

6 Experiment #2: Utility of the Approach

This experiment gathered data about why and what kinds of diagrams practitioners sketch in their daily work, and assessed the utility of our approach. The experiment was conducted in a controlled setting with the nine practitioners from the first experiment. Thus, we refer to Section 5 for demographic information about the participants.

6.1 Method

The experiment was done with each participant separately, and included three parts: an interview, the evaluation and a debriefing. We started with questions about (1) how participants use sketches in SE practice, (2) what kind of diagrams they sketch, and (3) how these sketches are re-used later on.

The participants were then asked to sketch an example diagram of a diagram type they use frequently in their daily work, and to think aloud while using the prototype. Like in the first experiment, communication was reduced to a minimum, and we recorded our observations. After the participants completed their sketches, they had to define one instance of each occurring symbol. We stored the resulting symbol library and asked the participants to sketch another diagram of the same kind, re-using the previously defined symbols whenever possible.

At the end we asked debriefing questions about the usability and utility of the prototype, and how participants would like to further use and process the drawn sketches.

The interview, the briefing, and the debriefing together lasted for 30 to 40 minutes. Working with the tool took between 20 and 30 minutes.

6.2 Results

Qualitative feedback from the participants was very encouraging. All but one practitioner told us that they know situations where a polished version of the tool would be useful to them in practice (RQ 3). Two participants explicitly asked us to keep them up to date regarding our research and provide a more advanced tool version.

When we asked participants whether they use paper and pencil, and whiteboards in their work, they all answered yes. Whiteboards and flip charts were mentioned to be more important than paper and pencil.

We also wanted to know from participants what kinds of diagrams they draw in practice on whiteboards and paper (RQ 3). Table 2 summarizes the answers. Participants answered that they do not just use certain diagram types, but they draw *"something similar to, but not quite a..."* or *"a simplified version of a..."* followed by the diagram type. Regarding UML models, use case diagrams were mentioned because they are *"something that stakeholders understand"*. Many participants said they draw diagrams to show some kinds of processes, e.g. business processes, and to show structures and relationships or transactions between stakeholders and systems, or between systems and subsystems. Two participants emphasized that they draw different kinds of diagrams dependent on their stakeholders' knowledge.

When we asked for the reason of using whiteboards or paper, the most frequent answer was that the large size of whiteboards and flip charts facilitates communication and reaching consensus (to reduce misunderstandings) in meetings and workshops (see Fig. 5). Participants also found it important to mention that whiteboards allow for collaboration and encourage attendees to participate.

Table 2. Diagrams drawn on whiteboards and paper as reported by participants

- Process models (block diagram with events and activities, BPM)
- Business models (clients, orders, storage dimensions)
- Mindmaps
- Flowcharts
- Architecture diagrams (layer cake)
- Use Case Diagrams
- System architecture models (dependencies between modules)
- Boxes and arrows
- Simplified activity diagrams (Boxes connected with signals)
- Transactions (systems and flows of information)
- Entity diagrams
- No particular diagram type, dependend on stakeholders
- Sequence diagrams

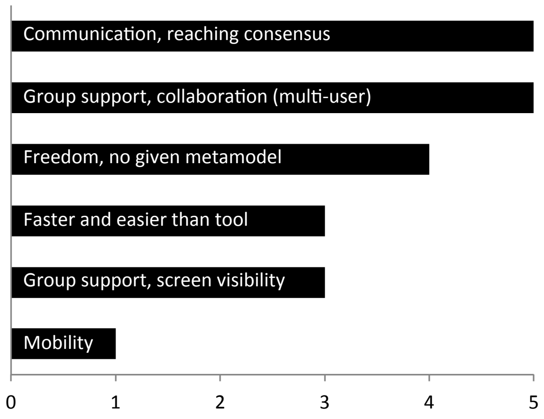

Fig. 5. Most frequently stated reasons why paper or whiteboards are used

The next reason for using physical media is that it supports freeform sketching. Participants mentioned that they find it faster and easier to draw on physical media compared to software tools. They stated that a large screen is preferred because many people can look at it at the same time.

We asked participants how the sketches drawn on physical media are re-used later on. Four participants said that they take a photograph and later re-create a model in another software tool using the photo as reference. Three participants stated that the content from the sketches is either communicated verbal, or a certain kind of documentation is created. Two participants usually take a photograph and put the picture directly into another document, where they add descriptions to it. Also, two participants told that in many cases the sketches only remain valid for some weeks, and are not used anymore afterwards (as they cannot be edited).

After the interview, participants used our tool to draw simple examples of their most used diagram types (RQ 3). Feedback about the usability and utility was similar to the one from the first experiment. *"A larger screen is needed"* was one of the most frequently given answers when we asked about their willingness to use an improved version of the tool in practice. Apart from this and the already mentioned usability issues, they were positive about a possible adoption in practice.

After participants have drawn diagram examples, we asked them what features exactly they would expect from an export function. Here, the answers were quite diverse: some just want to be able to distribute it. Some want to have beautified versions of the sketches, while others are happy with the sketchy look. Most of the participants want to have it in an editable form (either beautified or not). One participant would like to export a list of the sketched entities into MS Excel. Another participant wants to work iteratively on the sketches, having the option to synchronize them back an forth between our tool and another software tool.

6.3 Findings

With this experiment, we investigated what practitioners usually sketch in practice, whether these sketches can be drawn with the FlexiSketch tool, and whether practitioners like our approach, i.e. would be willing to adopt our approach in practice. We also wanted to see whether they can classify the symbols they draw into different types.

We conclude that participants liked our approach as it suits the kinds of diagrams they draw (RQ 3): no standard diagrams, but something similar. Participants also frequently said that they draw UML-like diagrams, introducing their own notation.

Participants were also able to categorize the drawn symbols by assigning types (RQ 2). We were afraid that users might not be able to handle even this first step of lightweight metamodeling, but some participants actually wanted to do even more. For example, they asked how they could define different types of associations, constraints, and how to declare a type as subtype of another one. However, other participants did not bother about metamodeling. This highlights the diversity of user skills which our approach should account for.

We probably have to trade the mobility of tablet computers against a larger screen (e.g. an electronic whiteboard) to gain a wider adoption in practice. Electronic whiteboards allow for multi-user input and thus facilitate collaboration, but they are far less wide-spread than mobile devices. Results suggest that the ability of a sketching tool to facilitate communication is important. This corresponds to findings from Ossher et al. [3].

When it comes to the question whether we should allow for an easier formalization of sketches by limiting the freeform sketching, participants agree that the freeform drawing is more important than formalization capabilities.

7 Threats to Validity

Construct Validity. We concluded that users are able to assign types to symbols. Depending on how we intend to transform informal sketches into semi-formal models, it might not be valid to state that users are able to provide relevant metadata if they are able to assign types. So we might be measuring what we mean to measure in special cases only.

Internal Validity. Participants voluntarily assigned types to symbols, as it was the only way to make copies of a symbol. This additional benefit can be seen as a motivating factor to define symbols, and therefore is a threat to internal validity.

Conclusion Validity. We did not try to measure a relationship between some treatment and some outcome, nor did we try to calculate statistical significance in our data. We were rather interested in assessing the feasibility of our approach and the utility of our tool.

External Validity. Our goal was to gather qualitative rather than quantitative data. However, the small sample size of our experiments is a threat to external validity. Many of the practitioners in our experiments have an academic background, and might therefore be more enthusiastic about our approach than software engineers in general. In the second experiment, participants decided to draw rather simple and high-level sketches because of time constraints and screen size limitations. Although these sketches contained key elements of drawings they use in real-world projects, this can be seen as a threat to external validity.

8 Related Work

Various mobile software tools for freeform sketching can be found online (e.g. Developer Whiteboard [11]). There are also modeling tools for both general-purpose and SE-specific modeling. For example, DroidDia [12] allows to create different diagrams like flow charts, Venn diagrams, mind maps, and so on. It provides predefined symbols as well as basic geometric shapes. Although mobile computing is a recent trend in SE (e.g. [13]), we are not aware of any existing work about mobile software tools that let users define and annotate their own diagram symbols. We fill this niche with our FlexiSketch approach.

Looking at desktop tools for sketch-based modeling support in SE, there are two threads of related approaches: (1) augmenting formal modeling tools with sketch recognition features, and (2) augmenting informal modeling tools (e.g. a tool mimicking a whiteboard) with features that allow a certain degree of formalization.

In thread 1, various approaches and tool prototypes have been developed that allow users to sketch diagrams (e.g. [14,15]). The key idea is that sketch recognition algorithms will convert the produced sketches into semi-formal models. However, this also means that a user, when drawing a sketch, is still following the tool's predefined notations. Users therefore have to understand a specific modeling language in order to produce sketches that can be converted automatically. The sketch recognizer cannot interpret sketches that do not adhere to the language. Therefore, these approaches limit creativity and expressiveness. They also distract users from the actual modeling task [16] and can cause additional overhead due to sketch recognition errors.

Approaches following thread 2 seem to be more promising. However, there are several challenges. For example, the set of possible sketches increases as soon as the user gains freedom in sketching. Thus, it gets more difficult for a tool to analyze and identify what the sketches actually mean. So far, only few researchers have tackled these issues and built tool prototypes that support users in drawing sketches independently of a specific modeling language. Such an example is the Calico prototype [17]. It provides mechanisms to structure sketches into different parts. Furthermore, a user can connect the parts with arrows. However, the user is not able to assign some meaning to the sketched symbols.

Other tools such as MetaEdit+ [18] and MaramaSketch [19] include metamodeling editors. These editors allow to define a custom modeling language. The language definition is then used to compile a modeling tool for the defined language. However, these tools require to create the full language definition first and then users must strictly adhere to it, thus preventing any flexible sketching.

Some approaches do not require users to define symbols at the beginning. For example, BITKit [3] is a flexible modeling tool that focuses on combining the advantages of office and modeling tools. However, it does not include freeform sketching. The Electronic Cocktail Napkin [1] is a freehand drawing environment for conceptual design, and allows to incrementally transform sketches into schematic drawings. It does not focus on SE, but on architectural design (e.g. buildings). The open source project Sketch for Eclipse [20], currently under development, is an API that provides sketching capabilities, a trainable gesture recognizer, and allows users to classify sketched elements. However, the project does not focus on user metamodeling. Since it is meant to be an API, it is not suited for end-users.

In summary, there is no existing solution that allows ad-hoc modeling and satisfactorily bridges the gap between freeform sketches and semi-formal models.

9 Conclusion and Future Work

In this paper we report on a tool-supported solution that combines freeform sketching with the ability to interactively annotate the sketches for an incremental transformation into semi-formal models. Software engineers using our approach have the possibility to evolve sketches into semi-formal models rather than re-modeling the information contained in the sketches in a software modeling tool. Our experiments provide first answers to our research questions: RQ 1: What are the differences between our tool and paper and pencil? User feedback indicated that sketching with FlexiSketch "feels" less natural and slower compared to paper and pencil, but additional functionality provided outweighs these limitations. RQ 2: Can users in general classify the elements they draw into types? Participants in our experiments had no problems doing so, therefore we answer this question with yes. RQ 3: Would users consider adopting our tool in practice? All but two participants were positive about adopting the tool in practice. We cannot generalize this result for a larger user-base, so we answer with a tentative yes.

The major contributions of this paper are:

- A new, flexible process for sketching and the step-wise formalization of these sketches. The core of our approach is not specific to the SE domain, but can also be valuable for many other domains.
- A tool prototype that highlights the feasibility of our approach. Users can sketch first and assign meanings to the sketches on demand.
- Two initial usability and utility experiments showing that software engineers are able to sketch their favorite diagrams with our tool, and can annotate them.
- Insights about the use of informal sketching approaches in SE practice.

Recent technological trends and the growing market of tablet PCs provide the opportunity to come up with novel solutions supporting flexible, mobile, ad-hoc modeling in SE. Our work describes one possible approach how tool support for early SE phases may look like.

Although first evaluations indicate the usefulness of our approach, more research is needed to gain answers to open questions. Our future research will focus on the following:

Hardware for Sketching. Modern tablet computers usually come with a capacitive touch screen. Digital pens must have wide tips in order to get properly recognized by the tablets. For some users, sketching with these pens feels unnatural. We need to investigate how we can make sketching feel more natural. Moreover, the small screen sizes of tables are an issue. Electronic whiteboards provide a large drawing space and facilitate collaboration, but limit end-user mobility and are far less pervasive than mobile devices. At the moment, a native Java version of FlexiSketch that runs on desktop machines and electronic whiteboards is planned, but our main focus remains on supporting SE activities with mobile devices.

End-user Lightweight Metamodeling. It is an open challenge how metamodeling can be made accessible to end-users [3]. Assigning types to symbols is only a first step towards a lightweight, end-user metamodeling approach. It is thus worthwhile to investigate how a more complete end-user lightweight metamodeling method could look like, and how it can be integrated into our approach. We want to explore how much information relevant for metamodeling is put into model sketches by users themselves, how much metamodeling is needed for exporting models into other software tools, and whether the tool could infer the missing information (semi-)automatically.

Field Studies. So far, we performed interviews and two controlled experiments. Once an improved version of our tool is ready for use in practice, we have to conduct case studies where our tool is used in the field during real-world projects. This allows to further assess the utility and adoption in practice of our approach.

We want to explore in which situations our tool can be used, and how it will be used. The studies will point out the benefits and limitations of our approach in more detail, and will enable us to refine it.

Acknowledgment. The authors would like to thank Sebastian Golaszewski, who made a big contribution to the FlexiSketch tool.

References

1. Gross, M.D., Do, E.Y.-L.: Ambiguous intentions: a paper-like interface for creative design. In: Proc. 9th ACM Symp. on User Interface Software and Technology, pp. 183–192 (1996)
2. Branham, S., Golovchinsky, G., Carter, S., Biehl, J.T.: Let's go from the whiteboard: supporting transitions in work through whiteboard capture and reuse. In: Proc. 28th Int. Conf. on Human Factors in Computing Systems, pp. 75–84. ACM, New York (2010)
3. Ossher, H., Bellamy, R., Simmonds, I., Amid, D., Anaby-Tavor, A., Callery, M., Desmond, M., de Vries, J., Fisher, A., Krasikov, S.: Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges. In: Proc. ACM Int. Conf. on Object Oriented Programming Systems Languages and Applications, pp. 848–864 (2010)
4. Wüest, D., Glinz, M.: Flexible Sketch-Based Requirements Modeling. In: Berry, D., Franch, X. (eds.) REFSQ 2011. LNCS, vol. 6606, pp. 100–105. Springer, Heidelberg (2011)
5. Wüest, D.: Bridging the gap between requirements sketches and semi-formal models. In: Doctoral Symposium of the 19th IEEE Int. RE Conf. (2011), `http://dx.doi.org/10.5167/uzh-55675`
6. Cherubini, M., Venolia, G., DeLine, R., Ko, A.J.: Let's go to the whiteboard: how and why software developers use drawings. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems, pp. 557–566 (2007)
7. Coyette, A., Schimke, S., Vanderdonckt, J., Vielhauer, C.: Trainable Sketch Recognizer for Graphical User Interface Design. In: Baranauskas, C., Abascal, J., Barbosa, S.D.J. (eds.) INTERACT 2007. LNCS, vol. 4662, pp. 124–135. Springer, Heidelberg (2007), `http://dl.acm.org/citation.cfm?id=1776994.1777013`
8. Moody, D.L.: The method evaluation model: a theoretical model for validating information systems design methods. In: Proc. 11th European Conf. on Information Systems, pp. 1327–1336 (2003)
9. Moody, D.L., Sindre, G., Brasethvik, T., Sølvberg, A.: Evaluating the quality of information models: empirical testing of a conceptual model quality framework. In: Proc. 25th Int. Conf. on SE, pp. 295–305 (2003)
10. Lewis, J.R.: IBM computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. Int. J. Hum.-Comput. Interact. 7, 57–78 (1995)
11. Developer Whiteboard, `https://play.google.com/store/apps/details?id=com.agilaz.whiteboard`
12. DroidDia, `http://www.droiddia.com/`
13. Seyff, N., Graf, F., Maiden, N.: Using mobile RE tools to give end-users their own voice. In: Proc. 18th IEEE Int. RE Conf., pp. 37–46 (2010)
14. Chen, Q., Grundy, J., Hosking, J.: SUMLOW: early design-stage sketching of UML diagrams on an e-whiteboard. Softw. Pract. Exper. 38(9), 961–994 (2008)

15. Hammond, T., Davis, R.: Tahuti: a geometrical sketch recognition system for UML class diagrams. In: AAAI Spring Symposium on Sketch Understanding, pp. 59–68 (2002)
16. Alvarado, C., Davis, R.: SketchREAD: a multi-domain sketch recognition engine. In: Proc. 17th ACM Symp. on User Interface Software and Technology, pp. 23–32 (2004)
17. Mangano, N., Baker, A., Dempsey, M., Navarro, E., van der Hoek, A.: Software design sketching with Calico. In: Proc. IEEE/ACM Int. Conf. on Automated Software Engineering, pp. 23–32 (2010)
18. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. In: Constantopoulos, P., Mylopoulos, J., Vassiliou, Y. (eds.) CAiSE 1996. LNCS, vol. 1080, pp. 1–21. Springer, Heidelberg (1996)
19. Grundy, J., Hosking, J.: Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool. In: Proc. 29th Int. Conf. on SE, pp. 282–291 (2007)
20. Sangiorgi, U.B., Barbosa, S.D.: Sketch: modeling using freehand drawing in Eclipse graphical editors. In: ICSE 2010 Workshop on Flexible Modeling Tools, Cape Town, South Africa (2010), http://www.ics.uci.edu/~tproenca/flexitools/papers/10.pdf