Undo and Erase Events as Indicators of Usability Problems

David Akers[1,2], Matthew Simpson[2], Robin Jeffries[2], Terry Winograd[1]

[1] Stanford University HCI Group
Computer Science Department
Stanford, CA 94305
{dakers,winograd}@cs.stanford.edu

[2] Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
{msimpson, jeffries}@google.com

ABSTRACT
One approach to reducing the costs of usability testing is to facilitate the automatic detection of *critical incidents*: serious breakdowns in interaction that stand out during software use. This research evaluates the use of *undo and erase events* as indicators of critical incidents in Google SketchUp (a 3D-modeling application), measuring an indicator's usefulness by the numbers and types of usability problems discovered. We compared problems identified using undo and erase events to problems identified using the user-reported critical incident technique [Hartson and Castillo 1998]. In a within-subjects experiment with 35 participants, undo and erase episodes together revealed over 90% of the problems rated as severe, several of which would not have been discovered by self-report alone. Moreover, problems found by all three methods were rated as significantly more severe than those identified by only a subset of methods. These results suggest that undo and erase events will serve as useful complements to user-reported critical incidents for low cost usability evaluation of creation-oriented applications like SketchUp.

Author Keywords
Usability testing, critical incidents, undo, erase, user-reported critical incident technique, Google SketchUp.

ACM Classification Keywords
H5.2. Information interfaces and presentation (*e.g.*, HCI): User Interfaces - Evaluation/methodology.

INTRODUCTION
Traditional laboratory usability testing [21] can only reveal certain aspects of a product's usability. In particular, usability research goals and product characteristics sometimes dictate the recruitment of a large pool of participants, which can be prohibitively expensive if the organization employs a traditional testing methodology.

One way to address this challenge is to facilitate the automatic identification of particularly troubling interaction episodes encountered during usability testing. Usability researchers refer to such episodes as *critical incidents*. There are two classes of approaches for identifying critical incidents without requiring the active attention of a usability expert: *event-based reporting* (in which algorithms attempt to discover potential critical incidents in usage log data), and *self-reporting* (in which the participants themselves identify their own incidents). Each class of approaches has its benefits and drawbacks. This paper compares two event-based approaches (where the events of interest are *undo* and *erase)*, with a self-reporting approach (the user-reported critical incident technique [9, 10]).

The test application for our approach, Google SketchUp [6], is a freely-available 3D modeling application with large-scale distribution around the world. It is used by architects and interior designers as part of their professional modeling workflows, and also by casual users with an interest in 3D modeling. SketchUp is intended as a playful conceptual design tool, although it can also be used for precise modeling of 3D objects.

SketchUp's wide range of uses and its unstructured workflow present unique challenges for event-based critical incident detection. Some event-based approaches seek to explicitly model expected user behavior (*e.g.*, [11]), detecting and reporting when users deviate from the predictions of the model. With SketchUp, however, it is difficult to imagine how one would construct such a model of expected behavior. There are simply too many different

3D objects that someone might choose to build, and too many ways of building them.

This motivates an alternate approach: rather than modeling successful interactions, we attempt to model *unsuccessful* interactions directly. In applications like SketchUp, attempts to correct mistakes represent natural indicators of unsuccessful interactions. We chose to focus on two such indicators: undo and erase events. We chose these particular indicators because we knew that they occur frequently during use, and we could measure their occurrence without having to modify the source code to SketchUp.

How useful are undo and erase events as indicators of usability problems in SketchUp? Knowing that both undo and erase are frequent operations that often indicate recovery from errors, it seemed reasonable to hypothesize that focusing on these events would reveal some of the same usability problems self-reported by users. However, we also knew that some usability problems (for example, problems related to feature discoverability) would be unlikely to produce undo or erase events as symptoms. Furthermore, we knew that some undo or erase events could lead to *false alarms* – they might sometimes indicate epistemic actions [14] rather than usability problems.

To assess the value of focusing on erase or undo, we ran a within-subjects laboratory experiment. We compared the numbers and characteristics of usability problems found by three types of indicators: undo events, erase events, and user-reported critical incidents. Results from 35 participants indicate that undo and erase episodes together revealed over 90% of the problems rated as severe, several of which would not have been discovered by self-report alone. Moreover, problems found by all three methods were rated as significantly more severe than those identified by only a subset of methods. These findings suggest that undo and erase events will serve as useful complements to user-reported critical incidents, for low cost usability evaluation of creation-oriented applications like Google SketchUp.

RELATED WORK

This research is motivated by prior work that has demonstrated the need for usability testing methodologies designed to support the testing of large numbers of participants in parallel. A large participant pool may be warranted when the goals require statistically significant results, when it is necessary to sample a variety of expertise levels [16], when users are allowed the freedom to choose their own task goals [22], or when there are many ways for users to accomplish the same goals (as with SketchUp).

Our approach is a derivative of the critical incident technique [5], in which significant events during interaction (positive or negative) are collected and analyzed by trained observers. (We chose to focus on negative incidents in this paper, since they are more likely to indicate usability problems.) del Galdo *et al.* [4] adapted the critical incident technique for use in HCI, and Winograd and Flores independently developed the theory of user breakdowns [24]. Hartson and Castillo devised the user-reported critical incident technique [9, 10], in which the users of the system are responsible for detecting and describing their own critical incidents as they occur. While their technique was designed for use in remote situations, some of their studies were done in the lab. The comparison of our approach with this technique forms the central theme of this paper.

Capra [1] developed and evaluated an augmented retrospective variant of the user-reported critical incident technique, finding it to be similarly effective to a contemporaneous reporting strategy. In this variant, researchers showed participants a video replay of their entire session, asking them to detect and describe critical incidents as they observed them in the replay. Our own implementation is a hybrid implementation of [9, 10] and [1], separating the detection and description phases. Participants detect critical incidents contemporaneously, but description is delayed until a retrospective phase (in which we prompt them with screen capture video centered about each detected incident). We chose this approach because it allowed us to compare self-reported incidents to those detected by undo and erase (for which a hybrid approach is the only reasonable option).

One difficulty Hartson and Castillo found with their first implementation of the user-reported critical incident technique was that users often initiated the reporting long after they experienced a problem [10] (making it more difficult to link the reported incident to contemporaneous video or other context). The solution described in their paper was to decouple the detection and description of incidents, which is exactly what our own implementation does by combining contemporaneous detection of incidents with retrospective description.

A legitimate concern for any usability evaluation method that relies extensively on retrospective commentary is whether important information is lost due to the fallibility of human memory. We are encouraged by results from a recent eye-tracking study [8]. In an evaluation of the stimulated Retrospective Think Aloud method, Guan *et al.* compared participants' original eye movements to their retrospective verbalizations, and found strong correlations. They concluded that stimulated Retrospective Think Aloud was both valid and reliable for reconstructing an account of a user's attention during a task.

Finally, our focus on undo and erase events is closely related to a case study described by Swallow *et al.* [23]. They instrumented a direct-manipulation visual builder application to record log data for a variety of indicators of critical incidents, including undo and erase. However, this study focused mainly on the "hit rate" for each indicator (the rate at which each indicator revealed actual usability problems, and not merely false alarms). They performed no severity analysis of problems, and did not compare their methods with self-reporting, as we do in this paper.

EXPERIMENTAL METHODOLOGY

This section describes the design of the experiment we ran to compare problems detected by erase and undo events to those detected by user-reported critical incidents. A series of three pilot experiments totaling 27 participants informed the design of the final experiment. The results of these pilot studies will be described as they relate to the design that we arrived at for the final experiment.

Participant Recruitment

Our goal in recruitment was to attract participants of a variety of backgrounds and SketchUp expertise levels. This variety increased the generality of the study, and made inter-group comparisons possible. We recruited a total of 70 participants, including both the pilot study and the formal experiment. Most participants (61) responded to flyers posted at coffee shops and in academic buildings at the University of Colorado at Boulder. To attract a higher percentage of SketchUp experts, we also enlisted 6 employees of Google who are specialists in 3D modeling with SketchUp. The remaining 3 participants were software engineering interns at Google, and participated only as part of the pilot experiments.

Of the 35 participants in the formal experiment, 19/35 (57%) were a near-equal mix of undergraduate and graduate students from the following departments: Architecture (7); Computer Science (4); Civil Engineering (3); Telecommunications (1); Aerospace Engineering (1); Astrophysics (1); Geography (1); and English (1). Of the 16 non-students, 6 were professional 3D modelers, 3 were software engineers, and the remainder had miscellaneous jobs unrelated to SketchUp.

Of the 35 participants, 10/35 (28%) had never used SketchUp before, 9/35 (26%) described themselves as novices with the interface, 9/35 (26%) described themselves as intermediate users, and 7/35 (20%) described themselves as experts. For a 90 minute session, participants were compensated with a $10 gift check, a short-term license to SketchUp Pro, and Google swag.

Experimental Setup

The formal experiment was divided into six 90-minute sessions, each with between 5 and 7 participants who worked in parallel on independent laptops (see Figure 1 for a photograph of the laboratory environment). Each of the laptops was an IBM ThinkPad T61p, with identical software configurations including a development version of SketchUp. Laptops were also equipped with screen capture recording software and dual headsets with microphones (for the paired retrospective session, described later).

We also instrumented SketchUp so that it could record time-stamped occurrences of undo and erase events, and added a button for participants to report critical incidents. We implemented these extensions as plug-ins using SketchUp's embedded Ruby API, thus avoiding having to make any modifications to the source code to SketchUp.

Experimental Protocol

The 90 minute experiment was divided into the following sections: Training in SketchUp (15 minutes), Training in Identifying Critical Incidents (20 minutes), Practice (10 minutes), Modeling Task (15 minutes), and Retrospective Commentary (30 minutes).

Training in SketchUp (15 minutes)

To familiarize everyone with SketchUp, participants watched three short new-user training videos [7]. The three videos we showed were: "New Users 1: Concepts," "New Users 2: Drawing Shapes," and "New Users 3: Push/Pull." Participants were encouraged to take notes.

Training in Identifying Critical Incidents (20 minutes)

To ensure that participants were adequately trained in reporting critical incidents, we gave extensive instructions and examples of incidents. For the purposes of a fair comparison, we tried to mimic the style and content of training described by Hartson *et al.* [10]. We did make several changes motivated by our first pilot experiment. In this experiment (n = 12), we observed that participants seemed less likely to report problems when they attributed the problems to themselves (rather than the software). We adjusted our instructions to emphasize that we were testing the software and not the participants. We also decided to refer to critical incidents as "interface issues," hoping that the more neutral terminology would encourage reporting.

Practice (10 minutes)

Participants were given 10 minutes to practice using SketchUp and reporting critical incidents. Participants were told to explore interface features and build whatever they wanted during these 10 minutes.

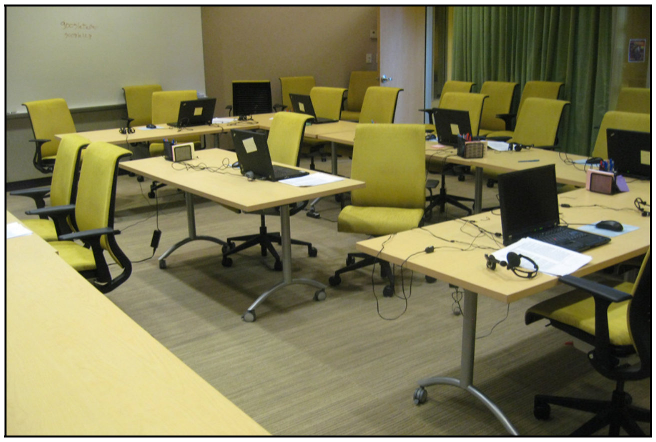

Figure 1: The experimental setup for our laboratory study. Seven laptops were identically configured with SketchUp. Participants worked in parallel; their actions were logged, and their screens were recorded. Notice that there were two chairs and headsets next to each computer, to facilitate paired-participant retrospective commentary sessions.

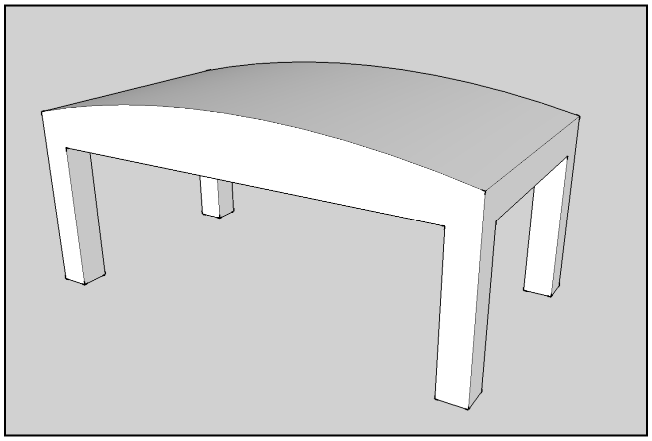

Figure 2: The "bridge" task. Participants had to draw this model in SketchUp, and report any critical incidents as they worked. We asked them to ensure that all four legs were the same height and shape. If they finished early, they were asked to resize their bridge to 5 ft. x 5 ft. and make three copies of it, laying them end to end.

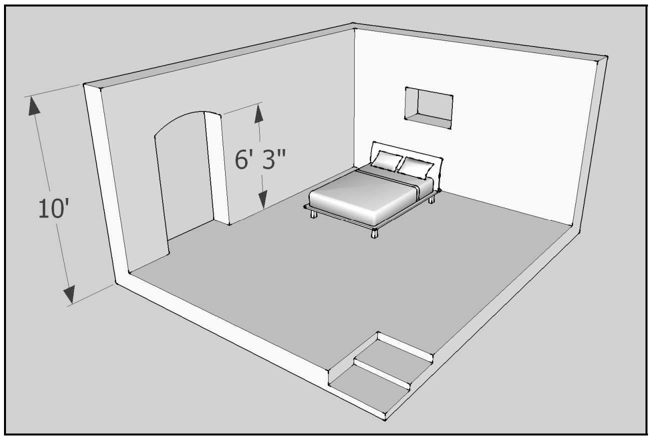

Figure 3: **The "room" task.** Participants were asked to draw this model of a room in SketchUp, and report any critical incidents as they worked. We asked them to ensure that the room was 10 ft. high, and the doorway was 6 ft. 3 in. high. They did not need to model the bed; they could insert it from the "components browser" and position it in the room. If participants finished early, they were asked to modify the bed to form two single beds, each with one pillow. They were also asked to add shadows to their scene.

Modeling Task (15 minutes)

We randomly assigned participants to one of two tasks: some completed the *bridge task* (Figure 2), while others completed the *room task* (Figure 3). Having a second task increased the generality of the study, but time constraints prevented us from giving both tasks to each participant. Each task had two phases; if participants finished the first phase, they could raise their hand and receive printed instructions for the second (more difficult) phase. We intended to keep all participants busy throughout the session, regardless of their expertise level in SketchUp. See the captions for Figures 2 and 3 for descriptions of the specific instructions provided to participants. We deliberately made the goals unambiguous and constrained; if we had allowed participants any flexibility in defining their own goals, then many of their undo and erase operations could have indicated changes of goal rather than usability problems.

We asked participants to report critical incidents as they worked. To report an incident, they simply clicked a "Report an Issue" button. Pressing this button triggered a log message that was written to a file; it had no visible impact to the user. Here we deviated from the approach of Hartson and Castillo [9, 10]. Instead of asking them to fill out a form immediately upon experiencing the incident, we simply marked the incident and allowed the user to continue immediately. After the task was finished, we extracted video episodes around each marked event and asked them to reflect on the episodes (see the next section on "Retrospective Commentary"). This was done to minimize disruption to the user, and to facilitate a fair comparison with undo and erase events (for which commentary *must* be collected retrospectively to avoid intolerable disruption).

We purposefully did *not* encourage participants to think aloud as they worked. Our early pilot experiments suggested that it is exceedingly difficult to think aloud while modeling in SketchUp, and thinking aloud appeared to greatly interfere with the ability to model successfully. The thought process behind using SketchUp is likely non-verbal: SketchUp users are forced to think spatially: in 3D shapes and orientations – not words. Asking them to comment retrospectively rather than concurrently at least allows them to focus their attention on putting their thoughts into words.

Retrospective Commentary (30 minutes)

Immediately following the completion of the task, we processed the video to extract 20 second episodes centered around each undo, erase, and self-reported incident. If two episodes would have overlapped (*e.g.*, a user repeatedly used undo), then we merged the episodes to form a single longer episode, identifying each individual event with a large caption in the video. We showed them the video episodes in a VCR-style interface, and automatically prompted them with questions about each episode. They answered the questions by speaking into their headsets, clicking on a button to indicate when they had finished answering each question. Since we asked different questions for each event type, we showed all of the episodes of each event type together, rather than interleaving them. To avoid confounding our results, we fully counterbalanced the order of the event types.

The first three questions were common to all three event types:

1. Please describe the events that led you to [undo/erase/report an issue]. Focus your answer on recounting a "play-by-play" of what you were thinking and doing at the time. If you can't remember, just say so and move on to the next episode.

2. In the events leading up to your [undo/erase/issue report], did the behavior of SketchUp surprise you? If yes, explain the difference between your expectations and what actually happened.

3. Did you find a way around the issue? If so, what did you do to get around it?

For undo and erase episodes, we asked two additional questions:

4. Did you report this as an issue?

5. If you did not report this as an issue, why do you think that you didn't?

In our first two pilot experiments, participants answered these questions by themselves, speaking into the microphone alongside other participants who were doing the same in parallel. Often the commentary that resulted was abrupt and awkward, and generally did not help in describing the underlying usability problem. To remedy this, in our third pilot test (n = 10), we tried pairing up participants and asking them to discuss each others' episodes together. The commentary was significantly improved – enough so that we decided it was worth the extra time required.

In the final experiment, each pair of participants was assigned roles as "speaker" and listener". (The participants swapped roles halfway through, and changed computers.) The speaker's job was to watch her own episodes and attempt to answer the prompted questions. The listener's job was to ask questions until the answers were completely clear. The listener, not the speaker, was responsible for deciding when to move on to the next question (by clicking a "Question has been answered clearly" button). In either role, participants were encouraged to use a shared mouse to point at objects rather than their fingers, so that we could capture these references in the screen capture recording.

We formed participant pairs by matching those with the most episodes with those with the least, in an attempt to reduce the variance in time required for each pair to complete the retrospective review. If an odd number of participants were present in a session, the experimenter became the listener for the participant with the most episodes.

DATA PROCESSING AND CODING

This section describes the analysis process employed to extract usability problems from the raw usability data. We experimented with applying the SUPEX method for structured usability problem extraction [2], but found it challenging to adapt these strategies to our combination of sparsely-sampled screen capture video and retrospective commentary. Consequently, our own extraction process was less structured, but roughly followed the process described by Howarth *et al.* [12]: we extracted usability problem instances, and merged these instances to form usability problems. The steps to our process are described below.

Discarding Ambiguous Participant Data

From an original set of 43 participants, we were forced to remove three participants because their microphones failed to work properly. We also needed to remove one more participant because he did not finish his retrospective during the time allotted. We decided to remove three more participants because they had been paired with the experimenter; we would like to run further studies to evaluate how the quality of the commentary might differ in these cases. Finally, we removed one participant because she could not even begin to answer the questions about her episodes. (She was utterly lost in SketchUp. Her problems were of the "how do I use a mouse?" variety.) Removing the data from these 8 participants left 35 participants.

Discarding Unclear Episodes

The 35 participants produced 353 episodes (139 undo episodes, 113 erase episodes, and 101 self-report episodes). Cumulatively, this equates to an average of 10.1 episodes per person, or 0.67 episodes per minute of SketchUp usage. From this initial set of 353 episodes, we needed to discard 25 episodes (8%) because the combination of commentary and screen capture video was not clear enough for the researcher to extract a complete usability problem instance. We discarded an additional 4 episodes (1%) in which the user could not remember enough about the episode to answer any of the retrospective questions.

Discarding False Alarms

From the remaining set of 324 episodes, we identified 64 episodes (19.8%) that contained no identifiable usability problems. We were conservative in identifying such "false alarms"; for example, we did not discard episodes in which the user immediately recovered from a mistake. Almost all (98%) of the false alarm episodes that we discarded were triggered by erase events. (One false alarm was generated by self report, when a participant accidentally pressed the button.) There were two varieties of *erase* false alarms. First, there were episodes in which a user erased an extra cosmetic edge that was a byproduct of the normal modeling process. (This is specific to SketchUp; most other 3D drawing programs do not work this way.) Second, there were episodes in which users created temporary construction lines to help them align multiple pieces of geometry, and then erased these lines when they were finished. Interestingly, this example could never have resulted in an undo operation; there is no way to undo a temporary construction line without also undoing the alignment action that follows it. This seems to be a general difference between undo and erase, and would likely hold true for other applications besides SketchUp.

Identifying Usability Problem Instances

After all of the data preparation steps described above, we were left with 260 episodes. A researcher analyzed these episodes to extract 215 unique usability problem instances. The mapping from episodes to usability problems instances was *many-to-many*, as described next.

Sometimes, a single episode would correspond to multiple usability problem instances. In identifying usability problem instances, we did not distinguish between problems that were found directly by a method, and those that were incidental to the method. For example, if a user experienced some problem, pressed undo because of the problem, and then experienced a second problem unrelated to the first, we would include both problem instances. This process produced 35 additional problem instances.

Sometimes, a single usability problem instance would correspond to multiple episodes. This happened only when multiple episodes overlapped in content. Of course, episodes of the *same* event type cannot overlap in the screen capture video (since otherwise our process would have merged them into a single longer episode). However, grouping the retrospective by event type necessitated that we avoid merging events of different event types. Therefore, it was possible for episodes of *different* event types to overlap in content. When participants saw the same interaction sequence for a second time, their responses to questions during the retrospective session were likely to be terse. ("I have already talked about that; refer to my previous answers.") We resolved such situations as follows: If a problem was mentioned during the commentary for an episode, and that same problem was visible in the video of an overlapping episode, then we counted it as a single problem instance discovered in both episodes. There were 50 pairs of partially overlapping episodes, and 20 triples.

Merging Usability Problem Instances

Next, a researcher merged the 215 problem instances to form 95 unique usability problems. It is critical that we applied a consistent merging strategy across all problems. Merging two problems requires generalization, since no two problem instances are exactly the same. Problem instances may differ along many dimensions: for example, the level of granularity of the problem, the immediate cause of the problem, the circumstances under which the problem occurred, the consequences of the problem, etc. We adopted a conservative merging strategy, merging problems if they only differed superficially. Nevertheless, our merge rate (2.26:1) was higher than we expected. We suspect that this may be due to the degree of specificity of our task goals; different users working on the same task tended to experience the same problems because they were all working toward identical goals.

At this point, we used a Poisson model [19] to estimate the total number of problems (N = 110), given the average probability of a participant finding the average problem (λ= 0.065). These numbers suggest that we probably found over 85% of the problems for the given tasks and methods.

Finally, a researcher wrote descriptions for each of the 95 usability problems. In describing each problem, our primary goal was to record what happened in the episode(s), and what the user said about what happened. Some examples of usability problem descriptions are:

1. **(Found by self-report only; rated as mild severity)** After creating a hole, one user judged the result by what he could see through the hole. Because the background (the other side of the hole) was similar to the material surrounding the hole, he had low confidence in his success and spent 10 seconds making sure that the action had the intended effect.
2. **(Found by undo only; rated as medium severity)** One user experienced difficulty resizing a rectangle with the Move/Copy tool. He said that he was surprised that it distorted into non-rectangular shapes as he dragged on an edge. He expected that SketchUp would remember that this shape was created as a rectangle, and keep that rectangle constraint through the rest of the modeling process. He worked around the problem by reversing his action and redrawing the rectangle in the new shape.
3. **(Found by all three methods; rated as high severity)** Several users experienced difficulty when they tried to copy and paste a rectangle, and align their copy to a point on an existing rectangle. The paste operation automatically triggered a "Move" command on the copied geometry, selecting a particular corner on the copied rectangle as the anchor point for the move. Users could not find a way to "snap" the copied rectangle into alignment with the edges of the target rectangle, since the anchor point did not correspond to any point on the existing rectangle. They could not find a workaround, and ended up with unaligned geometry.

Note that we did not attempt to infer the root causes of these difficulties, which often requires following intricate threads of causal reasoning [15]. Was the training video unclear? Did users' expectations stem from their prior use of other 3D modeling software? In this study, we avoided speculation on such possibilities; our goal was to provide designers and developers with as much information as possible to evaluate the design tradeoffs inherent in addressing the problems.

Coding for Problem Severity

Three independent raters (two expert interface designers for SketchUp and one intermediate user) coded each of the 95 usability problems for severity. Raters evaluated each problem for its estimated *frequency* (rated on a scale of 1-5), and a combination of its *impact* and *persistence* (also rated on a scale of 1-5). Frequency was estimated as the percentage of occurrence in the general population during an average modeling session. Impact was estimated as the time it would take to recover from the problem, while persistence was estimated as the extent to which the problem recurred or was difficult to work-around. The actual scales are shown in Table 1.

Problem impact and persistence:	
(1)	minor annoyance, easily learned or worked around
(2)	bigger problem (at least 3 minutes time lost), but still easily learned or worked around
(3)	minor annoyance, but will happen repeatedly
(4)	bigger problem (at least 3 minutes of time lost), and will happen repeatedly
(5)	showstopper (can't move forward without outside help; data loss; wrong result not noticed)

Problem frequency:	
(1)	problem will be extremely rare (less than 1/100)
(2)	some will encounter (at least 1/100, less than 1/3)
(3)	many will encounter (at least 1/3, less than 2/3)
(4)	most will encounter (at least 2/3, less than 100%)
(5)	everyone will encounter (*e.g.*, startup problem)

Table 1: Problem Severity Rating Scales

Frequency and impact/persistence ratings were added, and the final severity rating was obtained by reducing this sum by one; this produced an ordinal scale from 1 (least severe) to 9 (most severe). We labeled severity as follows: 1-2: mild; 3-4: medium; 5-9: severe (but 9 was never observed).

We divided the 95 problems into three sets: a training set (15 problems), a test set (10 problems), and an independent set (70 problems). Coders used the training set to discuss the severity scales and resolve differences in coding styles. Next, they independently rated the 10 problems from the test set, and then discussed the differences and adapted their ratings. Before the discussion, Cronbach's Alpha [3] was 0.75; after coders adjusted their ratings, it increased to 0.90. Convinced that coders were reaching strong agreement, we instructed them to independently rate the remaining 70 problems. For the final set of all 95 problems (using the adapted ratings from the test set), Cronbach's Alpha was 0.82. To reduce the effect of individual outliers, we chose the median of the three ratings as the severity statistic for each problem. Figure 4 shows a histogram of all severities.

RESULTS

This section describes the numbers and characteristics of usability problems discovered with each method.

Comparison Among Undo, Erase, and Self-Report

We define that a problem is *detected by a method* if at least one participant experienced an instance of the problem, as evidenced by video episodes and retrospective commentary associated with that method (undo, erase, or self report). We define that a problem is *detected by a set of methods* if the above statement is true for each method in the set (even if no particular participant would have contributed evidence of the problem from all of the methods in the set).

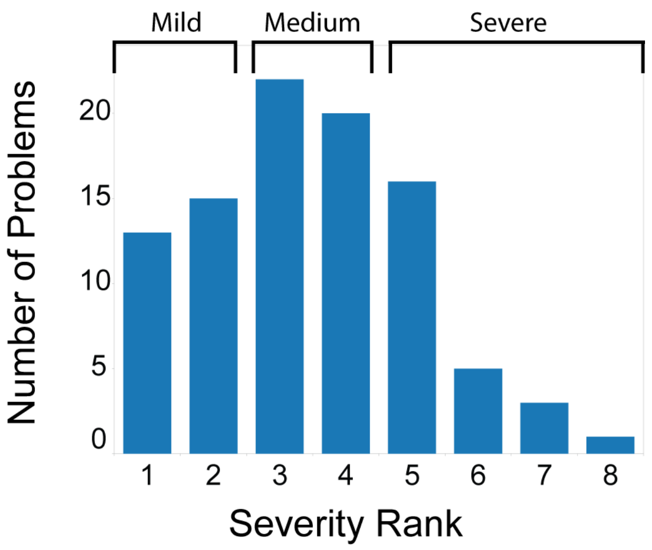

Figure 4: A histogram of the severity rank of problems discovered by any of the three methods. The median was 3, the mean was 3.45, and the standard deviation was 1.63.

The number of problems detected by each method or set of methods is depicted in Figure 5 in a Venn diagram. Problems identified by only one method are non-overlapping, while those that were found by a set of methods are depicted as overlapping with the other methods. Problem counts are shown as numbers within each category; counts of severe problems (those whose severity rating was at least 5) are shown in parentheses. We can conclude from this data that undo and erase combined to detect 74/95 (78%) of all problems. When considering only severe problems, the percentage identified by undo or erase rises to 23/25 (92%).

When considering problems detected by only one method, erase seems to have underperformed relative to undo and self-report. To investigate, we computed the total number of unique problems (both severe and otherwise) discovered by each method for each participant, individually. We then used paired-samples *t* tests to compare each pair of methods. For undo and erase, the difference is significant, $t(34) = 2.73$, $p = 0.01$. For self-report and undo, the difference is not significant, $t(34) = 0.973$, $p = 0.34$. Finally, for self-report and erase, the difference suggests a trend, but is not significant, $t(34) = 1.4$, $p = 0.17$. This last non-significant result may seem surprising, but consider that these statistical tests look at problem discovery on the level of the individual participant. This does not reflect the possibility of detecting the same problem with different methods across different participants.

Correlation between Problem Severity and Method

We investigated how median problem severity varied across methods and combinations of methods, and the results are shown in Figure 6. The median severity rating seems to increase with the number of methods. Each indicator provides independent evidence that a problem is severe; when all three indicators detect a problem, we can be more confident that the problem is not a false alarm. Taking problems as the unit of analysis, we tested for significant differences across all 7 categories using the Kruskal-Wallis test. The result is significant ($\chi^2(6) = 24.74$, $p < 0.001$). We also ran pair-wise comparisons using the Mann-Whitney test, and found significant differences for U vs. S&E&U ($z = 3.874$, $p < 0.001$), E vs. S&E&U ($z = 2.031$, $p = 0.046$), and S vs. S&E&U ($z = 3.373$, $p = 0.001$). All other comparisons were insignificant. Note that the Mann-Whitney test assumes that problems are independent (that they do not tend to co-occur more than they would by chance). Strictly speaking, problems are *not* independent, but the effect size is very large and unlikely to disappear when dependence is factored in. A similar independence assumption appears elsewhere in the literature (*e.g.*, [19].)

Correlations Between Problem Severity and Expertise

We also looked for correlations between the median problem severity and participants' prior SketchUp expertise. New users, novices, and intermediates all had a median problem severity rank of 4, while the median for experts was 2.5. However, the Kruskal-Wallis non-parametric test of the differences across all four groups was not significant, $\chi^2(3) = 2.73$, $p = 0.44$. Although the results were not significant, a possible explanation for the lower problem severity among experts would be that our tasks (even with their second phases) were conceptually easy for the experts in the study. Most of the problems that they encountered were minor nuisances (keyboard typos, etc.)

Reasons Problems Were Not Reported

We were particularly interested to learn something about the problems in the top three sections of the Venn diagram – those that were detected by erase and/or undo, but *not* detected by self-reporting. Why would people fail to report problems, when these problems were detected by other indicators? To begin to answer this question, we assessed the data collected on question #5 in the retrospective session: for erase and undo events that were not reported, why did the participant think that he did not report it? Of those times when people ventured to speculate, the explanations were revealing: 30/52 (58%) said that they did not report the problem because they blamed themselves rather than SketchUp. (This happened despite our repeated attempts to emphasize to participants that they should disregard the attribution of blame.) Another 16/52 (31%) said that the problem was too minor to report. The remaining 6/52 (11%) said that they should have reported it, but simply forgot. While it is easy to draw conclusions from these numbers, we must be cautious not to over-interpret; people are notoriously bad at introspecting about their own high-level cognitive processes [20]. However, we believe that the data combined with the subjective comments warrant further investigation into the reasons people do and do not report problems.

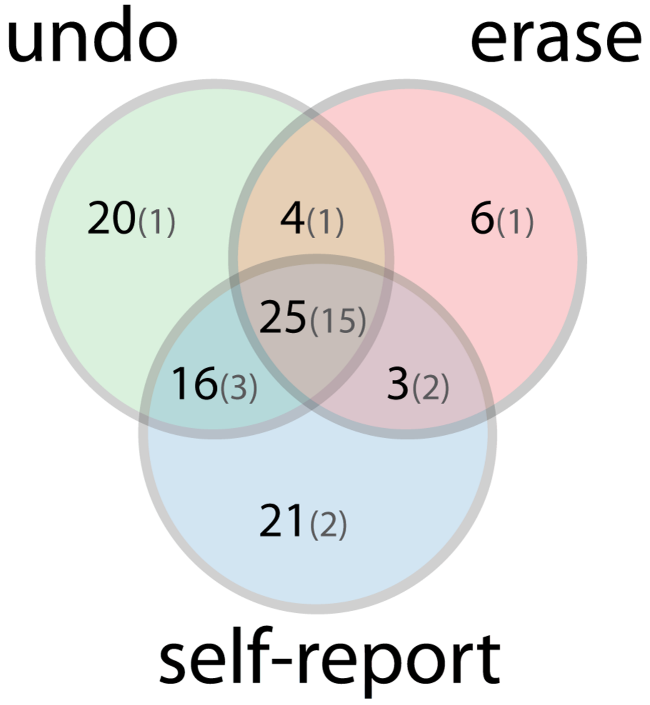

Figure 5: A Venn diagram showing the number of usability problems detected by each of the three methods. Problems in the middle were detected by all three methods, while those on the outsides were detected by only one method. Counts of severe problems (problems whose severity rating was at least 5) are shown in parentheses. Note that undo and erase combined to detect over 90% of the severe problems found by any of the three methods. Undo and self-report detected similar numbers of mild and medium-severity problems.

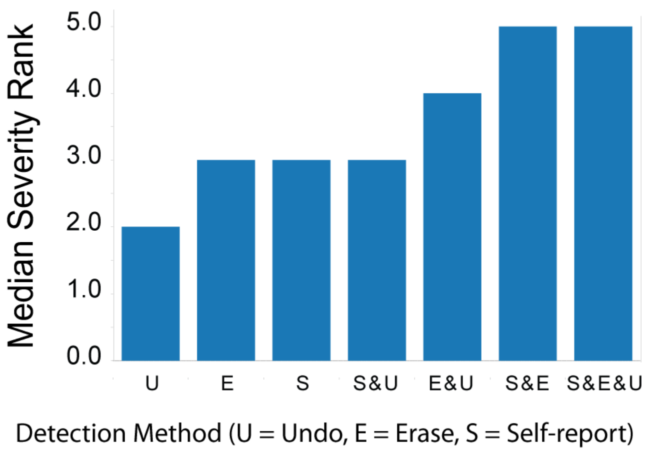

Figure 6: Median severity ratings for the problems detected by each method or combination of methods. Problems detected by only one method have lower median severity than problems detected by more than one method. Problems detected by all three methods have the highest median severity, nearly twice that of the median of problems detected by any single method alone.

DISCUSSION

This section reflects on our results, and discusses possible threats to internal and external validity.

Interpreting the Results of this Experiment

We were initially surprised to find that undo and erase detected such a high percentage of the problems (over 90% of the severe problems). Many problems, for example, do not seem likely to produce *either* an undo or an erase.

Upon reflection, there are three possible reasons why we believe that the problem detection rates for undo and erase were so high. First, sometimes undo or erase operations happened in circumstances we would not have expected. Consider the following real problem instance from our experiment: A user attempted to move a piece of geometry, and nothing happened (because, unbeknownst to the user, the geometry was anchored to its position). While it would seem that there was no reason to undo (since there was no actual change to the geometry), the user still executed an undo just to make sure that he hadn't changed anything.

Second, recall that we recorded problem instances even when the discovery of the problem was incidental to the method. (This accounted for nearly 20% of our problem instances.) Undo and erase operations tend to cluster at times when people are experiencing difficulties. Within the 20 second window of each screen capture episode, we often detected participants having difficulties unrelated to the undo or erase event that triggered the episode.

Finally, consider that our analysis of problem detection does not take into account the *rate* of detection. We counted a problem as being "detected" by a method even if only one participant experienced one instance of the problem in a single episode. The problem might have been detected by another method in 10 separate instances, but in our analysis these instances were redundant information. It is worth taking into account the rate of detection in future analyses.

Time Allocation for this Experiment

Of the 90 minutes in our experiment, participants spent only 15 minutes working on the task that we gave them. To some extent, the short task time reflects the goals of this study, which were to *compare* three evaluation methods rather than to use them. However, it is also useful to note that 20 of the 90 minutes were spent training participants in the use of the user-reported critical incident method. While it may be possible to reduce the amount of training, this is a fundamental difference between self-reporting and event-based methods. Event-based methods can be employed without any up-front training (and even without the user's prior awareness that they are being monitored).

Addressing Internal Validity Concerns

We were careful to try to minimize several possible threats to the internal validity of this experiment. Since we varied the method in a within-subjects manner, we fully counterbalanced the order of the methods in the retrospectives to avoid learning or fatigue effects. One other internal validity threat lies in the process of merging of usability problem instances. If we were inconsistent in how we merged problems, problems might end up at substantially different levels of granularity. (Consider the difference between, "Users have trouble selecting objects" and, "Users have trouble understanding how to select objects when using the scale tool.") The former would likely attract a higher severity rating, and would also be more likely to be detected by all three of our methods. Aware of this potential threat, we tried to write problem descriptions that were much more like the latter than the former, and took a conservative approach to merging. Unfortunately, there is no simple test for success; merge rates naturally vary with the frequency of a problem's occurrence, as well as its level of granularity. In the future, we plan to experiment with hierarchical descriptions [2], which represent difficulties at multiple granularities.

Generalizing from this Experiment

It is tempting to apply the conclusions of this experiment to other tasks, settings, types of applications, and evaluators. This section discusses the potential challenges involved.

Generalizing to other tasks

The conclusions of this experiment may depend to some extent on the modeling tasks that we chose. SketchUp is a large and complex application; it is not possible to comprehensively evaluate its usability by choosing a few representative tasks. That said, we tried to choose tasks that we expect are representative of new user goals (building and furnishing a room, and constructing a simple model). We hope that our results would generalize at least to these broader classes of modeling tasks. As mentioned previously, it is important to keep task goals precise to reduce false positives due to changes of goal.

Generalizing to other application types

It is somewhat more risky to generalize to other applications besides SketchUp. Certainly, the application would need to have undo and erase functions that are used frequently, and there must be some mechanism to instrument the application to capture them. This would include creation-oriented applications like word processors, page layout tools, graphics editing applications, content creation software for virtual worlds, etc. The value of our approach would vary across these application types.

Generalizing to other settings

We do not recommend attempting to generalize our conclusions to other experimental settings. The quality of our data may be highly dependent on the laboratory environment in which it was collected. In a remote evaluation setting, it would be difficult to facilitate the kind of paired discussions that we believe were critical to revealing some usability problems. It would be even more of a stretch to consider natural settings in which the participants are not aware of the experimental apparatus.

Generalizing to other evaluators

The evaluator effect [13] for traditional usability testing is also a concern for our method. Since we employed only a single evaluator to extract usability problems, it is possible that some of our findings would not generalize to other evaluators. We hope that this effect is smaller for our method than for traditional testing (since in our method the episodes are automatically selected), but only future studies can determine the extent of the effect.

CONCLUSIONS AND FUTURE WORK

The results from this experiment have convinced us that undo and erase events represent a promising way to detect problems in creation-oriented applications like Google SketchUp. We have been particularly encouraged by the problem detection rates, especially for severe problems.

The size of the participant pool is not the only consideration when designing a usability testing strategy (*e.g.*, consider the goal of frequent iteration of testing and software modification [18], or the importance of task coverage [17]). But for situations when study scale *is* a limitation, we hope that our approach is of interest. In this project, we iteratively improved our own methods by comparing them against a known method for cost effective usability testing, the user-reported critical incident technique. Future studies will compare our approach to traditional usability testing [21], and test with other applications besides SketchUp.

ACKNOWLEDGMENTS

Thanks to Bryce Stout, Charles Akers, and Xin Fu for their help with the study and the paper.

REFERENCES

1. Capra, M. Contemporaneous Versus Retrospective User-Reported Critical Incidents in Usability Evaluation. In *Proc. Human Factors 2002*. HFES (2002), 1973-1977.
2. Cockton, G. and Lavery, D. A Framework for Usability Problem Extraction. In *Proc. Interact 1999*. IOS Press (1999), 344-352.
3. Cronbach, L. J. Coefficient alpha and the internal structure of tests. *Psychometrika 16, 3* (1951), 297-334.
4. del Galdo, E.M., Williges, B.H., and Wixon, D.R. An Evaluation of Critical Incidents for Software Documentation Design. In *Proc. Human Factors 1986*. HFES (1986), 19-23.
5. Flanagan, J.C. The Critical Incident Technique. *Psychological Bulletin 51, 4* (1954), 327-358.
6. Google SketchUp, http://sketchup.google.com
7. Google SketchUp New User Tutorial Videos 1-3, http://www.youtube.com/user/SketchUpVideo.
8. Guan, Z., Lee, S., Cuddihy, E., and Ramey, J. The validity of the stimulated retrospective think-aloud method as measured by eye tracking. In *Proc. CHI 2006*. ACM Press (2006), 1253-1262.
9. Hartson, H.R., Castillo, J. C., Kelso, J., and Neale, W. C. Remote evaluation: the network as an extension of the usability laboratory. In *Proc. CHI 1996*. ACM Press (1996), 228-235.
10. Hartson, R. and Castillo, J.C. Remote evaluation for post-deployment usability improvement. In *Proc. AVI 98*. ACM Press (1998), 25-27.
11. Hilbert, D. and Redmiles, D. An approach to large-scale collection of application usage data over the Internet. In *Proc. Software Engineering 1998*. ACM Press (1998), 136-145.
12. Howarth, J., Andre, T. S., and Hartson, R. A Structured Process for Transforming Usability Data into Usability Information. *Jour. of Usability Studies*, 3,1 (2007), 7-23.
13. Jacobsen, N.E., Hertzum, M., and John, B. E. The evaluator effect in usability tests. In *Proc. CHI 1998*. ACM Press (1998), 255-256.
14. Kirsh, D. and Maglio, P. On distinguishing epistemic from pragmatic action. *Cog. Sci. 18* (1994), 513-549.
15. Koenemann-Belliveau, J., Carroll, J., Rosson, M. B., and Singley, M.K. Comparative usability evaluation: critical incidents and critical threads. In *Proc. CHI 1994*. ACM Press (1994), 245-251.
16. Law, E.L. and Hvannberg, E.T. Analysis of Combinatorial User Effect in International Usability Tests, In *Proc. CHI 2004*. ACM Press (2004), 9-16.
17. Lindgaard, G. and Chattratichart, J. Usability testing: what have we overlooked? In *Proc. CHI 2007*. ACM Press (2007), 1415-1424.
18. Medlock, M.C., Wixon, D., Terrano, M., Romero, R., and Fulton, B. Using the RITE method to improve products: a definition and a case study. In *Proc. Usability Professionals Association* (2002).
19. Nielsen, J. and Landauer, T. A mathematical model of the finding of usability problems. In *Proc. CHI/INTERACT 93*. ACM Press (1993), 206-213.
20. Nisbett, R. and Wilson, T. Telling more than we can know: Verbal reports on mental processes. *Psychological Review, 84*, 231-259, 1977.
21. Rubin, J. and Hudson, T. Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests. John Wiley & Sons, Inc. (1994).
22. Spool, J. and Schroeder, W. Testing web sites: five users is nowhere near enough. In *Extended Abstracts of Proc. CHI 2001*. ACM Press (2001), 285-286.
23. Swallow, J., Hameluck, D., and Carey, T. User Interface Instrumentation for Usability Analysis: A Case Study. *In Proc. Cascon 97*, 1997.
24. Winograd, T. and Flores, F. (Eds.) Understanding computers and cognition. Ablex Publish. Corp (1986).