Effects of Intelligent Notification Scheduling
Shamsi T. Iqbal and Brian P. Bailey
Department of Computer Science
University of Illinois
Urbana, IL 61801 USA
{siqbal, bpbailey}@cs.uiuc.edu

ABSTRACT
This work reports results from two studies investigating intelligent notification scheduling. The first study tested the performance of composite statistical models for detecting and differentiating three granularities (types) of breakpoints within novel task sequences. Results showed that the models detect breakpoints reasonably well, but do not perform as expected for differentiating their type. Our second study investigated how scheduling notifications at different types of breakpoints affects users and their tasks. Results showed that scheduling notifications to occur at breakpoints reduces frustration and reaction time relative to delivering them immediately. We also found that the content of a notification determines the type of breakpoint at which it should be scheduled. The overall concept of scheduling notifications at breakpoints matched well with how users preferred notifications to be managed. This indicates that users would be willing to adopt the use of notification scheduling systems in practice.

Author Keywords
Breakpoint, Interruption, Notification, Statistical models

ACM Classification Keywords

INTRODUCTION
Controlled laboratory studies have shown that scheduling notifications to be delivered at breakpoints during users’ tasks reduces the ensuing cost of interruption [1, 3, 19]. A central and important goal is to understand how to allow similar results to be realized for users in practice. As a first step toward realizing this goal, researchers have demonstrated the feasibility of building statistical models that detect breakpoints in real life interactive tasks [20].

At least two challenges now remain for achieving the goal. One challenge is to understand how well composite statistical models can detect and differentiate breakpoints for novel task sequences. Composite models are built by aggregating training data from multiple users [20]. Novel task sequences are those that are generated by different users performing tasks from the same domain, e.g., multiple programmers doing programming tasks. Because composite models require less effort to develop than per-user models and could be deployed widely, it is important to understand how well they perform for different users.

Assuming such models perform well, a second challenge is to carefully understand how scheduling notifications at breakpoints affects users and their tasks in practice. This is crucial because the impact of automatically scheduling notifications has not yet been studied in context of authentic tasks. It is thus not known whether deploying such systems would indeed have utility for the user.

In this paper, we report results from two user studies that directly address both of these challenges. Two complex task domains - diagram editing and programming – were used in the studies. Our first study tested the performance of composite models for detecting and differentiating three granularities (types) of salient breakpoints within novel task sequences. Results showed that the models detect breakpoints reasonably well (55.5% on average), but do not perform as expected for differentiating type. The main implication is that composite models are most useful for detecting breakpoints (without differentiating their type) for notification scheduling systems.

Our second user study investigated how scheduling notifications at different types of breakpoints affects users and their tasks relative to delivering them immediately. Breakpoints were being detected in real time using the models from the first study. Users identified the type of breakpoints retrospectively, as a means for effectively compensating for the outcomes of our first study.

Results showed that scheduling notifications to occur at breakpoints reduced frustration and reaction time relative to immediate. This was balanced against a relatively small deferral time. We also discovered that the relevance of notification content to the ongoing task determines the type of breakpoint at which it should be scheduled. This stresses the need for being able to differentiate breakpoint type in practice. The overall concept of scheduling notifications at breakpoints matched well with how users preferred notifications to be managed. This is important.
as it strongly indicates that users would be willing to adopt notification scheduling systems in practice.

RELATED WORK
We describe breakpoints, their use for notification management, and how they can be detected. We discuss approaches for predicting other interruptible moments and how the predictions can be used to schedule notifications.

Breakpoints, Their Use, and How to Detect Them
A breakpoint represents the moment of transition between two observable, meaningful units of task execution [23]. These moments reflect internal transitions in perception or action [24]. Prior work has identified three types (granularities) of perceptually meaningful breakpoints during interactive tasks – Coarse, Medium, and Fine [20]. Coarse breakpoints exist between the largest meaningful units of execution while Fine exist between the smallest. Empirical studies have shown that delivering notifications at breakpoints reduces interruption cost (e.g., resumption lag [19]). Further, results show that coarser breakpoints typically correspond to lower cost [1, 3, 19].

A major challenge is being able to detect and differentiate breakpoints in user tasks. A promising approach is using statistical models that map interaction to different types of breakpoints. Prior work has demonstrated the feasibility of building such models that detect and differentiate breakpoints with 69-87% accuracy [20]. It is important to note these accuracies were achieved by testing the models on the original training data (10-fold cross validation).

Our work investigates how well composite models are able to detect and differentiate breakpoints in novel task sequences. This means that the performance of the models is being evaluated on data generated from different users performing complex tasks from the same domain. This will provide understanding of how well composite models would perform if used in a notification scheduling system to implement defer-to-breakpoint policies.

Detecting Other Interruptible Moments
Other work has developed statistical models for detecting interruptible moments, but not breakpoints explicitly. These models typically leverage cues related to desktop activity, visual and acoustical analysis of the physical task environment, and scheduled user activities [13-15, 17].

For example, Horvitz and Apacible use these cues to infer a probability distribution over users’ attentional state, from which a cost of interruption is computed [13]. Fogarty et al. built statistical models that map interaction events (typing, scrolling, browsing, etc.) to one of three classes of task engagement [8], where ground truth was determined using reaction time to a secondary task. While models in this corpus of work have been shown to be effective for predicting interruptibility, they have either not been utilized for scheduling notifications or the effects of scheduling notifications have not been studied.

Beyond this body of work, our research further investigates statistical models that detect and differentiate breakpoints in complex tasks. Using these types of statistical models, we investigate the effects of scheduling notifications at breakpoints on users and their tasks.

Scheduling Notifications at Interruptible Moments
Only a few systems have been developed that schedule, or are capable of scheduling, notifications at interruptible moments. One such system is Lookout [12]. This system predicts a user’s dwell time on a communication message based on an analysis of its content. This prediction is then used to schedule delivery of automated assistance.

The Notification Platform is a system that modulates flow of messages (or notifications) from multiple sources to devices by performing ongoing decision analysis [15]. In this system, messages are delivered using the device and modality that is most beneficial for the user. These values are computed using a decision theoretic framework called Coordinate [16]. A later and related system, BESTCOM, considers social and task context, available channels, and preferences about communications to select the best timing and modality for interpersonal communications [16]. Other similar systems have also been developed [4, 9]. However, the impact of automated scheduling of notifications using these systems has not been studied.

Relative to this corpus of work, a primary contribution of our research is that we studied the impact of scheduling notifications on users and their tasks. We used one particular technique for scheduling, deferring notifications until breakpoints, but our results may help improve the design of similar scheduling techniques in other systems.

OASIS: A NOTIFICATION SCHEDULING SYSTEM
OASIS is a system that allows notifications to be deferred until breakpoints are reached during interactive tasks. This approach allows notifications to be presented in a timely manner, but at moments that have been shown to correspond with reduced interruption cost [1, 3, 11, 19].

As illustrated in Figure 1, OASIS consists of two primary components; the breakpoint detector and the scheduler. When an application wants to render a notification, it sends a request to the scheduler. The request consists of a flag indicating which policy to use and a maximum time it can wait. As a first step, four policies are supported; defer until next fine, medium, or coarse breakpoint; and defer until next breakpoint of any type. The scheduler queues the request until the specified type of breakpoint occurs or until the timeframe expires. In either case, the request is granted and the application can render its notification. For example, if a user is constructing a diagram and an e-mail notification is generated, our system would allow it to be deferred until the user finishes their current manipulation.

The breakpoint detector monitors the user event stream. Events are pooled for a few seconds and are then fed into
When an application wants to render a notification, it sends a request to our system. The system monitors the event stream and queues the request until an appropriate breakpoint is detected or the given timeframe expires. Requests can be held until the next Fine (F), Medium (M), or Coarse (C) breakpoint, or until the next breakpoint of any type occurs (Any).

![Diagram of OASIS]

Figure 1. A schematic of our notification scheduling system. When an application wants to render a notification, it sends a request to our system. The system monitors the event stream and queues the request until an appropriate breakpoint is detected or the given timeframe expires. Requests can be held until the next Fine (F), Medium (M), or Coarse (C) breakpoint, or until the next breakpoint of any type occurs (Any).

The first question is critical because users’ task behavior is widely known to be highly variable. It is thus not clear how well models for detecting breakpoints trained with one data set would perform on a different data set, especially if the data was generated by different users.

The second question is important because understanding effects of scheduling notifications at breakpoints has only been investigated to date using controlled and relatively simple tasks where breakpoints were selected a priori (e.g., [1, 2, 5, 18, 19, 22]). It is thus unknown whether similar results are possible when scheduling notifications in context of authentic, complex tasks and when the breakpoints are being identified in real-time.

**Task Domains**

Two domains were selected for this work; programming and diagram editing. These were selected because many users perform tasks in these domains, the tasks performed are typically complex, and the tasks are often intertwined with other activities such as Web browsing or managing communications. Microsoft’s Visual Studio and Visio were chosen as the specific applications for programming and diagram editing, respectively. For each application, we developed custom plugins that expose a large number of application events that can be monitored by our system.

**Training Initial Models**

Following procedures outlined in [20], we trained a set of statistical models for detecting breakpoints when working within Visual Studio and Visio (Fine and Medium), and when switching between higher-level activities (Coarse).

Six users (3 per domain) were recruited and our data collection software was installed on their machines. Users were asked to run the software the next time they would be focused on performing any task within the assigned domain for at least 90 minutes. They were also asked to perform the activity as usual, i.e., it was perfectly fine to check mail, play music, or read news intermittently. Our software collected user’s screen interaction, application and system-level events, and keyboard and mouse events.

The users performed a diverse set of complex tasks. For programming, one user was working on a Web-based graphics application using ASP.net. The second user was programming a notification display using Visual C#. The third user was writing C++ code to manipulate mouse and keyboard events for a distributed application. For diagram editing, one user was creating an information architecture for a research website. The second user was creating a project proposal outline for the local environment council. The third user was creating a system diagram for a poster.

Twelve independent observers were then recruited. Using a software tool, each observer reviewed two interaction videos and identified locations of perceived breakpoints and their type (fine, medium, coarse). The aggregated set of identified breakpoints was filtered to include only those breakpoints for which there was a minimum threshold of agreement. Details of this process can be found in [20].

Training examples were created from the events occurring around each breakpoint. Examples were also created for a random sample of moments not identified as a breakpoint (NAB). Extending prior work [20], the training examples included a much wider range of events. This was achieved by deriving feature values (e.g., changes in cursor direction) from other events (e.g., mouse position events...
Overall, results show that our models are able to detect breakpoint when it is not, was very low (<0.5%). Although the models do miss some breakpoints, the most egregious type of error, wrongly predicting a moment to be a breakpoint, was a breakpoint within a given interval. Examples were aggregated for each task domain, as analogous work has shown this to result in good performance [13]. The composite models were learned using the Weka machine learning toolkit.

The general model for detecting Coarse breakpoints had 11 features (predictive events), while application specific models for detecting Medium and Fine within Visual Studio and Visio had 13 and 17, respectively. For Coarse, features included switch to mail client, switch to IM client, and window minimized. For Medium and Fine in Visual Studio, features included document closing, build done, and switch to search. For Visio, features included shape added, application deactivated, document saved, and begin zooming.

### Training Results

Tables 1 and 2 show the accuracy of the models, with the diagonals showing correct predictions. Overall accuracies are reasonably high (> 87%). This is due in part to the large number of NABs, and the models having predicted most of them correctly (99% for the general model, 98% for programming, and 97% for diagram editing).

For the types of breakpoints, Coarse were predicted with 71% accuracy (60/85). Medium were predicted with 84% accuracy for programming and 56% accuracy for diagram editing. Fine were predicted with 96% and 58% accuracy for programming and diagram editing, respectively. Although the models do miss some breakpoints, the most egregious type of error, wrongly predicting a moment to be a breakpoint when it is not, was very low (<0.5%).

Overall, results show that our models are able to detect and differentiate breakpoints in the training data, with high accuracy, and the results are consistent with prior work [20]. This gave us high confidence that these models were robust enough for testing on novel task sequences.

### STUDY 1: EVALUATE BREAKPOINT DETECTION

The purpose of the first study was to evaluate how well breakpoints could be detected and differentiated within

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>60</td>
</tr>
<tr>
<td>Not Coarse</td>
<td>19</td>
</tr>
</tbody>
</table>

**Table 1.** Predicted vs. Actual for the general model used to detect Coarse breakpoints. Overall accuracy was 97.97%.

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Med</td>
<td>64, 40</td>
</tr>
<tr>
<td>Fine</td>
<td>0, 0</td>
</tr>
<tr>
<td>NAB</td>
<td>16, 5</td>
</tr>
</tbody>
</table>

**Table 2.** Predicted vs. Actual for the application models used to detect Medium and Fine for (programming, diagram editing). Overall accuracies were 96.7% and 87.6%, respectively.

For programming, one user was developing a graphical interface for a mobile device using Visual C#. Another user was programming a graphics rendering tool using Visual C++. The third user was writing code to process images files in Visual C#. For diagram editing, one user was creating an outline for her doctoral thesis. The second user was diagramming the logic flow of an interactive game. The third was creating a process diagram for a research paper.

### Procedure

The procedure consisted of two phases. In the first phase, users worked on their selected tasks with our software running. Our system monitored the event stream and used the originally trained models to detect whether and what type of breakpoint had occurred. Event data was pooled in 3 second bins, as this was found in the training phase to give the best accuracy. Each bin of event data and the related prediction (Fine, Medium, Coarse, or NAB) were logged to a file. Users’ screen interaction was recorded, and could be synchronized with the event data.

For the second phase, users used a software tool to review their own interaction videos and identify locations of the breakpoints and their type. We asked the users themselves to identify the breakpoints, rather than utilize independent observers, because a system like ours will ultimately need to be evaluated based on how well its predictions match a user’s own understanding of their tasks. Note that using observers in the training phase was important because it allows the most perceptually salient breakpoints to be detected and used for training, resulting in robust models.

### Measurements and Analysis

We compared the breakpoints detected by the composite models to the breakpoints identified by the users. A system-identified breakpoint was considered a match with a user-identified breakpoint if they were within 10s and of the same type. After testing several values, a 10s window seemed to best compensate for the difference between when a breakpoint occurred and when a user annotated it.
Users were provided with a description of convolution filters to images, and to implement at least three filters. Tasks required users to develop solutions to challenging, ill-structured problems during the study. Only high-level descriptions were provided, and it was up to the users to work out a desired solution. For programming, the task was to create a user interface for applying convolution filters to images, and to implement at least three filters. Users were provided with a description of convolution

**Discussion**

Results from this study highlight the significant challenge of using composite models to detect and differentiate breakpoints within novel task sequences. Several methods could be pursued to increase the accuracy of such models. For example, models could be trained using a much larger data set, they could be trained on a per user basis, or a combined approach could be followed. Including features representing additional task context must also be pursued.

Yet even if this problem can be mostly solved, and we believe that it will given the active ongoing research in this direction [6, 15, 16], a critical question still remains. Would scheduling notifications at breakpoints (assuming correct identification) have a positive impact for users?

To provide a first answer to this question, we wanted to build upon the fact that the models were able to identify the location of breakpoints with reasonable accuracy. We thus retrained our models to identify moments as either breakpoints or NABs, i.e., we collapsed breakpoints into one type. Applying these new models to the same data set resulted in 59% and 52% of user-identified breakpoints being correctly identified for programming and diagram editing, respectively. We judged this to be sufficient for moving forward with our second study.

As part of the study, we also wanted to test the effects of scheduling notifications at each type of breakpoint. We decided to use our system (with the new models) to detect the locations of breakpoints and to ask the users to identify type. This would effectively compensate for the differentiation performance of the models, and allow breakpoint type to be included in the analysis. We felt that this was important because it would show whether the ability to differentiate breakpoint type for scheduling notifications would have any benefit for users when performing complex tasks.

**STUDY 2: EVALUATE EFFECTS OF SCHEDULING**

The purpose of the second study was to evaluate how scheduling notifications at breakpoints impacts users and their tasks. Also, to investigate the interaction between content and scheduling policies, notifications were designed to be either relevant to the ongoing task or of general interest to the user (but not relevant to the task).

<table>
<thead>
<tr>
<th>User-identified</th>
<th>System-identified</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coarse</td>
</tr>
<tr>
<td>Coarse</td>
<td>33</td>
</tr>
<tr>
<td>Medium</td>
<td>22</td>
</tr>
<tr>
<td>Fine</td>
<td>25</td>
</tr>
<tr>
<td>NAB</td>
<td>68</td>
</tr>
</tbody>
</table>

Table 4. System- vs. User-identified breakpoints for tasks in diagram editing. Overall accuracy was 93.8%.

Results

Tables 3 and 4 show the distribution of system- vs. user-identified breakpoints, with the diagonals showing the matches. Data was aggregated across users in each domain. The overall accuracies were high (> 90%), but again, this is due in part to the large number of NABs, and the ability of the models to correctly predict most of them.

For breakpoints, Coarse was predicted with 41.5% and 41.3% accuracies during programming and diagramming, respectively. Medium was predicted with 20.4% accuracy for programming and 10% accuracy for diagram editing. Fine was predicted with 15% accuracy for programming and only 1.7% accuracy for diagram editing. Admittedly, these results were much lower than had been expected.

Closer inspection reveals that the majority of mismatches was due to users identifying breakpoints as Medium or Fine, while the system identified those same breakpoints as Coarse. On the one hand, this is in fact a very positive result because it shows that users and the system were agreeing on the location of the breakpoints, but were disagreeing on the type of those breakpoints.

Part of the reason behind the low accuracy in identifying the type of breakpoint was the inability of the models to understand users’ task context. For example, one user switched repeatedly between Gmail and Visio to retrieve documents related to her task. The system identified these switches as Coarse breakpoints, while the user identified them as Medium or Fine considering the relevance to the ongoing task. This illustrates how task context influences perceptions of breakpoint type as well as the necessity and challenge of integrating such context into the models.

The most egregious type of error, detecting a breakpoint when none was present, was still very low; 2.8% for programming and 2.3% for diagramming.

<table>
<thead>
<tr>
<th>User-identified</th>
<th>System-identified</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Coarse</td>
</tr>
<tr>
<td>Coarse</td>
<td>44</td>
</tr>
<tr>
<td>Medium</td>
<td>40</td>
</tr>
<tr>
<td>Fine</td>
<td>35</td>
</tr>
<tr>
<td>NAB</td>
<td>69</td>
</tr>
</tbody>
</table>

Table 3. System- vs. User-identified breakpoints for programming. Overall accuracy was 90.5%.

Users and Tasks

16 users (8 per domain) were recruited for the study. Users reported having moderate to expert skills in the respective domain. Users received $50 for participating.

Tasks required users to develop solutions to challenging, ill-structured problems during the study. Only high-level descriptions were provided, and it was up to the users to work out a desired solution. For programming, the task was to create a user interface for applying convolution filters to images, and to implement at least three filters. Users were provided with a description of convolution
filters, pointers to Web-based resources, and a skeleton C# project that they could build upon, if desired. Users were asked to make their code as efficient and readable as possible. MS Visual Studio was used for the task.

For diagram editing, the task was to design a floor plan for a model workspace in a Computer Science building. The space had to accommodate 6 students, and needed to include cubicles for daily work, a joint lab for conducting experiments, and a service room for relaxing and eating. Users were asked to create as many design alternatives as possible. The task was performed using Visio.

For both tasks, users were free to browse for examples, references, or any other desired information online. Users were informed that they needed to spend 2 hours on the task and should work at their own pace. Given the length of the task, users were also informed that they were free to perform other personal tasks such as check mail or read their favorite online news site. The goal was to have users work in a manner similar to how they would in practice.

To facilitate motivation, an additional $50 was offered to the user who created the highest quality solution in each task domain, as judged by independent experts.

OASIS was installed on the experimental machine and monitored the user’s activity to detect breakpoints. It also managed notification requests from a custom application.

**Notifications**

As users performed tasks, they occasionally received notifications. Two types of notifications were used:

- **Relevant.** These notifications provided examples (e.g., source code or floor plans), useful tips, or additional criteria that would be used for judging solution quality.

- **General Interest.** These notifications presented recent news grabbed from Google News or announcements from our department’s or institution’s homepage.

Two policies were used for delivering notifications:

- **Defer to breakpoint.** A request would be sent to OASIS, which would schedule the notification to appear at the next breakpoint detected in the user’s task sequence.

- **Immediate.** The notification was delivered immediately.

16 notifications were generated randomly within intervals spanning the 2 hour task period by a custom application. 16 was based on prior work showing that computer users receive on average about eight notifications per hour [21].

4 notifications appeared as soon as they were generated (Immediate). The other 12 were to be scheduled by OASIS to appear at breakpoints. However, depending on our system’s detection of breakpoints, it was possible for users to receive fewer. We chose to have more scheduled (12) than immediate (4) notifications to try to ensure that each type of breakpoint would be used (how types were identified is discussed below). Half of the notifications were Relevant and the other half were of General Interest. These were balanced between the two scheduling policies.

Notifications were rendered as a non-modal window in the lower right of the screen and contained a short snippet of the message content (figure 2). The window persisted for 7s. Users could select the text snippet to read the full message. The overall design was meant to simulate a technique commonly used today (e.g., by MS Windows).

**Experimental Design**

We used a 2 Activity (programming, diagram editing) X 2 Policy (breakpoint, immediate) X 2 Content (relevant, general interest) mixed design. Policy and Content were within subjects while Activity was between subjects.

**Procedure**

Upon arrival at the lab, we went through an informed consent process with the user. The user was provided with a description of the task and allowed to ask any questions. Users were contacted prior to their scheduled session so that the local machine could be configured with their favorite applications and bookmarks as best as possible.

The user was informed that during the task, notifications containing relevant or potentially useful information would occasionally appear. They were asked to select the notification whenever they noticed it and/or the task allowed. If selected, a dialogue box would immediately open, asking the user to rate his or her frustration with having received the notification at that moment. Once the rating was made, a Web page opened showing the full content of the notification. Users were then free to proceed as desired. The task session lasted for 2 hours.
Afterward, a post experiment interview was conducted. The experimenter launched a tool that showed the user’s interaction video along with the locations of the system-identified breakpoints. The experimenter navigated to each breakpoint and asked the user to agree or disagree with whether that moment was a breakpoint. If agreed, the user identified its type (coarse, medium, or fine) based on given descriptions. If disagreed, the user scrubbed the video to identify the closest point where they would have preferred to receive the notification and explain why.

**Measurements**

The following measurements were taken:

- **Frustration.** The rating was made using a 7-point Likert scale, ranging from very pleasing to very frustrating.
- **Reaction time.** This was measured as the time between when a notification appeared and the user selected it.
- **Resumption time.** This was the time from when the user closed the notification content page to when focus on suspended activity was resumed. This time would also include diversions into other activities, if any.

These metrics have been used to measure the effects of interruption in many previous studies (e.g., [3, 18, 21]). In addition, we solicited the user’s feedback on how s/he felt about having notifications deferred until breakpoints. We also analyzed their screen interaction videos to compare how users proceeded after responding to notifications delivered under different scheduling policies.

**RESULTS**

Out of a maximum possible 256 notifications, 64 were to be delivered under Immediate and 192 were to be delivered under Scheduled. Out of the 64 Immediate, 1 was not delivered due to the user finishing the task before the notification could be generated.

Out of the 192 Scheduled, 170 were delivered. 109 of these were delivered at moments that users agreed were breakpoints. This meant that our system had 64% accuracy in detecting breakpoints, a very positive result. Of these 109, users identified 26 as Coarse, 44 as Medium and 39 as Fine. These user-specified types were used as the values for the Policy factor in our analysis. The 61 cases where there was no agreement were excluded.

The remaining 22 Scheduled could not be delivered due to the system not being able to detect any breakpoint between when the notifications were generated and the end of the task session. These were excluded. Also, in some cases, a notification appeared while the user was still responding to another. This resulted in an additional 29 notifications being excluded to avoid any potential confounding effects. In sum, our rigorous filtering process left us with 143 data points for analysis.

For Scheduled notifications that were delivered, the mean deferral time (the time from when they were generated to when they were delivered) was 88.6s (S.D. 139.3s). This indicates that our scheduling system allows notifications to be delivered in a timely manner. Deferral times are consistent with the breakpoint distances reported in [20].

**User Reactions and Behavioral Responses**

The concept of scheduling notifications at breakpoints was well received by users and matched what they themselves preferred. For example, when scrubbing the video to select preferred moments to receive notifications that had not appeared at a breakpoint, they almost always described a moment that indicated the completion of an action. This is exemplified in many of their explanations:

- “After I have added this room [would have been a good moment]”
- “I wish it had waited until I was done with this [the stairs]”
- “Just before that - where I scrolled down the window …just when I ended this method.”
- “I would have preferred it [the notification] when I had just finished this line.”

We also discovered that there was an interaction effect between scheduling policy and notification content. For example, users expressed wanting notifications relevant to their ongoing activity to be delivered at Medium or Fine breakpoints as opposed to Coarse. Even though this may cause higher localized costs (e.g., in terms of resumption lag [19] or reaction time), users perceive a larger global benefit because the notification is received when its content can be best utilized, and precludes the need for a context switch. When a relevant notification was delivered at Coarse, we often observed users immediately returning to the activity they had just left. Users expressed they disliked these occurrences since they were intending to move away from the ongoing task. Receiving a relevant notification caused them to abandon that task switch.

For General Interest notifications, users stated that they wanted them to be delivered only at Coarse. This was also evident in their task behavior. For example, if a general interest notification appeared at Medium or Fine breakpoints, users cursorily read the content and attempted to return to the suspended task as soon as possible. If it appeared at Coarse, users would often read the content in its entirety, and then proceed with their intended task switch.

The key implication of these results is that notifications deemed relevant to the ongoing activity should be scheduled at Medium or Fine, while notifications of general interest should be scheduled at Coarse. This also indicates that notification scheduling systems must be able to detect all three types of breakpoints in practice.

A related but less commonly observed behavior was users initiating *chains of diversion* [21]. This refers to the
activities that a user performs after having attended to a notification but before resuming their suspended task.

25 chains of diversion were observed, 11 for diagram editing and 14 for programming. The nature of the diversion was a function of both the ongoing task and the notification content. For example, during diagram editing, general interest notifications caused users to enter a chain of diversion most often (9 of 11). During these diversions, users would check mail, weather or movie schedules, or browse online news. For programming, users went on a chain of diversion most often (11 of 14) after having received a relevant notification. Diverted activities were often related to the programming task, e.g., looking up code samples or browsing online forums. Policy did not seem to have an impact on the chain of diversion.

**Frustration**

A 3-way ANOVA showed main effects of Content (F(1, 109)=13.9, p<0.001) and Policy (F(3, 109)=5.4, p<0.002), and an interaction effect between Policy and Activity (F(3, 109)=4.7, p<0.004).

For Content, notifications that were of general interest caused more frustration (μ=4.98, S.D.=1.81) than those that were relevant (μ=3.59, S.D.=1.71). Several users stated that even if a notification may have been initially perceived as disruptive, if they determined the content to be relevant, they were more tolerant towards it. This finding is consistent with results in prior work [5, 10].

Due to the interaction, we examined effects of Policy within each Activity separately. For diagram editing, Policy had a main effect on frustration (F(3, 52)=6.2, p<0.001). Post hoc tests showed that notifications delivered at Coarse (μC=3.6, S.D.=1.99) caused lower frustration than at Fine (μF=5.5, S.D.=1.7). Notifications delivered at Medium (μM=2.6, S.D.=1.6) caused lower frustration than at Fine (p<0.001) and Immediate (μI=4.5, S.D.=1.58; p<0.037). No other differences were found.

For programming, trends were in the expected direction (μC=3.28, μM=4.39, μF=4.33; μI=4.82), but did not reach a level of significance. The lack of effect may be due to the fact that programming induced higher cognitive demands than diagram editing, causing users to experience similar levels of frustration across policies. This is further supported by the fact that 15 of the 16 notifications that users failed to respond to were during programming.

**Reaction Time**

A 3-way ANOVA did not reveal main effects of the factors on reaction time. However, inspection of the graph showed a very salient pattern in how users were reacting to relevant notifications delivered at breakpoints versus immediate (see figure 3). To explore this further, we collapsed breakpoints into a single Breakpoint level and reran the ANOVA for only relevant notifications.

Results showed a main effect of Policy (Breakpoint, Immediate) on reaction time (F(1, 66)=3.78, p<0.056). Users reacted to notifications at Breakpoints (μ=3.07s, S.D.=1.2) faster than at Immediate (μ=4.08s, S.D.=3.13).

A plausible explanation for this effect is that for Immediate, users would need to externalize information (e.g., finish the current line of code or complete alignment of shapes) into the task environment, causing slower reaction time. When delivered at breakpoints, users could switch their attention more readily to the notification, resulting in faster reaction time. Analysis of user behavior in the videos confirmed the veracity of this explanation.

The same pattern was not found for notifications that were of general interest. This may be because users did not anticipate being away from the task for long, therefore were less concerned about externalizing information.

**Resumption Time**

An ANOVA revealed an interaction between Content and Activity for resumption time (F(1,109)=7.75, p<0.006). For Diagram Editing, users resumed their activity faster after responding to notifications that were relevant (μ=4.65s, S.D.=4.4s) compared to those that were of general interest (μ=23.1s, S.D.=41.4s; F(1,54)=5.91, p<0.018). This result can be attributed to users initiating more chains of diversion after receiving general interest notifications, as previously discussed (see User Reaction).

For programming, users resumed their activity faster after responding to notifications that were of general interest (μ=8.8, S.D.=21.1) compared to those that were relevant (μ=16.6, S.D.=21.5). Differences were due to users having more chains of diversions after receiving relevant notifications, though these did not reach significance. Overall, these quantitative results reflect the qualitative observations discussed under User Reactions.
DISCUSSION

A central goal of this study was to evaluate the impact of scheduling notifications using various policies on users and their tasks. Our results showed that users experience meaningfully lower frustration when notifications are scheduled to occur at breakpoints than when delivered immediately. Further, scheduling notifications at Coarse and Medium breakpoints results in lower frustration than when scheduled at Fine. One explanation is that users experience temporary reduction in memory load at these moments, or are at a transition in their action sequence. Our results on frustration are consistent with [18].

Users reacted faster to notifications that were scheduled at breakpoints. For notifications delivered immediately, users had to quickly externalize their current thought into the task environment or finish the current action before responding. Similar observations have been reported in other empirical studies [8, 21]. Interestingly, this behavior was not observed when notifications were scheduled at breakpoints. This is likely due to users having just completed their current thought or action at that moment.

Reductions in frustration and reaction time must be balanced against the time that notifications are deferred. Our results show that the average deferral time was less than 90s. We believe this provides an acceptable balance.

Our results did not show that scheduling notifications at breakpoints affects users’ resumption time. Results did show, however, that resumption time depends upon the relevance of a notification to the user’s ongoing activity. This result is due to users following chains of diversion. For example, for diagram editing, notifications of general interest caused users to initiate chains of diversion most often, whereas for programming, it was the relevant notifications that caused these diversions. One implication is that task reminder tools (e.g., those discussed in [21]) can use the relevance of a notification to help detect whether a user is following a chain of diversion or not.

Another important finding is that users appreciate having notifications scheduled at breakpoints. The reason is that this technique closely reflects their own preference for how notifications should be managed. For example, when retrospectively selecting preferred moments for receiving notifications, users identified moments that represented the end of an action corresponding to the completion of a cognitive chunk, e.g., the end of a series of code edits. This strongly indicates that users would accept systems that schedule notifications at breakpoints in practice.

Our results provide further insights into how applications should utilize defer-to-breakpoint policies. For example, applications that generate notifications that are relevant to the user’s ongoing activity should request that they be delivered at Medium or Fine breakpoints. This would allow notifications to be delivered when they have the most utility (i.e., during the activity), but at less disruptive moments. In contrast, for notifications of general interest, applications should request that they be delivered at Coarse breakpoints. These would be the moments when delivering such notifications would be least disruptive.

A second goal of this work was to evaluate how well composite statistical models can detect and differentiate breakpoints within novel task sequences. In Study 1, results showed that our models were able to detect breakpoints with 52% (diagram editing) and 59% (programming) accuracy within novel task sequences. In Study 2, the models were able to detect 64% of the breakpoints accurately. These are very positive results since the breakpoints were the ones that users themselves identified in their own tasks. This is an important outcome because it shows that composite models can be used to detect breakpoints for different users performing the same type of complex task with reasonably high accuracy.

However, the models performed poorly for differentiating breakpoint type. For example, in Study 1, the models differentiated breakpoints with only 2-42% accuracy. Applying the same models to differentiate breakpoints in Study 2 did not yield better results. One implication is that scheduling systems may want to use composite models only for detecting breakpoints, i.e., without differentiation. Our results show that this can be done with reasonably high accuracy, and users can still receive a benefit in terms of reduced frustration and reaction time.

More flexible scheduling policies can be offered if the type of breakpoints could be differentiated. These policies would be useful, e.g., to allow notifications to be more effectively scheduled based on their relevance. Having various policies available would also allow applications to choose an appropriate balance between notification timeliness and levels of frustration for the user. Further work is needed to understand how to improve models for differentiating breakpoints. Related work on modeling interruptibility may provide applicable insights [7, 8, 13].

Two complex task domains were used for this research – diagram editing and programming. Both domains require some form of content generation. Our results are thus most applicable to domains with similar characteristics, e.g., document editing, image manipulation and electronic communication. Future work should study the effects of notification scheduling within other types of task domains such as information-seeking and data manipulation.

CONCLUSION AND FUTURE WORK

Research continues to move closer to being able to intelligently schedule notifications for users. A fundamental challenge is to understand the effects that this scheduling would have on users and their tasks.

Our work has made several contributions addressing this challenge. First, we conducted one of the first user studies
investigating the effects of notification scheduling. One promising method was studied - scheduling notifications at breakpoints. We found that this method yields lower frustration and faster reaction time compared to delivering notifications immediately. This method was also found to be consistent with how users prefer notifications to be managed. New insights were offered for how applications can utilize the relevance of their notifications’ content to more effectively select defer-to-breakpoint policies.

Second, we investigated how well composite statistical models detect breakpoints within novel task sequences. We found that these models can detect breakpoints with reasonably high accuracy (52-64%), but still struggle to differentiate their type. Systems that are only able to detect breakpoints can provide benefits to users (e.g. reduced frustration and reaction time), but being able to differentiate breakpoint type would offer more flexibility. Our work shows that this flexibility would be useful, e.g., to better balance timeliness and levels of frustration.

For future work, we intend to deploy our system and study its impact over a longer period of time. This should reveal new insights into how notification scheduling systems can be designed to be more effective. Second, we would like to develop models for additional commonly used applications so that more users can realize the benefits of intelligent notification scheduling. Finally, we would like to investigate techniques for improving the ability of models to detect and differentiate breakpoints.

ACKNOWLEDGMENTS
This work was supported in part by the National Science Foundation under award IIS 05-34462.

REFERENCES