Jason Carter, Mauro Carlos Pichiliani, Prasun Dewan (2018): Exploring the Impact of Video on Inferred Difficulty Awareness. In: Proceedings of the 16th European Conference on Computer-Supported Cooperative Work: The International Venue on Practice-centred Computing an the Design of Cooperation Technologies - Exploratory Papers, Reports of the European Society for Socially Embedded Technologies (ISSN 2510-2591), DOI: 10.18420/ecscw2018- 9

Exploring the Impact of Video on Inferred Difficulty Awareness

Jason Carter, Mauro Carlos Pichiliani, Prasun Dewan
Cisco Systems-RTP, IBM Research, Univ. North Carolina-Chapel Hill
jasoncartercs@gmail.com, mpichi@br.ibm.com, dewan@cs.unc.edu

Abstract. An important issue in many forms of collaboration technology is how video can help the technology better meet its goals. This paper explores this question for difficulty awareness, which is motivated by academic and industrial collaboration scenarios in which unsolicited help is offered to programmers in difficulty. We performed experiments to determine how well difficulty can be automatically inferred by mining the interaction log and/or videos of programmers. Our observations show that: (a) it is more effective to mine the videos to detect programmer postures rather than facial features; (b) posture-mining benefits from an individual model (training data for a developer is used only for that developer), while in contrast, log-mining benefits from a group model (data of all users are used for each user); (b) posture-mining alone (using an individual model) does not detect difficulties of "calm" programmers, who do not change postures when they are in difficulty; (c) log-mining alone (using a group model) does not detect difficulties of programmers who pause interaction when they are either in difficulty or taking a break; (d) overall, log-mining alone is more effective than posture-mining, alone; (e) both forms of mining have high false negative rates; and (g) multimedia/multimodal detection that mines postures and logs using a group model gives both low false positive and negatives. These results imply that (a) when collaborators can be seen, either directly or through a video, posture changes, though idiosyncratic, are important cues for inferring difficulty; (b) automatically inferred difficulty, using both interaction-logs and postures, when possible and available, is an even more reliable indication of difficulty; (c) video can play an important role in providing unsolicited help in both face-to-face and distributed collaboration; and (d) controlled public environments such as labs and war-rooms should be equipped with cameras that support posture mining.

Introduction

Collaboration technology can directly support some form of collaboration or provide awareness (Gutwin and Greenberg 2002) of collaborators to trigger some unplanned or opportunistic form of collaboration.

Awareness technology, like technology supporting direct collaboration, has supported sharing of collaborator state captured by multiple media. For example, workspace awareness such as scrollbar awareness (Gutwin and Greenberg 2002) allows sharing of the state of collaborators' user-interface while video awareness such as video walls (Abel 1990) and media spaces (Harrison and Minneman 1990, Mantei, Backer et al. 1991) supports sharing of their physical characteristics such as posture, expression, and gaze.

Moreover, awareness technology, like technology supporting direct collaboration, can attempt to give users the feeling of "being there" in one location or go "beyond being there" (Hollan and Stornetta 1992) by supporting forms of sharing not directly provided by face to face interaction. Sharing of collaborator screens (Tee, Greenberg et al. 2006) or videos supports "being there" while sharing of read/write shadows (Junuzovic, Dewan et al. 2007), radar views/multiuser scrollbars (Gutwin and Greenberg 2002), and video silhouettes (Hudson and Smith 1996) supports "beyond being there."

"Beyond being" there awareness technology supports sharing of state computed by the software. Dewan (2016) classifies such computed awareness into derived and inferred awareness. The former presents information that is logically derived from the information about the collaborators and their activities while the latter uses data mining/machine-intelligence techniques to make inferences, which may have false positives and/or false negatives.

One form of inferred awareness supported so far is whether a remote user is facing difficulty. An important reason for awareness is to determine if collaborators are in difficulty, and offer assistance if necessary (Gutwin and Greenberg 2002). Making the computer infer difficulty can reduce the amount of information to be transmitted to a remote helper and/or relieve the helper from manually determining if collaborators are in difficulty, thereby allowing the helper to discover and process difficulties of a larger number of users.

An important issue in many forms of collaboration technology is whether and how video can help the technology better meet its goals (Tang and Minneman 1990, Tang and Minneman 1991, Fish, Kraut et al. 1992, Tang and Isaacs 1992, Isaacs and Tang 1993). This paper explores this question for inferred difficulty awareness.

Driving Problem

Our initial motivation for this work comes from research by Herbsleb, Mockus et al. (2000) and Teasley, Covi et al. (2000) that shows that as distance among programmers increases, there are fewer opportunities to offer help to team members and the productivity of programmers decreases. For example, Teasley et al indicate that if someone was having difficulty with some aspect of code, another developer in the war-room "walking by [and] seeing the activity over their shoulders, would stop to provide help."

We can attempt to make help-giving independent of distance by providing collaboration awareness - (peripheral) knowledge of the remote collaborators and their activities – to an interested observer. Previous work in this area has required the observer to manually infer the status of programmers from their videos and workspace activity (Hegde and Dewan 2008) (Guo 2015). This approach has the disadvantage that team members must allocate precious screen space to awareness information and poll for difficulties. Having difficulty, by definition, is a rare event, if programmers are given problems they have the skills to solve. Thus, polling collaborators state to deduce this status can lead to wasted effort in trying to find "needles in haystacks." Moreover, such polling may, in fact, lead to overzealous help by tutors dedicated to the task of helping (Guo 2015).

One approach to address this problem is to ask programmers to manually indicate their status. Intuition tells us that those who are willing to manually change their status are likely to not set it back, just as people forget to change their busy status in an IM tool, or turn off the "call steward" light in a plane. Prior research has, in fact, shown that manually setting a "not interruptible" flag is unreliable (Milewski and Smith 2000). In the case of the difficulty status, previous work shows that people are often hesitant about explicitly asking for help: Begel and Simon (2008) found that students and new hires are late to use help; LaToza, Venolia et al. (2006) established that programmers often exhaust other forms of help before contacting a teammate; and Herbsleb and Grinter (1999) discovered that employees are less comfortable asking remote rather than co-located teammates for help. The first author's dissertation shows that the ability to press a help button during scheduled help sessions can lead to the opposite effect in which students over-ask for help.

For these reasons, we have iteratively developed an Eclipse extension, called Eclipse Helper, that automatically detects programming difficulty, communicates this information to interested observers, and allows the observers to offer help. As reported in (Carter and Dewan 2015), the system has been used in a field study in which distributed students were offered help with their homework in response to automatically detected difficulties. There were some instances where Eclipse Helper did not predict that students had difficulty, but the next day students came to office hours for help. However, each time the tool predicted a student was in

difficulty and the student was asked if they needed help, the student answered in the affirmative. This result is consistent with previous lab studies of the tool mentioned in the first author's dissertation, which also showed the lack of false positives but the presence of false negatives. This result provides the motivation for pursuing multimodal difficulty detection.

There has been much research recently in "affect detection" (Calvo and D'Mello 2010, Graziotin, Wang et al. 2014), which makes inferences about the mental state of users interacting with computers. The state may be an event-triggered ephemeral emotion or a more persistent mood.

As far as we know, ours is the only research on affect detection (D'Mello and Kory 2012) with the driving problem of providing help to (possibly remote) programmers. It is the dual of the problem in several other efforts of determining the emotions invoked when interacting with a computer-based tutor (D'Mello and Graesser 2009, D'Mello and Kory 2012, Grafsgaard, Wiggins et al. 2014, Vail, Grafsgaard et al. 2014). Our work detects an emotion required to determine the need for intermittent tutoring, while its dual detects emotions required to determine how well continuous tutoring is working. As a result, we cannot mine interaction with the tutor – which Grafsgaard, Wiggins et al. (2014) found was a primary feature in predicting emotions.

Inferred difficulty awareness can be used instead of in addition to physical or computer-supported awareness of collaborator's activities in both industrial and educational environments. Moreover, the potential helpers can be working exclusively on the task of providing help or interrupt their task to provide help based on difficulty cues from inferred and other awareness. In the rest of the paper, we first describe how mining video changes the algorithms and results of difficulty detection, and then discuss how these results can inform the various ways in which inferred difficulty awareness can be used.

Interaction Logs

To determine the additive value of mining video, we need to choose the best baseline algorithm that does not use video. Since our field study reported above, we have changed our algorithm for mining interaction logs and verified, through the lab study described below, that it reduces the false negative rate greatly (by 53%) while slightly increasing the false positive rate (by 3%). This improved algorithm, described below, serves as the baseline for the unimodal log-based algorithm used in our comparisons.

The algorithm divides the raw interaction log into 50-event segments, calculates, for each segment, the ratios of the occurrences of certain categories of commands in that segment to the total number of commands in the segment, and uses these ratios as features over which patterns are identified. Table 1 shows the

five command categories we use in our log-based algorithm and the commands in each category. The focus-in/-out commands indicate leaving/entering a programming environment window, and the show view command indicates opening a new kind of view such as a GIT view. The names of other commands explain their functions.

Table I. Mined Commands and Command Categories.

Command Category	Commands
Focus	Focus In, Focus Out
Delete	Delete Char/String, Cut
Insert	Insert Char/String, Copy, Paste, Replace, Move Caret, Select (Text/All), Line Start
Debug	Run, Add Breakpoint, Remove Breakpoint, Hit Breakpoint, Step Into/Over, Step Return, Compilation Error
Navigation	Open File, Find, Show View

These feature were chosen based on top-down thinking and analyses of logs from our studies, which showed that in the difficulty phases we observed, insertion ratios went down and one or more of the other ratios went up. Our log-based mechanism feeds these features (ratios) to the decision tree algorithm implemented in Weka (Witten and Frank 1999) to get raw predictions. Assuming that a status does not change instantaneously, the mechanism aggregates the raw predictions for adjacent segments to create the final prediction, reporting the dominant status in the last five segments. In addition, it makes no predictions from the first 100 events to ignore the extra compilation and focus events in the startup phase and account for the fact that in this phase programmers' interaction behavior is different as they "warm up."

Our previous work on log-based difficulty detection (Carter and Dewan 2010) shows that a group model (data of all users are used for each user) worked better than individual models (training data for a developer is used only for that developer), perhaps because there is uniformity among logs of different programmers and the group model offered more training data. Therefore, this algorithm built a group model.

Other research in affect detection has also mined interaction with the computer (Grafsgaard, Wiggins et al. 2014, Vail, Grafsgaard et al. 2014). However, as far as we know, ours is the only one that segments the log based on the number of events rather than time – that is, it computes features every C commands rather than every T time units. It is not clear a time-based approach can distinguish

between idle and difficult periods. There has been some research on automatically detecting off-task work (Baker 2007, Cetintas, Si et al. 2009) but it is targeted at math tutoring systems and makes use of tutor-specific features such as the average time taken by previous students to solve each tutor-specific problem. The set of user events we mine is also very different and far bigger than the ones used by other log-based schemes. For example, Grafsgaard, Wiggins et al. (2014) mine events implying successful compilations, start of a coding activity, and end of a coding activity to detect engagement and frustration – a much smaller and different set from our event-set.

Sensors for Physical Characteristics

Previous work in affect detection has shown that certain physical characteristics mined from videos of users correlate with users' affective states. Therefore, it was attractive to explore algorithms for difficulty detection that mine promising physical characteristics. To pursue this direction, we had to determine what kind of cameras we would use to identify such physical characteristics. Our answer was guided by their availability and apparent promise. We used the Microsoft Kinect camera, a commodity item today. We also used the Creative® Interactive Gesture camera, a brand new product the second author won as an Intel award, which is also intended as a commodity product. The Kinect camera is promising because it has been used by earlier work (Grafsgaard, Wiggins et al. 2014, Vail, Grafsgaard et al. 2014) to predict various emotions, and the Intel camera is promising because it directly captures positions of various facial features and research has shown that facial features correlate with various emotions (Machardy, Syharath et al. 2012, Grafsgaard, Wiggins et al. 2014, Vail, Grafsgaard et al. 2014).

Creative® Interactive Gesture Camera

As detailed in the first author's thesis, the Intel camera gave us much worse results than the Kinect camera – so we do not discuss the details of this experiment, focusing only on explaining what went wrong. There were times, shown in Figure 1, where the camera did not capture the facial positions correctly. In the left picture, the participant leans in very close to the camera. In the middle picture, the participant puts his hand on his mouth. In the right picture, the camera recognizes the participant's hand as her face. This problem is also identified by Machardy, Syharath et al. (2012) in research on determining if a video observer is paying attention. They found that glasses and hair falling on the face reduced the accuracy of their results. The problem was less severe in their study as the subjects were passively watching the video. As we see in Figure

1, our active code composition task aggravates the problem of the facial features being obscured.

This discussion provides insight about some of the choices made in previous work regarding the kind of data mined. Two studies in which the subjects were relatively passive – MacHardy et al on video watching and Fritz, Begel et al on code comprehension - mined only physical characteristics to make their inferences. They could not mine interaction logs as these were either not available (in the video study) or had only a small number of navigation events (the code comprehension study). Pure physical features worked well because the users were relatively still.

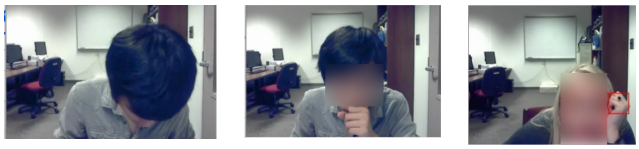

Figure 1. Some uncaptured facial positions

Kinect Camera

D'Mello and Graesser (2009) found that when participants are confused or frustrated, they tend to lean forward, and when they are bored, they tend to lean back. Based on this work, our expectation was that the Microsoft Kinect camera would measure body lean, which in turn, would correlate with difficulty. Figure 2 shows the placement of the Microsoft Kinect camera in our experiment described later. This camera captures the x, y, and z coordinates of 20 human joints at 5 fps. It could not capture all of the joints because, in our experiments, the desk that developers sat at occluded their lower body. This limitation made it difficult for the camera to capture any of the lower body joints. To support users who are sitting at a desk, it has a seated mode, which is designed to track users who sit, and does not attempt to capture joints below the hip. We used the seated mode to capture participants' joints.

Figure 2. Kinect Experimental Setup

There were times where the Kinect camera did not capture the arm and shoulder joints well, but it did capture the head joint correctly the majority of the time. Therefore, like some other studies (Grafsgaard, Wiggins et al. 2014, Vail, Grafsgaard et al. 2014), we used the distance from the head joint to the Kinect camera to measure body lean. This distance was the z coordinate of the head joint in meters.

Once we had this measure, the next question was at what rate it should be sampled. As mentioned earlier, the log-based algorithm calculates ratios every 50 events. However, in this case, this sampling scheme did not give us as good results as sampling the data every minute. Therefore, consistent with other work in posture detection (Grafsgaard, Wiggins et al. 2014, Vail, Grafsgaard et al. 2014), we used the time-based approach.

To convert the z coordinate to body lean (leaning forward, normal, leaning back) we averaged each participants' distance measure per minute and clustered each participants' data individually using the K-means clustering algorithm. Given the distance values and k, the number of clusters to produce, the algorithm partitions values into clusters based on the average Euclidean distance between them. We examined the algorithms' output with two, three, and four clusters. We used the output with three clusters because a) as mentioned above, previous work has shown that leaning back, normal, and leaning forward correlated with affective states, b) the standard deviation of one cluster was large when using two clusters, and c) four clusters offered no improvement over three. We then manually looked at representative images from each of the three clusters to see if the clusters represented the three different leans. Our samples indicated this was largely the case. Figure 3 shows examples of each type of lean from three participants captured by the Kinect camera, which were separated into three different clusters. Thus, our feature now was the cluster in which the z distance fell, which we refer to as the body lean or posture. It had three values: leaning back, normal, and leaning forward.

The next task was to choose an appropriate machine learning algorithm to mine this feature. As in the log-based scheme, we used the decision tree algorithm. Before we could feed the lean information into a machine learning algorithm, we had to determine whether we would use an individual or group model. Our intuition was to use an individual model because all developers may not be in the same posture when they are having difficulty – they may lean in different directions. However, we chose to try both individual and group models to evaluate which model would perform better

When using an individual model, we simply fed each user's per-minute lean to the decision tree algorithm. The process was more complicated in the case of the group model. Assuming that some users would lean forward when in difficulty and some would lean backward, we labeled all postures that were leaning back or

leaning forward as “NOT NORMAL,” and all normal leans as “NORMAL.” We fed this label to the decision tree algorithm.

To the best of our knowledge, the idea of using K-means clustering to classify leans and combining leaning back and forward into one feature in a group model is unique to our work.

Multimodal Algorithm

The basic idea behind combining the two unimodal algorithms is straightforward: use the features from both approaches together. However, there were two incompatibilities between them that had to be resolved. First, as mentioned earlier, in the log case, our previous work found that the group model worked better than the individual model, while, as we will see later, in the Kinect body-lean case, the individual model worked better. Second, as also mentioned earlier, in the two approaches, the features were computed at different moments - every 50 events for interaction logs and every minute for body lean.

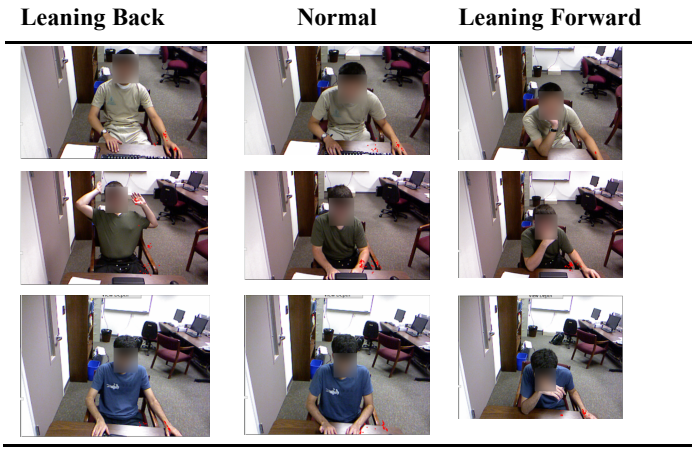

Figure 3. Examples of participants’ leaning back, normal, and leaning forward postures (Kinect camera).

After experimenting with several alternatives, we found that conforming the Kinect body-lean approach to the log-based approach worked the best. Thus, in the combined approach, we built a group model and computed Kinect features every time we had new log features. From a practical point of view, a group-model is preferable to an individual model as it does not require each new user to train the system. Other multimodal affect detection research uses time-based segmentation for all data, implicitly assuming no off-task activity.

Lab Study

To compare the unimodal algorithms with each other and with the multimodal algorithm, we needed to conduct a controlled lab study in which output of our sensors and the interaction logs were recorded. We had to choose our tasks carefully because having difficulty is a rare event. Therefore, we had to ensure developers face difficulty in the small amount of time available for a lab study, and yet did not find the problems impossible. After piloting, we settled on a problem that required participants to use the AWT/SWT toolkit to implement a GUI. Table II shows the tasks that had to be implemented in the study.

Sixteen student programmers participated in the study. They were given at least an hour and a half to complete as many subtasks as possible and were free to use the Internet. We used our previous implementation of the log-based difficulty detection mechanism (used in the field study) to record participants' programming activities and predict whether participants were having difficulty or making progress, and provided a user-interface to correct these predictions and possibly ask for help (Figure 4). The programmers could correct a wrong (a) difficulty prediction by pressing the "I am making progress" button, and (b) progress prediction by pressing the "I am solving the problem on my own" or "I am asking someone for help" buttons. If they needed help, they were instructed to discuss their issue with the first author. Help was given in the form of URLs to API documentation or code examples. By measuring how often the developers corrected their status and explicitly asked for help, we could, measure the accuracy of an inference algorithm with respect to the immediate perceptions of the developers. The videos around the difficulty points were then shown to both the participants and two observers, who confirmed them.

Table II. Mined Commands and Command Categories.

Tasks
Create a program that visually represents a car with a red body and two black tires.
Allow the user to use arrow keys to move around the car in any direction (up, forward, left, and right) by 10 pixel decrements/increments.
Allow the user to make the car a bus by clicking anywhere on the screen. A bus has an extra body that should be colored black. It should be positioned directly on top of the car. When the extra body is on top of the car, it should move with the rest of it.
Allow the user to make the bus a car by pressing the 'r' key. The extra body should be removed
Allow the users to scale up the car/bus 2X, each time they press the 'm' key
Allow the user to scale down the car/bus 2X, each time they press the 's' key

Draw a transparent square (not a rectangle) with yellow borders. The car/bus should be inside the square.
Do not allow the car/bus to go outside of the square (when moving and resizing the vehicle).

The participants' actions were captured using the Intel and Kinect cameras. However, poor lighting forced us to discard camera recordings for six of these participants. Therefore, we report the results of only ten subjects in our comparisons. The length of the study (on average 129 minutes interaction) was much larger than the duration of some other studies (e.g. 32 minutes in (D'Mello and Graesser 2009)) with more participants, and thus compensated to some extent for the discarded data. The number of used participants is typical for studying software developers Overall, the ten participants yielded 45 hours of screen and camera recordings, parts of which were examined to identify features and gather ground truth

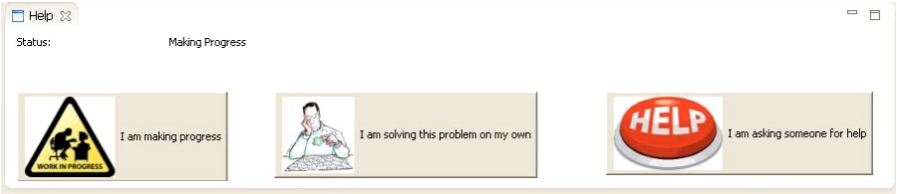

Figure 4: Correcting Predictions

Other schemes for gathering ground truth regarding affective states have relied on post-task participant surveys (Grafsgaard, Wiggins et al. 2014, Vail, Grafsgaard et al. 2014), random interruptions (Fogarty, Hudson et al. 2005), and constant observations (D'Mello and Graesser 2009). The first approach would not support our goal of making instantaneous predictions; the second would not work because, as discussed below, difficulty is a rare event and thus would not be captured by random queries; and the third approach is more labor-intensive than the one we used.

Metrics and Filtering

There were 814 predictions made for these ten users, 55 of which were difficulty predictions. Thus, the number of difficulty segments was much smaller than the number of progress segments, which is consistent with the intuition that having difficulty should be a rare event. This means that an algorithm that always predicted making progress would have very high accuracy (93% in this experiment) but not serve our goal of providing help to needy programmers. For this reason, unlike other multimodal effect detection schemes (D'Mello and Kory 2012), we calculated true/false positive/negative rates, which are presented below using confusion matrices. In the training sets, we used the Weka SMOTE filter (Witten and Frank 1999) to boost the members of the minority class (difficulty).

In our evaluations, we used the standard technique of k-fold cross-validation, which executes k trials of model construction, and splits the logged data so that 90% of the data are used to train the algorithm and 10% of the data are used to test it. We used the value 10 for k, which is typical for such validations.

Unimodal Log-Based Results

Table III shows the results (for the ten participants for which we had good camera data) with the group-based unimodal log-based algorithm. As we see here, the algorithm missed 27% of the time that participants had difficulty. Consistent with the results of our previous log-based algorithm, there were very few (3%) false positives. We extended this analysis by considering all sixteen subjects for whom we had log data. This time, the scheme gave 21% false negatives and 10% false positives. Thus, the larger data set improved the false negative rate at the cost of a higher false positive rate.

Table III: Confusion Matrix for Log Mining Algorithm.

	Predicted difficulty	Predicted Progress
Actual Difficulty	40 (True positives)	15 (False negatives)
Actual Progress	20 (False positives)	739 (True negatives)

Unimodal Lean-Based Results

The confusion matrix in Table V shows the results of the lean-based approach when we used the Kinect camera and built an individual model. As we used a time-based (rather than command-based) segmentation of the interaction sessions, the number of predictions is different from the ones in the log-based approach.

Table V: Confusion Matrix for Kinect (Individual Model)

	Predicted Difficulty	Predicted Progress
Actual Difficulty	333 (True positives)	126 (False negatives)
Actual Progress	108 (False positives)	333 (True negatives)

This scheme has the same false negative rate (27%) as the log-based approach (27%). However its false positive rate is much higher – 25% instead of 3%.

Table VII shows the results of the group model. Both the false positive (45%) and false negative (31%) rates are higher in this case, which is consistent with our intuition about individual lean idiosyncrasies.

Table VI: Confusion Matrix for Kinect (Group Model)

	Predicted Stuck	Predicted Progress
Actual Stuck	315 (True positives)	144 (False negatives)

Actual Progress	198 (False positives)	243 (True negatives)

None of these results is good when compared to the interaction log algorithm One possible reason is given by analyses of some of the Kinect recordings. We found that some people are calm and do not change body lean when in certain kinds of difficulties, while some people are fidgety and change posture when making progress. This observation and the fact that the individual model gave better results than the group model is consistent with previous results showing that lean differences in posture for the same emotion can be accounted by the differences in subjects' extraversion trait score, and extroverts shift posture more often than introverts (Grafsgaard, Wiggins et al. 2014, Vail, Grafsgaard et al. 2014).

Multimodal Results

As mentioned earlier, in the case of the multimodal approach, we built a group model, even though the lean-based approach gave better results when we used individual models. The confusion matrix in Table VII shows our results of the multimodal approach. In comparison to the log-based approach, the false negative rate went down dramatically from 27% to 7%, allowing us to catch many more difficulties, at the cost of the false positive rate going up slightly from 3% to 5%. Intuitively, the reason for (a) a low false positive rate is that the log-based algorithm was able to filter out the noise from users who were fidgety when they were not having difficulty, and (b) a low false negative rate is that when users were not interacting much with the computer, the body lean was able to compensate for the reduced activity. In terms of accuracy, these results provide the modest gains reported in other studies (D'Mello and Kory 2012) – their importance is based on our application and the rarity of the negative emotion.

Table VII: Confusion Matrix for Multimodal Model

	Predicted Difficulty	Predicted Progress
Actual Difficulty	51 (True positives)	4 (False negatives)
Actual Progress	35 (False positives)	724 (True negatives)

Limitations

Equipment: The Kinect camera had difficulty capturing participants' positions correctly. There are many reasons for this problem such as lighting conditions. Thus, our results are a function of the environment we used. Moreover, the Kinect camera is not standard equipment for programmers and students, which limits their use to helping those who program in special inverted classrooms and

labs. These problems are shared with other multimodal research, which, as D'Mello and Kory (D'Mello and Kory 2012) point out, involves "intrusive, expensive, and noisy sensors ."

Data: Body lean is only one physical feature that can be mined to detect difficulty. Fritz et al. (Fritz, Begel et al. 2014) showed that eye-tracker, an electrodermal activity sensor, and an electroencephalography sensor could be used to predict whether developers would find a code comprehension task to be difficult. The results could be better if we used additional equipment.

Features: It is possible that different features extracted from the tracked data could give better results. For example, a focus-out immediately followed by a focus-in could be classified as navigation to another programming window, allowing other focus events to imply going to an external application to resolve the difficulty.

Choice of machine learning algorithm: We did try another model - a hidden Markov model (HMM) - for the Kinect case, as mentioned in the first author's thesis. The decision tree gave much better results. It may be possible that an alternative model would give better results.

Lab study size and composition: We believe the results from this study do not suffer from over-fitting because (a) the duration of the task was long for a lab study, (b) the results are consistent with our earlier longer-duration and larger lab and field studies of a pure log-based scheme, (c) we used ten-fold cross-validation, and (d) we were not able to get good results by mining facial features or by using a group model for the posture-only scheme. Nonetheless, a larger study would give stronger results. Our "realistic" GUI problem used popular software such as Java and Swing, but it was one created by the experimenters. As in numerous other academic studies, participants in this study were students.

Evaluation: We used the well-accepted Weka's built-in cross-validation mechanism for calculating the confusion matrices and information gain. It would be useful to also do leave-one-out analysis in which we use for testing the data for a specific individual and for training the data for all other participants.

Hypothesized Implications for CSCW

As mentioned earlier, the driving problem for this work is providing unsolicited help to programmers. Here we consider potential relevance of this work in some of the contexts in which such help has been or can be provided. These are hypothesized implications needing, of course, validating future experiments.

Let us first consider software development involving academic or industrial programmers responsible for a common project, a task within the project, or a subtask within the task.

Face-to-face pair programming (Cockburn and Williams 2001): In such programming, one programmer, called the driver inputs a program based on constant consultation with a programmer seated next to the driver, called the navigator, who is expected to be aware of every physical characteristic and action of the driver and thus, does not need any form of computer-supported awareness to determine the difficulties of the driver.

Distributed face-to-face programming (Baheti, Gehringer et al. 2002): This is a variation of the above scenario in which a distributed navigator continuously views the remote desktop of the driver. Arguably, desktop sharing provides all the computer-supported awareness a navigator needs to determine driver difficulties – there is no need for sharing of video or inferences made from it.

Local (Nawrocki, Jasinski et al. 2005) *and Distributed* (Dewan, Agarwal et al. 2009) *Side by Side programming*: As in pair programming, two developers work on the same task and have a persistent audio channel to talk to each other. The difference is that they can work in parallel, using different workstations, on different subtasks of the task given to them. In the local case, they sit side by side so that they can constantly monitor each other's work. In the distributed case, they have a dedicated workstation to show their partner's screen, thereby ensuring that this awareness does reduce the screen real estate available for their own work (Dewan, Agrawal et al. 2010). Pair programming is a special case of side-by-side programming in which only person input at a time. When developers work in parallel, the relevance of our results depends on how much they talk to each other about their problems and how often they monitor their partner's screens and/or persons. The more they talk to each other about their problems, the less relevant our results. When monitoring their partner, our results about feature importance imply that developers should use posture changes of their partners to infer difficulties, and should also look at the workspaces of the partners, especially if the partner is calm. Finally, the cross-validation results imply that difficulty inferences made through our multimodal algorithm are reliable and thus should trigger such monitoring or conversations with the partner to validate them. Dewan, Agrawal et al have run experiments with distributed side-by-programming to determine how often partners do concurrent work, talk to each other, and create conflicts (Dewan, Agarwal et al. 2009, Dewan, Agrawal et al. 2010). While these experiments identify variations based on the pair, they show that the awareness of the other workstation was sufficient to avoid conflicts in all cases. This discussion motivates analogous experiments to determine if workstation awareness is also sufficient to manually infer all difficulties, without the need for automatic inferences.

Radical co-location (Teasley, Covi et al. 2000): Here, developers in a software team are located in a single "war-room" or "bullpen". Our implications here are variations of the ones given above. Team members who can monitor the persons of each other and should use posture changes as difficulty cues. Similarly, they

should use our algorithm for more reliable inferences. Inferences are more important in this case because of the larger group of developers who could be helped (by simply walking over to their seat).

Distributed teams: Here, the team is distributed in different offices, buildings or locations. Again, our work shows the importance of (a) monitoring posture changes (in remote video feeds) and (b) using our inference algorithm. An important difference is that posture monitoring is limited to a small number of remote developers because of the real-estate needed to display remote video feeds. Another important difference is that the cost of investigating a difficulty is high because a virtual collaborative session must be established with the remote collaborator. One way to address this problem is to accompany a difficulty inference with context useful for validating it and determining the nature of the inferred difficulty (Carter and Dewan 2018).

Let us now consider scenarios in which instructors help students with difficulties.

Homework at unscheduled times: This situation, explored in (Carter and Dewan 2015), is identical to the distributed- team case in terms of the implications of our work.

Homework at scheduled times: Here, instructors have allocated time to help remote students, and thus can constantly monitor the work of these students. Codeopticon (Guo 2015) has developed a scheme to support such monitoring. Experience with the system shows that (a) some instructors were overzealous in offering help, and (b) some help offers from the vast majority of instructors were rejected by students. Our work implies that these problems can be reduced by displaying videos and/or difficulty inferences. More important, it shows that instructors do not have to display and constantly monitor workspaces of remote collaborators. Instead, they can perform some background work during this time, and rely on contextualized inferred difficulty awareness to switch to the foreground task of helping students.

Lab-work: This is like the case above, except that there is no need to find real-estate to display the work and/or videos of the students to be helped. The need for reliable manual or automatic difficulty inference is particularly strong here as the cost of missing a difficulty is high in time-bound labs.

Cameras that give joint information are not part of the work environment today. This is why the cues given by viewing the posture changes of collaborators (who can be viewed directly or through video feeds) are important, even though these signals are weaker than the inferences of the multimodal algorithm. Another implication of our work is that controlled public environments such as labs and "war-rooms" should be equipped with such cameras. A related implication is that cameras feeding video to difficulty-inference algorithms or remote helpers should focus on the bodies rather than the faces of programmers.

Conclusions and Future Work

In comparison to other forms of awareness, inferred difficulty awareness offers the promise of opportunistic help that requires less information communication, display, and human processing. Given that, by definition, difficulty is a rare event (otherwise the worker and activity are mismatched), it makes it easier for helpers to find difficulty-needles in the haystack of collaborator-states, thereby making it easier for them to support a larger number of collaborators in difficulty, while doing their own work simultaneously. It does so by using machine-learning to automatically finding these needles. This paper has explored (a) how a state of the art difficulty-inference algorithm should be changed if the haystacks searched include not only the interaction logs but also the videos of the workers and (b) the impact of making the change.

As it is a technical paper, it has focused on the narrow issue of determining the inference to be shared, rather than user-interfaces for sharing this state and triggering opportunistic help giving, which are, arguably, orthogonal issues, addressed by our previous work.

Our work has drawn from previous research in multimodal/multimedia emption detection. It is the only such work that (a) is targeted opportunistic help, (b) addresses programming difficulty, an emotion related to but not the same as confusion, frustration, and code comprehension difficulty, (c) mines a unique and large set of workspace commands, not considered in any previous work on multimodal inference, (d) uses K-mean clustering to classify leans and combines leaning back and forward into one feature in a group model, (e) does an evaluation based on true/false positives/negatives (rather than accuracy) to account for the fact a negative emotion occurs rarely if the task is matched to the subject, (f) shows that log-mining is more effective than posture-mining, though both have high false negative rates, (g) identifies the additive value of mining commands (postures) in addition to postures (commands) and contradicts previous work, based on different metrics (true/false positives/negatives rather than accuracy), that this value is modest (D'Mello and Kory 2012), (h) gathers ground truth by asking programmers to correct predictions of the current implementation of the algorithm in a system they use for everyday work, (i) shows that the additive value of mining postures depends on whether the subject is calm or fidgety, (j) identifies the problems of mining facial features when programmers are actively programming, (k) shows that the individual model worked better in the posture-only case, and (l) does command-based rather than time-based segmentation in the multimodal/multimedia case. Its results strengthen previous findings that show that (a) multimodal effect detection is superior to unimodal detection, and (b) the predictive power of postures depends on the personality of the subject.

As D'Mello and Kory (2012) point out, given the problems of detecting emotions, no one study, no matter how large, can provide the definitive word on the additive value of mining multiple interaction modes. This is why our confirmation of similar earlier findings are perhaps as important as our new ones. This implies that further work is needed to confirm our new results.

Direct observations of the workspace and physical characteristics of collaborators (by observing them directly or through distributed communication channels) give more information than difficulty awareness. For instance, they indicate if the collaborators are frustrated or in difficulty. If such observations are to be replaced (rather than augmented) with inferred awareness, it is important to investigate multimodal inference of other emotions relevant to software development (Dewan 2015).

Moreover, it will be useful to explore the use of integrated log and video-based difficulty/emotion detection in particular and emotion-detection in general to trigger opportunistic collaboration on other kinds of activities such as writing (Long 2016).

This work provides a basis for pursuing some of these promising research directions.

Acknowledgment This research was supported in part by the USA National Science Foundation IIS 1250702 award, an Alumni Fellowship from the Computer Science Department of the University of North Carolina at Chapel Hill, the Brazilian Conselho Nacional de Desenvolvimento Científico e Tecnológico Split Research Scholar Fellowship CNPq process number 203037/2013-8, and an Intel Equipment Award.

References

Abel, M. (1990). Experiences in Exploratory Distributed Organization. Intellectual Teamwork: Social and Technological Foundations of Cooperative Work, NJ, Lawrence Erlbaum.

Baheti, P., E. F. Gehringer and P. D. Stotts (2002). Exploring the Efficacy of Distributed Pair Programming Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods - XP/Agile Universe 2002 Springer-Verlag: 208-220

Baker, R. (2007). Modeling and Understanding Students' Off-Task Behavior in Intelligent Tutoring Systems Proc. CHI.

Begel, A. and B. Simon (2008). Novice software developers, all over again. International Computing Education Research Workshop.

Calvo, R. A. and S. D'Mello (2010). "Affect Detection: An Interdisciplinary Review of Models, Methods, and Their Applications. ." IEEE Transactions on Affective Computing 1(1): 18-37.

Carter, J. and P. Dewan (2010). Design, Implementation, and Evaluation of an Approach for Determining When Programmers are Having Difficulty. Proc. Group 2010, ACM.

Carter, J. and P. Dewan (2015). Mining Programming Activity to Promote Help. Proc. ECSCW, Oslo, Springer.

Carter, J. and P. Dewan (2018). Contextualizing Inferred Programming Difficulties. Proceedings of SEmotion@ICSE, Gothenburg, IEEE.

Cetintas, S., L. Si, Y. P. Xin, C. Hord and D. Zhang (2009). Learning to Identify Students' Off-Task Behavior in Intelligent Tutoring Systems. Proceedings of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling, IOS Press: 701-703.

Cockburn, A. and L. Williams (2001). The Costs and Benefits of Pair Programming, Addison Wesley.

D'Mello, S. and A. Graesser (2009). "Automatic Detection of Learner's Affect From Gross Body Language." Appl. Artif. Intell. 23(2): 123-150.

D'Mello, S. and J. Kory (2012). Consistent but modest: a meta-analysis on unimodal and multimodal affect detection accuracies from 30 studies. Proceedings of the 14th ACM international conference on Multimodal interaction (ICMI), ACM, New York, NY, USA.

Dewan, P. (2015). Towards Emotion-Based Collaborative Software Engineering. Proc. CHASE@ICSE, Florence, IEEE.

Dewan, P. (2016). Inferred Awareness to Support Mixed-Activity Collaboration. Proc. IEEE CIC. Pittsburgh.

Dewan, P., P. Agarwal, G. Shroff and R. Hegde (2009). Distributed Side-by-Side Programming. ICSE Cooperative and Human Aspects of Software Engineering Workshop, IEEE.

Dewan, P., P. Agarwal, G. Shroff and R. Hegde (2009). Experiments in Distributed Side-by-Side Programming. ICST CollabCom, IEEE.

Dewan, P., P. Agrawal, G. Shroff and R. Hegde (2010). Mixed-Focus Collaboration without Compromising Individual or Group Work. Proc. EICS.

Fish, R. S., R. E. Kraut, R. W. Root, Ronald E. Rice, "Proceedings of the Conference and M. on Computer Human Interaction (CHI) '92, CA, May 1992, pp. 37-48. (1992). Evaluating Video as a Technology for Informal Communication. Proc. CHI.

Fogarty, J., S. E. Hudson, C. G. Atkeson, D. Avrahami, J. Forlizzi, S. Kiesler, J. C. Lee and J. Yang (2005). "Predicting human interruptibility with sensors." ACM Trans. Comput.-Hum. Interact. 12(1): 119-146.

Fritz, T., A. Begel, S. Mueller, S. Yigit-Elliott and M. Zueger (2014). Using Psycho-Physiological Measures to Assess Task Difficulty in Software Development. Proceedings of the International Conference on Software Engineering.

Grafsgaard, J. F., J. B. Wiggins, A. K. Vail, K. E. Boyer, E. N. Wiebe and J. C. Lester (2014). The Additive Value of Multimodal Features for Predicting Engagement, Frustration, and Learning during Tutoring. ICMI, ACM.

Graziotin, D., X. Wang and P. Abrahamsson (2014). "Software Developers, Moods, Emotions, and Performance." IEEE Software 31(4): 24-27.

Guo, P. J. (2015). Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming. ACM Symposium on User Interface Software and Technology (UIST).

Gutwin, C. and S. Greenberg (2002). "A Descriptive Framework of Workspace Awareness for Real-Time Groupware." Comput. Supported Coop. Work 11(3): 411-446.

Harrison, S. and S. Minneman (1990). The Media Space: A Research Project into the Use of Video as a Design Medium. Proceedings of the Conference on Participatory Design.

Hegde, R. and P. Dewan (2008). Connecting Programming Environments to Support Ad-Hoc Collaboration. Proc 23rd IEEE/ACM Conference on Automated Software Engineering, L'Aquila Italy, IEEE/ACM.

Herbsleb, J. and R. E. Grinter (1999). Splitting the Organization and Integrating the Code: Conway's Law Revisited. Proceedings of International Conference on Software Engineering.

Herbsleb, J. D., A. Mockus, T. A. Finholt and R. E. Grinter (2000). Distance, dependencies, and delay in a global collaboration. Proc. CSCW.

Hollan, J. and S. Stornetta (1992). Beyond Being There. ACM CHI Proceedings.

Hudson, S. E. and I. E. Smith (1996). Techniques for Addressing Fundamental Privacy and Disruption Tradeoffs in Awareness Support System. Proc. ACM CSCW.

Isaacs, E. and J. C. Tang (1993). What Video Can and Can't Do for Collaboration. Proc. Multimedia.

Junuzovic, S., P. Dewan and Y. Rui (2007). Read, write, and navigation awareness in realistic multi-view collaborations. Proc. CollaborateCom, IEEE.

LaToza, T. D., G. Venolia and R. Deline (2006). Maintaining mental models: a study of developer work habits. Proc. ICSE, IEEE.

Long, D. (2016). Mixed-focus Difficulty-Triggered Collaborative Writing: Interoperable Architecture, Implementation, and Evaluation, University of North Carolina at Chapel Hill.

Machardy, Z., K. Syharath and Prasun Dewan, CollaborateCom (2012). Engagement Analysis through Computer Vision. Proc. CollaborateCom, IEEE Press.

Mantei, M., R. M. Backer, A. J. Sellen, W. A. S. Buxton, T. Milligan and B. Wellman (1991). Experiences in the Use of a Media Space. Proceedings of CHI'91.

Milewski, A. E. and T. M. Smith (2000). Providing Presence Cues to Telephone Users. Proc. CSCW.

Nawrocki, J. R., M. Jasinski, u. Olek and B. Lange (2005). Pair Programming vs. Side-by-Side Programming. Software Process Improvement, Springer Berlin / Heidelberg. 3792/2005: 28-38.

Tang, J. and S. Minneman (1991). VideoWhiteboard: Video Shadows to Support Remote Collaboration. Proc. ACM CHI.

Tang, J. C. and E. Isaacs (1992). "Why do users like video? Studies of multimedia-supported collaboration." Computer Supported Cooperative Work (CSCW) 1(3).

Tang, J. C. and S. L. Minneman (1990). VideoDraw: A Video Interface for Collaborative Drawing. Proc. CHI.

Teasley, S., L. Covi, M. S. Krishnan and J. S. Olson (2000). How does radical collocation help a team succeed? Proc. CSCW.

Tee, K., S. Greenberg and C. Gutwin (2006). Providing Artifact Awareness to a Distributed Group through Screen Sharing. Proc. ACM CSCW (Computer Supported Cooperative Work).

Vail, A. K., J. F. Grafsgaard, J. B. Wiggins, J. C. Lester and K. E. Boyer (2014). Predicting Learning and Engagement in Tutorial Dialogue: A Personality-Based Model. ICMI, ACM.

Witten, I. H. and E. Frank (1999). Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann.