2015 IEEE 10th International Conference on Global Software Engineering

Approximator: Predicting Interruptibility in Software Development with Commodity Computers

Paolo Tell, Shahram Jalaliniya, Kristian S. M. Andersen,
Mads D. Christensen, Anders B. Mellson, Jakob E. Bardram
The Pervasive Interaction Technology Laboratory
IT University of Copenhagen
Rued Langgaardsvej 7, DK-2300 Copenhagen, DK
Email: {pate,jsha,mdch,ksma,anbh,bardram}@itu.dk

***Abstract*—Assessing the presence and availability of a remote colleague is key in coordination in global software development but is not easily done using existing computer-mediated channels. Previous research has shown that automated estimation of interruptibility is feasible and can achieve a precision closer to, or even better than, human judgment. However, existing approaches to assess interruptibility have been designed to rely on external sensors. In this paper, we present Approximator, a system that estimates the interruptibility of a user based exclusively on the sensing ability of commodity laptops. Experimental results show that the information aggregated from several activity monitors (i.e., key-logger, mouse-logger, and face-detection) provide useful data, which, once combined with machine learning techniques, can automatically estimate the interruptibility of users with a 78% accuracy. These early but promising results represent a starting point for designing tools with support for interruptibility capable of improving distributed awareness and cooperation to be used in global software development.**

I. Introduction

Globalization is an economical and societal trend that has pushed industries to move from local to global markets requiring practitioners to increasingly work in distributed arrangements. This change of work environment has had significant negative repercussions especially in extremely cooperative workspaces like software development. In global software development, many of the mechanisms that facilitates collaboration in co-located arrangements are absent or disrupted [15] and maintaining 'workspace awareness' (i.e., "the knowledge about others' interaction with the space and its artifacts" [12]) as well as supporting informal awareness (i.e., "the general sense of who's around and what they are up to" [12]) has became challenging and yet increasingly important.

An awareness of team members' presence is crucial in facilitating social interaction [12] as it allows people to predict whether interrupting a person at a specific time would be appropriate or not. And according to a study by Fogaty et al. [8], an 'interrupter' (i.e., the person provoking the interruption) is able to process such decision with an accuracy of 77% by simply observing the 'interruptee' (i.e., the person that is the target of the interruption [18]).

However, since workspace awareness based on physical co-presence is not an option in distributed software development, remote team members often need to rely on alternative means to compensate for the absence of such information. In 2003, Herbsleb and Mockus recognized that "[j]udicious use of presence awareness technology has the potential to substantially lower the difficulty and frustration associated with contacting a remote colleague" [16].

A common approach to gauge the availability and spread awareness information in distributed teams has been to utilize status information in instant messaging systems [27], [6]. However, more advanced systems to provide automatic assessment of distributed team members' status have been investigated (e.g., [3], [10], [18], [24], [32]). Nonetheless, most of these systems rely either on external sensors, making the system quite challenging to deploy, or on a configuration step of the system, making the initial installation cumbersome and not as straightforward as a user would hope for. Finally, these systems do not expose information to external ones, making their use limited to the user interface provided by the system itself.

In this paper, we present the design, implementation, and evaluation of Approximator: a presence and interruptibility sharing system for global software development that relies entirely on the sensing ability of a commodity laptop to estimate the interruptibility of its user. We present an experiment that replicates the one by Fogarty et al. [8] in which the accuracy achievable with Approximator is compared against the one achieved by humans. This experiment shows that the information aggregated from several activity monitors (i.e., key-logger, mouse-logger, and face-detection) provide useful data, which, once combined with an adaptive algorithm, can automatically estimate the interruptibility of users with a 78% accuracy.

In the remainder of this paper, we will review in Section II related technologies designed to predict availability and interruptibility as well as review the work by Fogarty et al. Section III will present Approximator, and Section IV will detail the interruptibility decision models manually designed. Section V presents the replication of the evaluation by Fogarty et al., and its quantitative results will be presented in Section VI. Section VII presents a discussion of the system and the results in the light of the theoretical framework of interruptibility systems presented in [18]. The paper is concluded in Section VIII.

II. Related Work

Two types of research is related to Approximator and the present study; (i) systems that have been designed to improve

978-1-4799-8409-1/15 $31.00 © 2015 IEEE
DOI 10.1109/ICGSE.2015.16

informal awareness as well as to convey and predict availability and interruptibility; and (ii) experimental studies on predicting human interruptibility.

A. Awareness, Availability, and Interruptibility Systems

Even though it has been proven that people are able to create ad-hoc practices to mitigate inappropriate interactions when collaborating closely [14], early works on media spaces technologies have investigated ways to improve such awareness and sense of presence among physically distributed co-workers to foster spontaneous interaction as normally seen among collocated co-workers (e.g, Portholes [7]). Such media space technologies have proven useful in fostering remote awareness and are now being incorporated in commercial tools like Google Hangout.

From these initial media spaces investigations, the focus in designing presence awareness systems moved to more abstract representation of such information. Possibly the first work in such direction, the Peepholes system [11] provided information about the presence of people and supported presence awareness by means of iconic presence indicators. This work showed that it was feasible to overcome most of the limitations and concerns of media spaces and provided the foundation for a new wave of systems. Contrary to the predecessors in fact, the Peepholes system did not require expensive hardware and high bandwidth to transfer video streams; and, maybe most importantly, provided a solution to the concerns related to privacy. However, the Peepholes system was only able to indicate whether a person was present or not and, even though crucial given the step forward in presence awareness systems, this was not sufficient to properly support the decision making process behind the establishment of a connection with a remote colleague. Therefore, researchers investigated the concept of interruptibility and the "tension between wanting to avoid interruption and appreciating its usefulness" [20]. Systems thereafter started to support both the manual[1] (e.g., Rear View Mirror [13] and ConNexus [30]) as well as the automatic setting of users' interruptibility levels. In the remainder of this section, we will describe a set of systems representing the latter group: availability-sharing systems that rely on an automatic assessment of the interruptibility of users to model their availability.

MyTeam [24] was one of the first attempts in automatically assessing the availability of a user. A main motivator behind this work was an early study by Whittaker [33], which showed that over 60% of all business phone calls failed to reach the intended party. MyTeam addresses this issue by providing an overview of the automatically detected presence as well as other relevant information of the users. This was done by visualizing a stripe composed by tiles each portraying a compact overview of a user (not necessarily running the client application). The tiles presented information like the user's picture, location, activity, and status. The MyTeam system collected primarily information from an active badge indoor location system to automatically update presence as well as from a keyboard and mouse activity-monitoring application on the user's primary computer. This automatically sensed status information was shown by color-coding the user's tile. Users could also manually set two additional status updates as being either 'busy' or 'invisible/unknown'. These last two options were fully controlled by the user and supported the integration of additional textual information like 'do not disturb' or 'gone for the day'. The system was evaluated by 13 participants during a four-week period with great success. However, two main drawbacks were reported: (i) the stipe could not be personalized and occupied a significant portion of the screen, and (ii) the client did not allow initiating communications directly from the user interface.

Lilsys [2] was designed to acquire data from sensors deployed in the user's office, which were aggregated to represent his or her unavailability. But this information was not directly exposed to all users of the system, thereby impacting on privacy concerns. The main sensing and data acquisition modules of Lilsys were combined together in a physical device that could be places on the user's desk. Sensors included phone, door, motion, and sound detectors as well as computer activity monitors. This information was used to infer the unavailability of the user. The physical device allowed users to turn it off, to overview the outputs from the sensors through small lights, and to temporarily override the inferred system status to the maximum unavailability level. The system was deployed for a seven months study in the office of 4 users. Results were very promising and users reported high satisfaction insomuch that some of them continued to use the Lilsys system after the study was completed. Problems reported from the study ranged from privacy concerns due to the quantity of sensors deployed in the office, to the intelligibility of the unavailability representations used in the user interface. Finally, users also reported the system to be quite demanding in terms of interaction required to access the unavailability information of users.

MyVine [10] was designed to obtain sensor data without installing any additional hardware or sensing infrastructure. The information detected by MyVine was speech though a microphone, location via network information, computer activity, and calendar entries. This information was used to model availability that was shown to users connected to the system. The system used continuous values (i.e., 0-100) to report availability level and it allowed users seeking information to gradually refine the information by interacting with the contacts. As an additional feature to previous systems, MyVine would suggest the most appropriate mean to initiate communication. This was shown via three icons below the picture of a user. MyVine was deployed in a four-week study with 26 participants. Interestingly, participants used the system chiefly as a presence indicator and less frequently to view the availability level before engaging in communication (which was typically done via another system, like IM).

Finally, InterruptMe [18]. The design of this system was informed by a six-dimension design space on availability-sharing systems, derived from an extensive review of existing systems. InterruptMe was designed to balance the tensions between interrupters and interruptees. The system differed from previous systems by presenting availability information to users on a per-colleague and per-communications medium basis. Therefore, the data collected through both background monitoring software as well as physical sensors deployed around the office space of a user were filtered based on

[1] See [27] and [6] for an investigation of the use and role of instant messaging systems in distribute teams.

user settings. The InterruptMe system was not subject to a deployment study so no evidence exists on its effectiveness in the wild.

B. Experimental Studies

Besides manipulating information that is most relevant to their current task, knowledge workers like software developers need to manage multiple tasks and cooperate effectively among several colleagues to be successful [31]. In distributed environments, in which such activities need to be performed via computer-mediated means, the establishment of the communication channel between the interrupter and the interruptee required to engage in the activity can have undesired repercussions due to the interruption happening at an inappropriate time [26], [25]. Starting from a Wizard of Oz feasibility study in 2003 [21], Fogarty et al. have presented a series of studies (i.e., [9], [10], [8], [1]) in which they thoroughly investigated the uses of a sensor-based system to automatically assess the interruptibility of humans.

One of their studies [8] shows that humans reach 77% accuracy when assessing whether or not a person is 'highly non-interruptible', whereas an assessment based on a model constructed on simulated sensors data is capable of making this same distinction with an accuracy of 82%. It is important to note that for both assessments, the baseline reachable by simply stating constantly the opposite fact (i.e., 'not highly non-interruptible') is 68%. The range of sensors used in this experiment was very wide as their goal was to maximize the accuracy of the models that could be constructed based on the automatically collected data. Sensors included keyboard and mouse activity monitors, calendar monitors, microphones, cameras, location detection systems (based on network ip), door sensors, motion sensors, phone sensors, etc.

Another study [9] also analyzed different reduced sensors combinations. In particular, they considered the scenario of a model based solely on data collected by the built-in microphone and the computer activity of regular laptops. Clearly, this scenario is extremely interesting, as it requires no physical deployment of any kind and could permit a study based on a larger deployment. The results were very promising but not consistent across the subjects; one of the chosen candidates in fact could not be assessed as accurately as the others, and the model made no use of microphone data. In discussing this, Fogarty et al. hint at the possibility that the issue could be caused by noise due to the proximity of the microphone to the laptop; however, we speculate that another explanation could be related to the different working environment of the subjects. The best results seemed to be obtained by managers who might be working in a less crowded and noisy environment making the data from a microphone good indicator of the subject activity.

III. Approximator

As discussed above, it has been shown that the sensing ability of a commodity laptop can provide useful, reliable, and usable interruptibility information [9]. Moreover, experimental results has showed that support for interruption management should not be done in a separate system, but should be built into the primary tools used for coordination and communication. These studies show that integration with the users' primary application—like an IDE for a programmer—is of paramount importance to facilitate adoption of presence and interruptibility support (e.g., [22], [10]).

Based on these two observations, the design goal of Approximator was twofold: to experiment with the sensing ability of commodity laptops to automatically estimate the interruptibility of programmers with little to no pre-configuration of their laptops, and to design an infrastructure able to expose accurate interruptibility information as a service to be utilized by other application, like an IDE or chat application.

Figure 1 depicts a high-level overview of the Approximator architecture[2]. As it can be seen, Approximator consist of three main parts:

- An *infrastructure* aggregating and classifying data, and running interruptibility predictions.
- *Activity monitors* collecting information from users' laptops.
- *Clients* subscribing to presence and interruptibility information from the infrastructure.

Fig. 1. High-level overview of the Approximator system architecture. The dashed line shows the Approximator API that allows activity monitors to send data to the infrastructure and allows clients to subscribe to presence and interruptibility information.

The central element of Approximator is the *infrastructure* component, a server component deployed to a cloud service that represent the access point for both the activity monitors pushing the data collected on users' laptops as well as third-party client applications requesting data. The outer dashed circle illustrates that the infrastructure expose an application programmer interface (API) supporting communication with activity monitors and clients.

Activity monitors are the data providers. An activity monitor is tied to a specific user, and a user can have an undefined number of monitors assigned. Activity monitors register themselves with the infrastructure, which returns a user-specific endpoint and thereafter are allowed to send raw readings to this endpoint. Finally, the APIs can be queried by *clients* to gather presence and interruptibility information inferred by the infrastructure. This information can then be utilized to e.g. visualize such information to the users.

A. Infrastructure

The main responsibilities of the infrastructure are to: (i) receive data from the remote activity monitors (see Section III-B); (ii) aggregate such data based on the models for presence and predict interruptibility (see Section IV); and (iii) make such information available through the Approximator API.

For the infrastructure, it is important to design for scalability in order to make the system usable in a global software

[2]The server side of Approximator and the client used for the evaluation are available on GitHub at https://github.com/mofus/GSE.

development context. It should be able to handle multiple activity monitors for multiple users, multiple client applications, accessed from many different geographical locations. However, since the infrastructure is responsible for inferring presence and interruptibility, it is a potential bottleneck. To address scalability, the design of Approximator is based the actor model by Carl Hewitt [17], which uses actors as main units of work that communicate among each other through message passing and do not share state. This makes it possible to easily parallelize the execution by running actors on different resources, which leverage the benefits of a cloud-based infrastructure where additional resources can be dynamically allocate when needed. Approximator is implemented in Akka[3] and Scala[4], which implements an actor model on the Java VM.

The Approximator API is implemented as a REST-based service using a WebSocket communication layer. The service creates the following endpoint:

- **/register {PUT}.** This endpoint allows users to register with a sensor- and user- name pair (String, String). It returns the endpoint that the user should send updates to, i.e., the first user will get the endpoint /user/1 and subsequent users will get a path with an increased userid value.
- **/user/id(int) {GET & PUT}.** After the register method is called the server creates two new endpoints for each user based on the path returned during the registration (e.g., /user/1 in the example above) by creating both a GET and a PUT endpoint for that specific URL. The PUT is used to provide new sensor readings; while, the GET returns the current predicted interruptibility.
- **/users {GET}.** This endpoint returns the path and current predicted interruptibility of all registered users.

B. Activity Monitors

Activity monitors gather low-level information about a user's presence and behavior. Activity monitors are in general designed to run on commodity laptops, which we define a device that is generally available for sale, runs a common PC operating system (OSX, Windows, or Linux), and is equipped with a camera, input devices (i.e., keyboard and mouse), a built-in microphone, and built-in radios (e.g., Wi-Fi and Bluetooth). To communicate with the infrastructure, activity monitors need to register via the Approximator API by providing an identification of the user being monitored. This identification is then used to aggregate sets of activity monitors together around the notion of user facilitating the retrieval and analysis process.

Looking at a commodity laptop, there is a range of sensing capabilities that can be monitored. Table I provides an overview of these. In the following, we will review these sensing capabilities and discuss their strengths and weaknesses. Table I also lists if a particular sensor was included as part of this study. As we will explain later, a combination of few activity monitors (i.e., sensing input devices and the camera) turned out to be sufficient for an adequate accuracy in interruptibility prediction.

Type	Strength	Weakness	Incl.
Input devices	Reveals user activity	No ID of user; No detection of absence	✓
Microphone	Remote sensing; ID of user	Noisy environment; No detection of absence	✗
Camera	Remote sensing; ID of user	Only in front of camera	✓
Radio	Remote sensing; sensing mobility	Unstable	✗
Process monitor	Reveals activity types	Requires user-specific knowledge	✗
Calendar	Reveals current and future activity	Requires user-specific knowledge	✗
Geo-location	Helps map activity to location	Provides little information on its own	✗

TABLE I. SUMMARY OF ACTIVITY MONITORS.

By collecting data from various input devices (e.g., keyboard, mouse, and trackpad), user presence and activity can be detected. However, information from these input devices will not enable the system to identify a specific user in front of a given machine. Moreover, information from input devices provides a very weak indicator of absence. In fact, a user could be away from the computer, talking on the phone, reading on the screen, or doing any other activity not involving the use of input devices but still be present.

Using a microphone as an activity indicator has been shown to be a very accurate information able to overcome the weakness related to absence of input devices monitors. A microphone monitor can be used to assess the presence of a person even when not interacting with the laptop. Moreover, using speech recognition techniques, voice input could be used to establish the identity of a user. However, a microphone suffers from the same limitation of input sensors, as the lack of input is not a meaningful or useful piece of information. Beside presence information, a microphone can be used to infer interruptibility; according to Fogarty et al. [8] speaking represents a situation of non-interruptibility. However, this does not take into consideration a shared, noisy, and cooperative environment such as an open office space with, e.g., software developers. In such environments there is a high chance of informal communication that hardly represents a non-interruptible situation.

Like the microphone, the camera can provide information about the presence of a user within the camera field of view even when the user is not interacting with the machine. Given the common positioning of cameras on the monitor bezel, a camera detects a user only when in front of the laptop rather than within the microphone's range. Face-recognition approaches could be used to strengthen the information and determine the identity of the user visible to the camera, and other computer vision techniques could be applied to perform basic activity recognition for inferring interruptibility.

Since users on average own about three to six computing devices [19], radio signals like Bluetooth or WiFi signals from such devices can be used to infer the users proximity to the machine. Moreover, if offices are equipped with e.g. Bluetooth beacons, the system can infer indoor location. The weakness of using radio for proximity detection and identification is the

[3]http://akka.io/

[4]http://www.scala-lang.org/

overhead and instability of radio communication and need for discoverability of e.g. Bluetooth devices.

Software process monitors can be used to sense interruptibility. Usually a process monitor is provided with a list of processes (software applications) running on the user's computer. The list consist of both work and leisure associated processes. By means of this information, it is possible to obtain information about the type of application that the user is currently using, which can provide knowledge that would help infer interruptibility. For example, if a user is using an IDE, this indicates that he is busy with software development, whereas when using a web browser, he is reading facebook. Similarly, a simple calendar monitor can help identify user's activities during the day and, in doing so, convey information about their availability. However, such monitoring of a user would require us to build user-specific models that map software processes and calendar entries with their 'interruptibility'. Such user-specific configuration works against our design goal of building a general model and these types of monitors was hence excluded from this study.

Geo-location information can be obtained from the IP address of the laptop or by using location estimation based on WiFi triangulation. Knowing the whereabouts of the machine neither infers presence nor interruptibility on its own. However, it does provide other activity monitors and the decision module with useful information regarding the geographical setting of the machine, which would allow them to reason considering the coarse user location, e.g., office, home, etc.

In summary, Table I provides an overview of the possibilities that have been considered and their strengths and weaknesses. In order to move forward with the study, we decided on only supporting a minimum set of sensors, which is in line with the study of Fogaty et al. [9] in which it is shown that significant results can be obtained with a reduced set of sensors and, in particular, those available in commodity laptops. In their evaluation, the sensors included were: mouse and keyboard monitors, process monitor, and microphone. Therefore, we decided on including sensing of input devices (mouse and keyboard) since this provide information about user activity directly on the laptop, as well as the camera to sense presence in front of the laptop as well as remote (non-keyboard/mouse) activity. As such, we wanted to investigate the accuracy of prediction of interruptibility based on these two simple activity monitors.

IV. Interruptibility Model

To estimate whether (i) a software developer is present and (ii) to assess his or her level of interruptibility, the Approximator has access to data originated from: a keyboard monitor, a mouse monitor, and a camera monitor for face detection. Raw data from these monitors alone cannot be used to assess these two characteristics of software developers. Therefore, we defined the concept of *presence* as the assessment of Approximator of a user (not necessarily the owner of the laptop) being in close proximity to the laptop. The presence feature is binary as a user can be either present, if detected by any monitor, or not present, if not detected by any monitor. The concept of *interruptibility*, instead, was defined as the assessment of Approximator of the degree to which a user detected as present can be interrupted. The interruptibility value is also binary (i.e., interruptible/non-interruptible).

Given these definitions, raw data can be used directly to assess presence; however, interruptibility requires a decision model. Initially, we attempted to define a simple model of a software developer based on preliminary tests and experience working in software development environments. It soon became clear that this model (described in the Section IV-A) was limited and could match only hardcoded work patterns. Therefore, to improve the accuracy of the estimations provided by Approximator we refined the interruptibility decision model by using machine learning techniques; this approach is detailed in Section IV-B.

A. Initial Model

The initial model comprises three main scenarios, which are assessed by a decision model component located on the infrastructure server component. The scenario-detection algorithm acts on a sliding window of incoming activity monitor data, which for this study was set to 30 seconds based on initial tests as depicted in Figure 2.

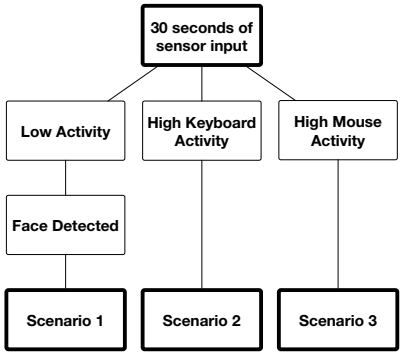

Fig. 2. Initial decision model.

Scenario #1 is inferred using face detection. When a user sits in front of the computer, the face detection monitor will detect the user as being present. Detection happens with a frontal or profile detection depending on the angle with which the camera sees the users face. This scenario is meant to capture the situation in which a user is sitting in front of the computer but does not interact with it. Common examples of this scenario are: a user reading on the screen, a user having a conversation for instance on Skype, or similar.
Scenario #2 is connected to high keyboard activity and identifies a user is doing concentrated work. Typical examples of this scenario are a user writing code or other keyboard intense tasks.
Scenario #3 on the contrary, is characterized by high mouse activity[5]. The assumption behind this scenario is that heavy mouse activity is an indicator of casual activity in software development. Examples of this scenario are web browsing, activity on the file system like moving files, or simply a switch of context between work sessions.

[5]Note. In the unlikely case of data from the mouse and keyboard monitors being equally weighted, the initial model of Approximator would have evaluated Scenario #3 due to the code structure.

After testing this decision model, we noticed that the accuracy of the system was very dependent on the test subject being analyzed. Clearly, the interruptibility decision model was unable to recognize the different work patterns of the test subjects as relying heavily on the assumption that keyboard input entails non-interruptibility and mouse events relate to interruptibility.

B. Improved Model

To tackle the problems identified with the initial decision model, we decided to utilized a machine learning algorithm. The features selected for the machine learning algorithm were computed from the mouse and keyboard of the participant's laptop. Samples were drawn at ∼5hz with a moving window size of 35 seconds of data resulting in a sample size of 175 (35x5 = 175) without any overlap between samples. We calculated the mouse mileage and used the moving average for 4 instances of data as the first feature. For the keyboard sensor, we used a binary feature in which a pressed key would generate 1 while no key event would generate 0. To calculate the second feature, we used the same moving average of 4 instances of the keyboard sensor. The classification model was built in the Weka machine-learning environment. We tried different classifiers and the NNge classifier[6] outperformed others. The performance of the classification model was evaluated using a standard cross-validation approach with 10 folds [23].

V. Evaluation

To evaluate Approximator against the results presented by Fogarty et al. in [8], we ran an experiment closely replicating theirs. Their approach, defined as a bottom-up approach, consists of three phases.

A The 'data collection' phase comprised the audio and video recording of subjects (software developers in our case) working in their actual environments. During this phase, the system was firstly gathering data from the various sensors (or activity monitors in our case) and secondly requesting test subjects twice an hour, but at random intervals, to self-assess their interruptibility level.

B The 'human assessment' phase focused on the human estimation of interruptibility. A survey was performed in which participants were asked to assess the level of interruptibility of specific segments of the recordings captured during the data collection phase.

C The 'automatic assessment' phase concerned the evaluation of the improved estimation models described in Section IV-B, which was tested against the dataset recorded during the data collection phase.

A. Data Collection

Purpose: to collect audio and video recording of software developers working in their actual environment while gathering data through the Approximator system as well as requesting periodic interruptibility self-assessments.

Subjects: two male subjects participated in this phase[7]. *Subject A*, age 25, works for a security company in which his primary task is to maintain and develop the company system. Since the company has offices in several countries, test subject A often does parallel development with the remote teams. He works in shared offices with three to four other employees depending on the day; these colleagues are always assigned to the same projects, and they share a similar set of skills. This is a rather important detail to describe his work practices as they often engage in discussions, i.e., to share knowledge. Finally, the office has a physical door that can be closed, but this is rarely done. *Subject B*, age 34, works for a software company in which his primary task is to develop server solutions. Also his company has offices in several locations and, as a company policy, it allows its employees to work from home. During the days in which he was recorded, subject B worked in his home office located in the basement of his house. Therefore, in the case of subject B all communication with colleagues happened either online, through instant messaging, or via audio chats.

Setup: a Sony Handycam HDR-CX360 with a wide-angle lens was used to record both audio and video (MP4, 720x576, 25 frames/sec.). As shown in Figure 3 and 4, the camera was positioned at an angle that: (i) did not allow for the content of the screen to be visible, (ii) captured as much as possible of the office, and (iii) clearly showed the face of the test subjects, their hands, and the input devices.

Fig. 3. Overview of the experiment setups illustrating camera position.

The Approximator system was deployed on both subjects' laptops: a Microsoft Surface Pro 3 i5 processor 8GB RAM with Windows 8.1 for subject A; and a Lenovo T440s i5 processor 8GB RAM with Windows 7 for subject B. Both machines are used by the subjects on docking stations that automatically connect the laptops to an external monitor, a keyboard, and a mouse. In both setups, the machines were connected to internet during the entire evaluation, thereby allowing the activity monitors to collect and send data to the infrastructure without any interruptions.

A small application was used to prompt both visual and acoustic notification at random intervals averaging at two times per hour. At each prompt the test subjects were required to self-report their own level of interruptibility on a scale from one (highly interruptible) to five (highly non-interruptible) by holding up the corresponding amount of fingers on one hand to the camera when prompted; alternatively, even though less preferred, a verbal assessment was also accepted. The test subjects were not allowed to interrupt or pause the application,

[6]NNge: Non-Nested Generalized Exemplars.

[7]In [8], test subjects were four for an overall recording time of 602 hours.

but could request retroactively to have sequences deleted if necessary. Recordings in which the test subjects were not present during a prompt were excluded from the evaluation. Whereas, in cases in which test subjects clearly registered the prompt but did not signal their interruptibility due to them being either on the phone or engaged in another activity requiring their full attention, the interruptibility level was manually recorded during a post-analysis as highly non-interruptible.

The data collection phase of the experiment ran two workdays from 9:00 am to 4:30 pm yielding a total of 30 hours of video and 43 prompts. Out of these 43 prompts, eight were discarded since the subjects were not present at the time of the prompt.

Interruptibility level	**1**	**2**	**3**	**4**	**5**
Frequency	9	6	3	9	8
Total number of prompts					35

TABLE II. FREQUENCY OF INTERRUPTIBILITY VALUES REPORTED BY THE TEST SUBJECTS, RANGING FROM 1 (HIGHLY INTERRUPTIBLE) TO 5 (HIGHLY NON-INTERRUPTIBLE).

B. Human Assessment

Purpose: to have all instances registered in the data collection phase assessed by humans.

Subjects: for the human assessment, we chose subjects from the software development community to partake in the survey under the supervision of two researchers. During the survey, demographic information were collected (i.e., nationality, age, and gender); these are reported in Table III.

Nationality	**#**	**Age**	**#**	**Gender**	**#**
Danish	13	18-24	7	Male	16
Romanian	3	25-34	10	Female	3
Belgian	1	35-44	1		
German	1	45-54	1		
Polish	1				
Total participants	19		19		19

TABLE III. DEMOGRAPHIC DATA OF SURVEY PARTICIPANTS BY COUNTRY OF ORIGIN, AGE AND GENDER.

Setup: Figure 4 shows the application created to peer review the collected data. Through the application, a participant was able to review a set of videoclips from the data collection phase and assess the interruptibility level of the subject shown. The same scale used during data collection ranging from one (highly interruptible) to five (highly non-interruptible) was also used in this process. The application allowed replaying the videoclip as many times as needed to feel comfortable when providing an assessment. Each sequence played in full length before the user could rate it and proceed to the next. Finally, given that the Approximator system did not include microphone data, the sound was removed from the videoclips.

Method: Participants were introduced to the survey by describing them the task using the following sentence: "If you were a colleague of these people and needed to talk to them, would you consider them interruptible?" Each participant was asked to assess one of five unique sets of videoclip with seven different video clips from the first stage of the experiment plus

Fig. 4. Screenshot of survey webpage.

three random repetitions from the same set; these were used to ensure consistency within the participant's answers. Each clip contained the last 30 seconds leading up to the prompt message requesting subjects form the data collection phase to self-assess their interruptibility; this last fragment was clearly not included in the clip. If two or more of the three repetitions deviated of at least two points from the original answer, the results from the participants were considered invalid.

C. Automatic Assessment

Purpose: to evaluate the decision models during post analysis of the dataset obtained in the data collection phase. Results were compared to the ones presented in [8] and [9].

Method: a cross fold validation with 10 folders [23] was used on 90% of the raw data logged during data collection to construct a decision model that was then evaluated on the remaining 10%.

VI. RESULTS

To compare the human estimation against the automatic prediction from Approximator, we plotted the data into three confusion matrices. The first two compare the self-assessment answers of test subjects A and B to the ones of the survey participants collected in the human assessment phase. In particular, Table V presents data including the full scale used during the data collection phase and the human assessment one (i.e., including values from 1 to 5); whereas, Table VI presents the same data but aggregated as explained below. The third matrix compares the self-assessment answers of the test subjects to the predictions of Approximator done in the automatic assessment phase. This matrix already considers the compression of the 5-point scale to binary value. By finding

correlations between hits in the self-assessment / survey participants and self-assessment / Approximator results, we can compare the Approximator performance to the human judgment, while, at the same time, relating our findings to those presented in [8].

Review Group Results: 19 participants accepted to participate in the survey. As previously mentioned, the 35 videoclips were divided into five sets (i.e., 7 unique clips and 3 repetitions for control per set). Sets A, B, C, and D were reviewed by four participants, while only three surveyed set E. All surveys were valid and had consistent answers confirming the confidence of the participants in providing their estimations.

Test Set	Adjusted	Original
A	50.99%	21.42%
B	57.14%	21.42%
C	82.14%	39.28%
D	78.57%	32.14%
E	47.61%	28.57%
Average	63.09%	28.57%

TABLE IV. ACCURACY OF SURVEY PARTICIPANTS INTERPRETING TEST SUBJECTS INTERRUPTIBILITY.

Nonetheless, although answers were consistent and participants clearly understood the task, the accuracy of the answers given was overall poor. As shown in Table IV, the average precision among all sets is 28%. We attribute this low score to the difficulty of precisely rating another person's interruptibility on a scale from one to five. To account for this, we used a scale conversion similar to the one used by Fogarty to obtain a binary representation of the data. However, differently from Fogarty, which converted 5 as 5 and all other values as 1, to maximize accuracy we mapped values 1-2-3 to 1 and 4-5 to 5. As shown in Table IV, this approach lead to a significant improvement of the average accuracy, which increased to 63%.

During the data analysis, we compared the result from the survey participants to the data from the self-assessment from the test subjects A and B. The resulting confusion matrix is reported in Table V; rows correspond to the values reported by the test subjects, and columns correspond to the values from the survey participants. The unshaded diagonal represents instances in which the estimator subject correctly estimated the same value given by the test subjects gathered during the data collection phase. This matrix shows that the overall accuracy (calculated by summing up the percentages from the main diagonal) lies at 30.08%, and the accuracy within one (calculated by summing up the percentages from the three central diagonals) is 62.41%.

Approximator Results: the activity monitors from the Approximator system collected data throughout the 30 hours of the experiment resulting in a total collection of 86,823 mouse events, 16,151 face detection events, and 28,629 keyboard events. In this data analysis, only the events related the videoclips shown to the survey participants were considered (see the distribution overview in Table VII). Based on this data, we performed the same analysis used for the survey data.

The performance of the classification model was evaluated using a standard cross-validation approach with 10 folds [23]. Our classifier was able to classify 77.77% of the instances correctly. The confusion matrix and a detailed overview of the classification performance are presented in Tables VIII and IX.

		Estimator Subject Value				
		Highly Interruptible				Highly Non-Interruptible
		1	2	3	4	5
Test Subject Value	1	12 9.02%	9 6.77%	11 8.27%	3 2.26%	0 0%
	2	4 3.01%	6 4.51%	8 6.02%	5 3.76%	1 0.75%
	3	3 2.26%	0 0%	4 3.01%	3 2.26%	2 1.50%
	4	4 3.01%	5 3.76%	9 6.77%	12 9.02%	4 3.01%
	5	7 5.26%	5 3.76%	4 3.01%	6 4.51%	6 4.51%

TABLE V. CONFUSION MATRIX COMPARING THE ESTIMATOR SUBJECT VALUES TO THE VALUES OF THE TEST SUBJECT. OVERALL ACCURACY: 30.08% ACCURACY WITHIN 1: 62.41%.

		Estimator Subject Value	
		Other Values	Highly Non-Interruptible
Test Subject Value	Other Values	57 42.86%	14 10.53%
	Highly Non-Interruptible	34 25.56%	28 21.05%

TABLE VI. REDUCED CONFUSION MATRIX COMPARING THE ESTIMATOR SUBJECT VALUES TO THE VALUES OF THE TEST SUBJECT. OVERALL ACCURACY: 63.91%.

VII. DISCUSSIONS

This study followed the approach and evaluation technique of Fogarty et al. [8]. Differently from their work, this study investigated software developers and collected data by leveraging only the sensing ability of common laptops and, in particular, used a reduced data set from input devices and the camera. The study also showed that static ad-hoc models cannot be used to assess different work interruptibility, while machine learning techniques can be used effectively to provide customized models fitting individual users. The analysis of automatic assessment of the interruptibility level of humans obtained an accuracy very close to the ones reported by Fogarty et al. in [8] and [9]. We demonstrated that interruptibility can be predicted automatically as accurately as or better than a human (i.e., 78%) using very little training of the machine-learning model (i.e., 175 seconds of recorded data).

Figure 5 provides an overview that compares the results from the Fogarty et al. study (left-hand side) with the results obtained in our study (right-hand side). Our results are promising when comparing them firstly with the accuracy of human estimations reported to be 77% [8] and secondly to the decision models using only sensors present in laptops (including process monitors) used in [9], which reached an accuracy ranging from 78.9% to 81.6% depending on the work patterns of the subject observed. In other word, Approximator is able to predict interruptibility at the same level, or better, than a human, and obtain almost similar accuracy as the study by Fogarty et al. while using fewer sensors. This result has significant repercussions as it shows that human assessment can be substituted by an automatic one.

Event Type	Test Subject 1	Test Subject 2
Mouse events	984	534
Keyboard events	319	218
Face detection events	244	113
Total	1547	865

TABLE VII. DISTRIBUTION OF EVENTS COLLECTED BY APPROXIMATOR DIVIDED OVER THE TEST SUBJECTS.

		Estimator Subject Value	
		Other Values	Highly Non-Interruptible
Test Subject Value	Other Values	25 69.44%	1 2.77%
	Highly Non-Interruptible	7 19.44%	3 8.33%

TABLE VIII. CONFUSION MATRIX PRODUCED IN THE DECISION MODEL REFINEMENT. OVERALL ACCURACY: 77.77%.

Even though initial, these results represent a starting point for designing support for interruptibility into tools for communication, coordination, cooperation, and distributed awareness to be used in global software development. As initially described, the architecture of Approximator was designed as a cloud-based service that client applications can access and get interruptibility information from.

One usage of Approximator is to utilize interruptibility information in communication tools such as instant messaging (IM), email, and video communication tools. For example, studies of IM use has shown that practitioners often adopt ad-hoc practices to better coordinate availability with distant colleague [27], [6] even when the very same IM systems provide features to set availability/status information. In fact, in practice, it is often the case that such manual setting is not utilized, which results in people being constantly in a 'do-not-disturb' status throughout their working hours. Integrating a system like Approximator would allow setting the status of IMs both automatically and in a much more fine-grained manner throughout a working day.

Similarly, several extension of integrated development environments (IDEs) have been proposed that adds features for supporting team cooperation and coordination. For example, JazzBand from the Jazz Research Project [4] helps improve team awareness, Palantir [29] detects parallel code changes, and Lighthouse [5] supports team coordination by suggesting potentially conflicting changes. Such systems would be able to integrate the interruptibility information provided by Approximator allowing users of their technology to, for instance, assess whether a colleague would be available for clarifying a code conflict.

The present study also has limitations. First, the study is very limited only involving two subjects and the results are only generalizable to a limited extend. Much more data sampling and training of the prediction model would be needed to provide a more stable model. Second, the accuracy of the prediction is still fairly low and is only predicting interruptibility on a very coarse-grained binary level (i.e., highly non-interruptible or "other" as shown in Table VIII). Third, one of the goals of the investigation reported in this article was to minimize the set of monitors utilized by Approximator; however, some of the ones described in Section III-B represent

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Other Values	0.962	0.7	0.781	0.962	0.862	0.631
Highly Non-Interruptible	0.3	0.038	0.75	0.3	0.429	0.631
Weighted Average	0.778	0.516	0.773	0.778	0.742	0.631

TABLE IX. DETAILED ACCURACY BY CLASS.

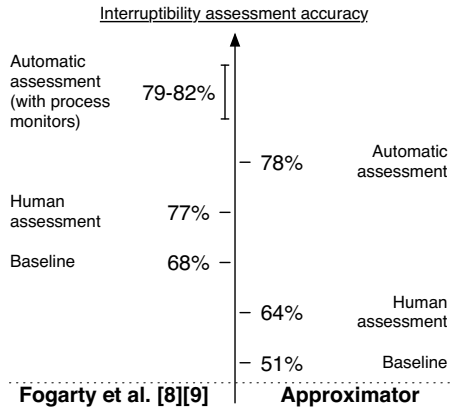

Fig. 5. Summary of interruptibility assessment accuracy reported in the studies by Fogarty et al. [8], [9] and the ones achieved with Approximator. Note: the value for the baseline represents the result obtainable by selecting always the option of non highly non-interruptible and is calculated by summing up all the values from the dataset not falling in the highly non-interruptible group, i.e., including 1-2-3-4 for [8] and 1-2-3 for ours.

interesting candidates for further investigation. In fact, the use of a machine learning algorithm would allow be able to handle personalization of the incoming data. This would allow the same model to take into account personalized information about, e.g., software processes and calendar entries.

VIII. CONCLUSIONS

In this paper, we have presented Approximator: a system designed to estimate the interruptibility of a user based exclusively on the sensing ability of commodity laptops. Our experimental results show that data aggregated from several activity monitors (i.e., key-logger, mouse-logger, and face-detection) in combination with a machine learning technique can be used effectively to automatically estimate the interruptibility of software developers with a 78% accuracy. These early but promising results represent a starting point for designing support for interruptibility into tools for distributed awareness and cooperation to be used in global software development. Our future directions are focused on strengthening such results and integrating Approximator with commonly used IMs and IDEs to test the impact of automatic interruptibility assessment in a larger globale software development setup.

ACKNOWLEDGMENT

This research has been supported by the Danish Agency for Science, Technology and Innovation under the project "Next Generation Technology for Global Software Development" (NexGSD), #10-092313.

REFERENCES

[1] D. Avrahami, J. Fogarty, and S. E. Hudson. Biases in human estimation of interruptibility: effects and implications for practice. In *CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Request Permissions, Apr. 2007.

[2] J. B. Begole, N. E. Matsakis, and J. C. Tang. *Lilsys: Sensing Unavailability*. ACM, Nov. 2004.

[3] S. A. Bly, S. R. Harrison, and S. Irwin. Media spaces: bringing people together in a video, audio, and computing environment. *Communications of the ACM*, 36(1), Jan. 1993.

[4] L.-T. Cheng, S. Hupfer, S. Ross, and J. Patterson. Jazzing up Eclipse with collaborative tools. In *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, pages 45–49, New York, NY, USA, 2003. ACM.

[5] I. A. da Silva, P. H. Chen, C. Van der Westhuizen, R. M. Ripley, and A. van der Hoek. Lighthouse: coordination through emerging design. In *Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange*, pages 11–15, New York, NY, USA, 2006. ACM.

[6] Y. Dittrich and R. Giuffrida. Exploring the Role of Instant Messaging in a Global Software Development Project. In *Global Software Engineering (ICGSE), 2011 6th IEEE International Conference on*, 2011.

[7] P. Dourish and S. Bly. Portholes: supporting awareness in a distributed work group. In *CHI '92: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 541–547, New York, New York, USA, June 1992. ACM Request Permissions.

[8] J. Fogarty, S. E. Hudson, C. G. Atkeson, D. Avrahami, J. Forlizzi, S. Kiesler, J. C. Lee, and J. Yang. Predicting human interruptibility with sensors. *Transactions on Computer-Human Interaction (TOCHI*, 12(1):119–146, Mar. 2005.

[9] J. Fogarty, S. E. Hudson, and J. Lai. Examining the robustness of sensor-based statistical models of human interruptibility. In *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Request Permissions, Apr. 2004.

[10] J. Fogarty, J. Lai, and J. Christensen. Presence versus availability: the design and evaluation of a context-aware communication client. *International Journal of Human-Computer Studies*, 61(3):299–317, 2004.

[11] S. Greenberg. Peepholes: low cost awareness of one's community. In *CHI '96: Conference Companion on Human Factors in Computing Systems*. ACM Request Permissions, Apr. 1996.

[12] C. Gutwin, S. Greenberg, and M. Roseman. Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation. In *Proceedings of HCI on People and Computers XI*, pages 281–298, London, UK, 1996. Springer-Verlag.

[13] M. Handel and J. D. Herbsleb. *What is chat doing in the workplace?* ACM, Nov. 2002.

[14] R. Harr and M. Wiberg. Lost in translation: investigating the ambiguity of availability cues in an online media space. *Behaviour & Information Technology*, 2008.

[15] J. Herbsleb. Global software engineering: The future of socio-technical coordination. *Future of Software Engineering*, pages 188–198, 2007.

[16] J. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, 29(6):481–494, June 2003.

[17] C. Hewitt, P. Bishop, and R. Steiger. A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, pages 235–245, San Francisco, USA, 1973. Morgan Kaufmann Publishers Inc.

[18] J. D. Hincapié-Ramos, S. Voida, and G. Mark. A Design Space Analysis of Availability-sharing Systems. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 85–96, New York, NY, USA, 2011. ACM.

[19] S. Houben, P. Tell, and J. E. Bardram. ActivitySpace: Managing Device Ecologies in an Activity-Centric Configuration Space. In *ITS '14: Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*. ACM Request Permissions, Nov. 2014.

[20] J. M. Hudson, J. Christensen, W. A. Kellogg, and T. Erickson. "I'D Be Overwhelmed, but It's Just One More Thing to Do": Availability and Interruption in Research Management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 97–104, New York, NY, USA, 2002. ACM.

[21] S. Hudson, J. Fogarty, C. Atkeson, D. Avrahami, J. Forlizzi, S. Kiesler, J. Lee, and J. Yang. Predicting human interruptibility with sensors: a Wizard of Oz feasibility study. In *CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 257–264, New York, New York, USA, Apr. 2003. ACM Request Permissions.

[22] M. Jiménez, M. Piattini, and A. Vizcaíno. Challenges and improvements in distributed software development: a systematic review. *Advances in Software Engineering*, 2009:3:1–3:16, 2009.

[23] R. Kohavi and others. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, 1995.

[24] J. Lai, S. Yoshihama, T. Bridgman, M. Podlaseck, P. B. Chou, and D. C. Wong. MyTeam: Availability Awareness Through the Use of Sensor Data. In *INTERACT*. Citeseer, 2003.

[25] G. Mark, V. M. Gonzalez, and J. Harris. No task left behind?: examining the nature of fragmented work. In *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2005.

[26] G. Mark, D. Gudith, and U. Klocke. The Cost of Interrupted Work: More Speed and Stress. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*, New York, USA, 2008. ACM.

[27] B. A. Nardi, S. Whittaker, and E. Bradner. Interaction and outeraction: instant messaging in action. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM Request Permissions, Dec. 2000.

[28] T. Niinimä andki, A. Piri, C. Lassenius, and M. Paasivaara. Reflecting the Choice and Usage of Communication Tools in GSD Projects with Media Synchronicity Theory. In *Global Software Engineering (ICGSE), 2010 5th IEEE International Conference on*, pages 3–12, Aug. 2010.

[29] A. Sarma, D. F. Redmiles, and A. Van Der Hoek. Palantir: Early Detection of Development Conflicts Arising from Parallel Code Changes. *Software Engineering, IEEE Transactions on*, 38(4):889–908, July 2012.

[30] J. C. Tang, N. Yankelovich, J. Begole, M. Van Kleek, F. Li, and J. Bhalodia. ConNexus to awarenex: extending awareness to mobile users. In *CHI '01: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Request Permissions, Mar. 2001.

[31] S. Voida, E. D. Mynatt, B. MacIntyre, and G. M. Corso. Integrating virtual and physical context to support knowledge workers. *Pervasive Computing, IEEE*, 1(3):73–79, 2002.

[32] R. Want, A. Hopper, V. Falcão, and J. Gibbons. The active badge location system. *ACM Transactions on Information Systems (TOIS)*, 10(1):91–102, Jan. 1992.

[33] S. Whittaker. Rethinking video as a technology for interpersonal communications: theory and design implications. *International Journal of Human-Computer Studies*, 42(5):501–529, May 1995.