Opportunities for Automating Email Processing: A Need-Finding Study

Soya Park
MIT CSAIL
soya@mit.edu

Amy X. Zhang
MIT CSAIL
axz@mit.edu

Luke S. Murray
MIT CSAIL
lsmurray@mit.edu

David R. Karger
MIT CSAIL
karger@mit.edu

ABSTRACT

Email management consumes significant effort from senders and recipients. Some of this work might be automatable. We performed a mixed-methods need-finding study to learn: (i) what sort of automatic email handling users want, and (ii) what kinds of information and computation are needed to support that automation. Our investigation included a design workshop to identify categories of needs, a survey to better understand those categories, and a classification of existing email automation software to determine which needs have been addressed. Our results highlight the need for: a richer data model for rules, more ways to manage attention, leveraging internal and external email context, complex processing such as response aggregation, and affordances for senders. To further investigate our findings, we developed a platform for authoring small scripts over a user's inbox. Of the automations found in our studies, half are impossible in popular email clients, motivating new design directions.

CCS CONCEPTS

• **Social and professional topics** → **Automation**; • **Information systems** → *Email*;

KEYWORDS

email; task management; personal information management

ACM Reference Format:
Soya Park, Amy X. Zhang, Luke S. Murray, and David R. Karger. 2019. Opportunities for Automating Email Processing: A Need-Finding Study. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3290605.3300604

CHI 2019, May 4–9, 2019, Glasgow, Scotland UK

ACM ISBN 978-1-4503-5970-2/19/05...$15.00
https://doi.org/10.1145/3290605.3300604

1 INTRODUCTION

In the 50 years since email was invented, it has become not only a ubiquitous tool for private and group communication [6, 32, 58] but also a place for managing personal information [10], activities and events [5, 56], and tasks [12]. As a result, email today has become closely associated with work, and for many people, a majority of their workday is spent within email [21, 56].

Given the workload it generates, there has been a long-standing desire to *automate* various aspects of email processing, reaching at least as far back as `procmail`, released in 1990, which let authors write regular-expression scripts to filter email into chosen folders. Over time, richer automation functionalities have emerged. For example, Boomerang [8] permits users to defer received emails, pushing them off to be re-received at a later date. Different apps offer different automation features, forcing a user to juggle a suite of 3rd party plugins to manage all their needs, rely on a human assistant, or simply do everything manually [5].

Approach. In this work, we sought to explore the breadth of needs that users have regarding automating their email experience, with the eventual goal of designing a useful, general purpose email automation system. We conducted this work through a series of three distinct need-finding probes as shown in Table 1. First, we aimed to gather from a broad audience of users wishful thinking *ideas* for automating email in their own inboxes. From a formative design workshop with 13 email users who were asked to brainstorm commands in natural language that they would like to execute, we developed a series of common categories of needs, including attention management and prioritization, filing and labeling, automated content processing, and rules based on contextual modes. These categories informed the design of a survey that we distributed to 77 additional people. The survey contained sections dedicated to each category with open-ended prompts asking for more ideas for automations users would like to perform.

Moving from users' ideas regarding automation, we next sought to identify the ways in which users were *already* automating aspects of their email to see which needs have been addressed. As automation capabilities available to end users

in many email clients are limited, we instead examined automations implemented by programmers, who have greater ability to carry out their desires for personalized automation. We conducted this probe by mining public repositories on Github that make use of the IMAP (Internet Message Access Protocol) API. From around 500 scripts, we developed a taxonomy of 8 types of automated email processing tools.

Even for programmers, writing a standalone mail automation program is a substantial task. This could deter many automations from ever being created. Therefore, informed by these two probes, we developed a platform called YouPS that makes it possible for users to write, test, debug, save, and continually execute simple Python scripts that can manipulate emails arriving in their inbox. YouPS exposes a small set of Python functions for basic actions such as accessing an email field or moving an email to a particular folder, hiding all the complexity of turning these function calls into invocations of the IMAP API to manipulate the users' email on their server. This platform served as a probe into what kinds of scripts email users would write given the opportunity to *easily* test and run them over their own email. We invited 12 subjects to write and execute scripts on YouPS for a week; we then examined their scripts and interviewed them about their experiences.

Results. Our design workshop identified several common categories of needs in email automation, which were elucidated through our later studies. The first was a *richer data model* for an email to capture latent structured information such as priority, topic, deadline, and need for a reply. The next was automation leveraging the *context* of an email, both within and beyond the email inbox. Some examples include the time of day, characteristics of the email thread (e.g., previous replies by a recipient, number or rate of responses by others), and the state of the recipient (e.g., busy, sleeping, in the office, on vacation). These attributes and contextual information are needed to drive automations that help recipients with *attention management* of email through the configuration of different forms of notifications and presentations of messages. In an interesting inversion, we also found that *senders* wanted to leverage rich data and context to *reduce load on recipients*, for example by automatically delaying an email from being sent until the recipient is in a not-busy context or at a suitable location. Finally, users sought *automated content processing*, for example to aggregate replies to an invitation or to extract attached photos into a relevant storage location.

From these findings, we consider how email systems could better allow users to automate and customize email handling. We found that many users' needs required information and affordances that are not currently available in email systems. From our three studies, 47%, 90%, and 40% respectively could not be expressed using common email clients today, for instance with Gmail and Outlook tools for writing filter rules. Thus, meeting those needs requires new features within email systems, such as expanding email's data model to add latent structure, as well as incorporating more context. In addition, systems need new mechanisms for users to be able to express how they want emails to be presented and processed.

Probe Method	# of Participants or Scripts	% of Needs Impossible to Express in Current Interfaces
Open-ended survey	77	47.1%
Large-scale email scripts review	499	90.6%
Email programming deployment	12	40.4%

Table 1: Three different probes of email need-finding.

On the other hand, we also found that over half of the automation examples from the survey and the deployment of YouPS could already be implemented in today's email systems, as shown in Table 1. However, some implementations are not straightforward, involving hacks that repurpose existing email features like the read/unread signal. This lack of usage coupled with workarounds using existing features suggest that email users are not satisfied with existing tools for automation in current email interfaces. We touch on future work towards designing interfaces for general purpose email automation authoring.

2 RELATED WORK

Email management has been explored through many different lenses by the HCI community. Some of the categories identified in our probes have been touched on by prior work.

Organizing Email. While there has been substantial research on email users, much of it has focused on organization and retrieval needs rather than automation needs. Email users view email as a repository for personal information management (PIM) [55], where they have different strategies for retrieving information [3, 40, 47, 49, 51]. For instance, users frequently send emails to themselves as a way to store information [10]. Szóstek et al. [50] identified six needs for information organization: email annotation, reliable structure, prioritizing emails, informative overview, flexible sorting, and efficient search. Finally, others study usage beyond PIM, including shared inboxes jointly accessed by a team [39]. Our research builds upon this work by identifying additional needs as well

as considering how systems could address such needs, given that current email clients do not fully address them.

Systems for Triaging Messages. When it comes to existing automations, automatic classification is one that is widely offered in email clients. Many email clients automatically classify and prioritize emails using machine learning techniques [25, 29, 57]. Beyond spam classification, clients like Gmail offer additional classifications by default such as "Social" or "Promotions". However, users may wish to classify their emails in a different or more fine-grained manner than the default classes email clients provide. So many clients provide simple explicit rule-authoring interfaces for end-users and programmers [23, 44]; for example, users can specify rules for placing messages in folders.

While most existing filter interfaces are focused on explicit metadata within messages, other ways of classification and sorting of messages have been proposed [15, 42, 50]. For instance, research has found that email users tend to see messages as tasks and have a desire to conceptualize email as a task management tool [4, 13, 27]. In this work, we provide empirical examples of the kinds of alternative classifications, prioritizations, and actions following from these that email users would like, in order to guide the design of future systems. We also demonstrate how desired classifications and priorities can be time-varying based on the user's state.

Automation for Recipients & Senders. Email users are both senders and recipients, and systems provide automation capabilities addressing both roles. For instance, email clients provide two types of reminders. Email users can set a reminder for messages to get back to it later (reminder as a recipient) and remind their recipients to solicit responses (reminder as a sender) [1, 8, 18, 19, 34]. There are also platforms that allow users to write simple triggers and actions to integrate with different applications [26, 37].

As recipients, users want to easily draft responses [28], automatically adjust views for different email data [16], and organize cluttered messages [17]. Email users also want to aggregate replies to manage bulk emails [30] and automatically process responses using pre-defined queries [36]. Senders on the other hand would like to hint to their recipients how to respond [20]. Borenstein et al. [9] suggested Tcl language-enabled email to deliver executable contents (e.g., a prompt window) to recipients. In this work, we draw from and build upon this prior work to explore the general space of email automation for both email recipients and senders.

3 PROBE 1: WISHFUL THINKING

To identify common categories of desired automations that are difficult or impossible to recreate with current email inbox technology, we began by asking email users to come up with ideas for ways they would like to automate their email that they were not already doing. As open-ended articulation of needs can be challenging for users, we began with a small, in-person, formative workshop to develop initial categories of needs that then informed the structure, questions, and examples in a survey that we distributed at a larger scale. The survey aimed to determine whether the needs identified in the workshop would be reflected in a broader and more diverse population as well as gain deeper insight into user desires for automation.

Formative Design Workshop. We conducted a design workshop with 13 participants (10 females, 3 males). Subjects were all computer science students with knowledge of programming. We asked them to use natural language or pseudocode to write email rules that they would like to execute on their inbox as either senders or receivers of email, invoking any methods or objects that they could imagine existing. Over the course of 45 minutes, participants came up with 42 different rules. Afterwards, the authors as a group analyzed the rules that participants generated and identified five major categories of needs.

Survey. Based on the needs identified from the design workshop, we designed a survey to explore each category of needs in more depth. We asked open-ended questions about how the user would like to automate their email. To prompt users to think outside the realm of what is currently possible but also be explicit in their descriptions, we told participants to imagine using a *smart robot* that can organize their email and change how it is presented to them. The questions offered rules drawn from the workshop as examples. The survey was structured into eight randomly ordered sections based on the needs identified in the workshop:

- Triaging and prioritizing incoming emails
- Moving incoming emails to different locations
- Labeling emails with richer data as sender or receiver
- Sending email only at a particular time or context
- Sending email to only the right people
- Different email modes that can be toggled (vacation, etc.)
- Altering the presentation of emails in the inbox
- Aggregating or processing multiple emails or email replies

The survey was distributed through various mailing lists within a private university and by word of mouth. It was taken by 77 people (31 females, 41 males, 1 other, rest unanswered). The median age range was 20–29. While most of our respondents were affiliated with the university, 34.5% of our respondents were not students or faculty. In addition, 48% of respondents did not have technical backgrounds, instead working as administrative staff or as students in non-technical majors. All of our participants fall into the general category of "knowledge workers", an important category that has been the focus of much study [11, 46].

We used a grounded theory approach [48] to identify themes in the survey answers. Two authors individually open-coded the responses, then discussed them after the fact, merging similar codes and grouping codes into distinct themes. To validate the merged codes, we selected 10 survey participants at random, and the two authors independently re-labeled each of their survey responses using the merged codes. From the 90 responses that were coded, we found an inter-rater reliability (IRR) of 79% using Cohen's κ.

Results

We describe the main categories of needs uncovered in the workshop and the analysis of the different survey responses within each category. In the survey, we compared responses between academics and non-academics, as well as between people with and without technical backgrounds, overall finding no major differences between these groups. Thus, we present survey results across all participants.

Richer Data Models. Many of the email rules that participants devised required access to *latent* information about the message that was not explicitly given in the email headers or in attributes of each message like sender or date. From the survey, we identified the following desired latent data: progress (e.g., pending, done), deadline, topic, priority, task. Users desired *automatic annotation* of messages with these latent attributes that could be used to drive rules. For example, there were rules such as: "`If the content of the message seems like it requires an action, label it as 'pending'`" and "`If the email has a deadline, tag the deadline`".

Because prioritization was a commonly articulated need in the workshop, in the survey we asked users to describe attributes that signal priority specifically. Respondents used information about the sender, assigning high priority for messages from important contacts and lowering the priority of automated or blast emails. Respondents also wanted to prioritize by the number of follow-up tasks, so a message requiring no action (e.g., an FYI) would have a lower priority. Finally, 14% of users wanted to take into account their previous interactions with a person or organization when prioritizing an email. For example, some respondents wanted to have emails marked as high priority if they had sent responses to the last several emails from the same sender and low priority if they had not opened or sent a response.

Using Internal/External Context. While some attributes of an email such as deadline can be determined by examining the message in question, others require access to a broader *context* [2, 53] around the message. This includes both *internal context* involving other messages within the inbox of the sender or receiver generally, as well as *external context* [60] regarding the state of the user and the world (which is often time-varying). For example, the rule "`Send a message if I haven't touched base for a month`" requires interaction history with the recipient of the message (internal context), as mentioned in the prior category. Another state that was often referenced was whether a recipient had replied to the original message: "`If the recipient hasn't replied for n days, send them a reminder`".

In contrast, "`Don't notify emails from these campus mailing lists during my summer vacation`" requires external context that involves determining whether the arrival time of an email is during a user's summer vacation. The external context may also change more frequently, for instance when one respondent said: "*I don't think priority is a consistent definition. For 30-40 hours per week, my research, colleagues, advisors are high on my priorities. However, outside of that time, my priorities shift around a little: my side project becomes my focus...one idea is to have these incoming messages reflect this ebb and flow of priorities.*" Motivated by several rules during the workshop referencing different external contexts, we asked survey participants to come up with *email modes* that could trigger different behavior in their inbox. Respondents thought of many types of modes:

> *conference, work, vacation, home/family, busy, study, distraction, class, important, person, block-/harassment, waiting, application, sleep, weekend, evening, summer, semester, daytime, emotion, recreation, meeting, ignore*

Each of these modes came with distinct rules for behavior. For example, a user's conference mode had a rule that highlighted only conference-related emails (e.g., meet-ups, announcements). The emotion mode came from a user who proposed having different modes that would kick in depending on their mood, such as when they felt anxious or tired.

Attention Management. Many users' needs for richer data models and context related to the end goal of *managing attention* as a recipient of email. Users described many ways to leverage these priorities to change the notification behavior or presentation of emails, including (automatically) marking them as read or unread, moving them to other less or more attended folders, hiding emails until a particular time, and bringing them to the top of the inbox or pushing them farther down. The way users repurposed interface affordances for attention management—for example, marking a read message as unread in order to make it stand out—raises the question of whether there might be other attention-getting techniques that would be a better fit.

How to configure *push notifications* also arose frequently, reflecting that email (once an asynchronous communication medium) has taken on some attributes of more synchronous communications such as instant messaging. Users overall did not want to be notified about every email, but did want to

be immediately notified about certain prioritized emails [35]. Users considered a variety of push notification channels such as buzzing the phone and sending SMS messages. Other users came up with automatic rules to mark-as-read emails that should *not* trigger such notifications. Still others wanted to mute notifications entirely or receive digest notifications after they had accumulated for a while instead of for every email. Examples of notification rules users wrote include:

- `Don't notify me about emails from these campus mailing lists during my summer vacation`
- `If this sender sends me too many emails within a short period, mute emails from the sender`

Sender Affordances. While email research generally focuses on users reducing their *own* email workload, evidence shows that users care significantly about the impact their email sending has on *others'* workloads, and seek ways to reduce that impact [58]. Some subjects asked for a reaction or upvote feature to signal that they saw a message without burdening the recipient with another response. As another example, senders wanted to be able to set emails to expire at a certain time or when other conditions were met: "`For event announcement emails, self-destruct the emails from recipients' inbox after the event`". Context regarding a recipient's availability or responsiveness [52] could also be used by a sender to decide when or how to send a message. Unfortunately, today, such context is often delivered to senders in a catchall fashion using auto-replies (e.g., out-of-office replies), which forces an extra message. For instance, one person wanted to send an automatic response to incoming emails if their inbox had surpassed a certain number of unread emails.

As an additional way to reduce load on recipients, senders wanted a way to target only the right recipient *subset* of a mailing list. In particular, in the workshop there was a high demand for querying particular groups of recipients (e.g., people I have exchanged email with more than three times, people who reside in a particular city, people in my address book). These queries typically required metadata from previous emails (internal context) or even from 3rd party applications such as a calendar (external context):

- `If I reply to an email thread, send my reply only to those who expressed interest`
- `Send only to people in lab right now because I need a stapler`

Recognizing that recipients may not always be paying attention to the group thread or mailing list, several users also wanted the *tagging* feature common in social media to signal importance to certain recipients.

Altering Inbox Presentation. In addition to the existing capabilities in email clients for managing presentation, such as moving, hiding, or sorting emails, we asked survey participants to consider potential *new* presentations of their inbox [14, 22, 54]. We found many interested in alternative views for email threading. Participants suggested different presentations such as a tree-like network visualization of replies per email thread, echoing prior work [45]. Participants also wanted to break up long messages, group together short messages, or group by different attributes like sender. One participant said: "*In Slack, you can send individual sentences/thoughts one-by-one as you're typing a larger message, so it's easier to divide big thoughts into bite-sized chunks. I mean, who wants to send a one-line email?*" Connected to message segmentation, another participant said she'd like to be able to reference or quote from prior emails so that recipients could follow the source.

Content Processing and Aggregation. Finally, we found many rules regarding complex processing of content within emails in order to make decisions about forwarding emails to other people [33] or extracting information to send to other platforms or store in other formats. For instance, several subjects said that they wanted to extract event information to store within a calendar application. Other examples include:

- `If I forward this email to a certain email address, parse the content and add it as a note`
- `If a message contains attachments, add them to one of my cloud folders`

Survey respondents additionally came up with ideas for *response aggregation* [30]. For example, when scheduling a meeting, attendees can either respond to everyone, or to the sender, but a sender cannot collectively aggregate responses into a poll. Respondents said they would like to see a summary of a group of responses rather than a series of different emails containing all of the original responses. Also, others mentioned interest in some sort of chart, calendar, or other aggregate visualization of responses. Besides the novel presentation required, this demands a richer data model for email to *represent* the data to be aggregated.

4 PROBE 2: EXISTING AUTOMATION SOFTWARE

Our second probe aimed to identify email automation needs that (programmer) users have taken into their own hands to address. To do so, we mined Github to identify and examine code written to automate email processing. In contrast to our first probe which only called for (lightweight) wishful thinking, this probe allows us to identify needs that were important enough to actually implement. Of course, our focus on code means our user population has narrowed to programmers. Probe 1 did suggest that overall, programmers and non-programmers recognize similar needs, so seeing what programmers implemented may suggest automations of value to non-programmers. We also did find that Probe 2

Category	Count
Triggering actions based on content	134
Altering default presentation of email client	118
Archiving emails to new locations	110
Email as middleware	48
Customized notifications	39
Email analytics and productivity tools	25
Detecting and removing spam	25
Inbox cleanup and "Inbox Zero"	23

Table 2: 8 major functions of email scripts on Github.

revealed some needs that a non-programmer may not have or may not realize can be resolved by automation.

Method. On Github, we searched for code that made use of IMAP (Internet Message Access Protocol), the standard protocol for connecting to a user's mail-server account and accessing and manipulating email on it. We focused our search to code written in Python, which has a built in library called `imaplib`. We also included searches for two particularly popular wrapper libraries for `imaplib` called `imappy` and `imapclient`. To exclude the many artifacts that used IMAP simply to send application notifications, we also limited the search to repositories that included the term "email" in their `README` documentation.

From an initial 524 scripts, we excluded 25 that were code-example skeletons that had not been filled in, resulting in 499 scripts. The first author went through each script using an open-coding approach as in the survey to identify the main functions performed. Eight codes emerged, with each script labeled with one or more codes. To validate our codes, a second author used the identified categories to independently re-label a random sample of 30 scripts, achieving an inter-rater reliability of 64.9% using Cohen's κ.

Results

In Table 2, we present the 8 major categories of functions and their frequency in our dataset. Some scripts performed two or more functions and so had multiple labels. While many of the scripts addressed needs that were found in the first probe, we also noticed some new categories of needs.

Processing, organizing, and archiving content. Over a fifth of the scripts triggered some sort of action (e.g., send auto-responses, mark read, move to a folder) based on information parsed from an email. The most popular example was extracting part of an email and then replying with or sending the processed content to another user. Almost equally popular were scripts that exported content from the email system. Some scripts attempted to download message contents, particularly attachments (e.g. receipts, contact info), to places such as their desktop or the cloud.

Many of these scripts echoed the need for content processing and aggregation in Probe 1 to automate repetitive tasks. Many scripts also extracted latent structure from emails, reflecting the desire for richer data models. Overall, we found complex processing needs were prevalent, comprising around 50% of scripts. This may be because people who process large amounts of information via email were compelled to write scripts to save themselves time and effort and thus were more represented in Probe 2 than in Probe 1.

Altering the default presentation of email clients. Echoing survey results on altering inbox presentation, a number of scripts focused on displaying email threads in a different way, mimicking applications like Slack or Facebook. We also found scripts with more minor tweaks to current email client presentations. For instance, there were scripts that displayed one's inbox not by email subject lines but rather by some metadata of the email content. Several scripts also aimed to personalize push notifications for new emails, similar to rules in Probe 1 regarding attention management. For example, one script interfaced with a Raspberry PI and displayed visual effects when an email arrived in the user's inbox.

Email as middleware. In a divergence from the survey, we found scripts that attempted to use the inbox as *middleware*, towards performing actions outside of a user's inbox. Given the accessible and ubiquitous nature of email, some scripts enable users to use email as a form of cloud storage. Others enable users to trigger defined actions via email, such as uploading pictures to an online library, or controlling software remotely through the use of keywords. This technique is relatively technical, so it is understandable that it didn't occur to our survey respondents as a possibility. Previous studies [38] have found similar uses, such as users using a folder to label emails that have to be printed.

Inbox cleaning and spam removal. We saw a number of scripts focused on inbox cleanup to remove unimportant or spam emails. Some scripts attempted to delete unimportant messages regularly so that users could focus their attention on important messages. For instance, we encountered scripts concerned with the "Inbox Zero" approach to email management. The goal of this approach is to achieve as close to zero messages in a user's inbox at any given time. These scripts offer similar functionality to the sweep rule feature in Outlook, which runs at regular intervals to delete emails based on a user's rules.

Some scripts were also focused on auto-removing spam emails. Unlike the survey respondents, programmers incorporated engineering techniques to automatically extract

spam-related attributes of emails at a fine-grained level. Some of these scripts used machine learning libraries to train spam filters. It was unclear based on the scripts alone whether the programmers were attempting to replace current spam filter functionality in common email clients, or whether they were simply trying to implement basic machine learning models. Spam was not mentioned as frequently in Probe 1, possibly because users were satisfied with their existing spam filters.

Email analytics and productivity tools. Finally, we observed instances of *the reflective conversation* [24] when it came to tasks within emails, such as scripts that collected analytics such as response rate or frequency of recipients. The scripts provided statistics and insights to the users about how well they accomplished and completed email actions. Popular statistics like response rate were used as a measure of productivity week-to-week. This category was not mentioned as often in Probe 1, again possibly due to an over-representation of users in Probe 2 who have more intense workloads within email, or due to a correlation between programming skills and an interest in visualizing data about or optimizing one's own activities (i.e., the "quantified self").

5 PROBE 3: FIELD DEPLOYMENT OF SIMPLE INBOX SCRIPTING

Our final probe involved having users author rules to address their needs in a real environment over time. Even for programmers, writing an email automation is a significant programming task, which will likely deter many of them from undertaking the work. Thus, to inspect how programmers might customize email handling if it were less work, we deployed an experimental email engine that aims to dramatically simplify the task of authoring email rules. In contrast to Probe 1, we can see the rules users would actually write, as well as revise, over the course of a week on real emails, as opposed to simply users' ideas for automation.

To facilitate this probe, we developed **YouPS**, a system that lets users write email processing rules in Python and executes them over their IMAP API mailboxes. YouPS provides a Python environment populated with objects representing the user's email and folders, and methods that can access and manipulate them; it implements those methods by interacting with the user's IMAP server. Table 3 lists some example methods of YouPS. Given the importance of context in our prior probes, we provided access to their inbox's status and interaction histories with specific contacts (internal context) and *modes* that let users indicate and leverage a variety of external contexts.

Design of YouPS. YouPS provides an editor for writing so-called *filter* rules for incoming emails. Each rule is triggered only when there is a new email at the user's inbox. Users can add multiple editor tabs, each corresponding to a separate email *mode*. Users can write different rules for each mode so that their inbox behaves differently depending on the current mode. User can set their active email mode using a dropdown menu on the YouPS web interface or by calling methods to change the mode programmatically. Users can view execution logs and rule output in a console window below the editor as seen in Figure 2.

Each YouPS method prints logs when it is executed, and users can also choose to print logs using Python's native `print` methods. Before actually running their rules over their inbox, users can test out their rules by telling YouPS to print out all logs from a user's rules but not actually execute the action. Users can also enable and disable different email rules. If an error occurs while processing a user's rules, YouPS

Method	Description
[get, set]_mode()	Managing email modes
get_history(contact)	Return interaction history with the given contact
get_[content, date, label, sender, subject, recipient]()	Return metadata of the email

Table 3: Examples of YouPS methods

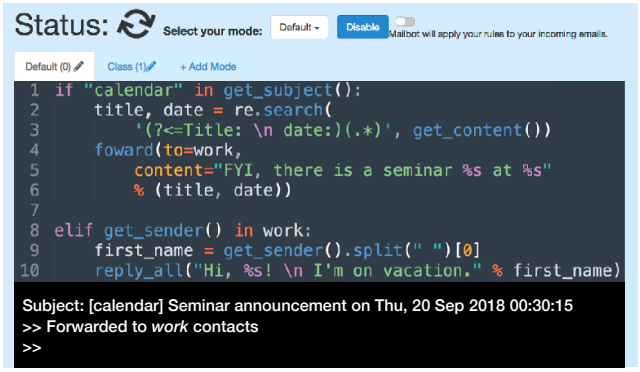

Figure 1: The YouPS interface. Users can program and debug their email rules with interactive editors and a console.

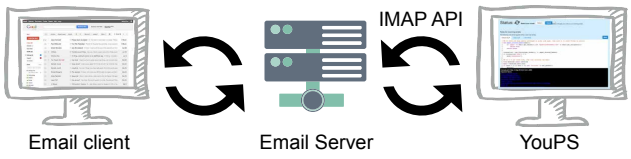

Figure 2: YouPS accesses each user's inbox through IMAP and generates a sandbox environment to execute their rule.

notifies users by sending them email and logging the error to the interface.

Deployment. We recruited 12 email users (3 females, 9 males, mean age=23.5) through a university lab mailing list and word-of-mouth. Participants were undergraduate or graduate students studying fields in engineering who could program in Python. We introduced participants to YouPS and had them link their email to the system and author scripts for their inbox. Over the course of the week, they could return to the interface to edit, debug, or create new scripts. After the deployment, we had a one-on-one interview with each participant. Participants were compensated $35 for their time.

Results

We present our analysis of rules made by participants as well as themes from the interviews, where users described their experience writing, editing, and testing email rules over time. Almost all the rules that participants wrote were comprised of chained conditionals. Participants wrote 4.8 `if` conditions on average per script. The conditions involved the subject line 60 times, the sender information 49 times, contents of the email 41 times, and interaction history 25 times. Actions performed on a condition included moving to a folder (40 instances), marking as read (22), labeling (6), programmatically switching a mode (6), and deleting a message (3).

Of the YouPS rules that our users created, we found that 40.4% could not be expressed using common email clients today, while the rest were common capabilities (e.g., filtering by keyword or sender). Below is an example of a script triggered by a sender and email body that performs the action of moving to a folder. It is possible to do this using email filter tools today.

```
auto_read_sender = ["LevelUp", "Twitter"..]
if get_sender() in auto_read_sender
    or auto_read_sender in get_content():
    mark_read(True)
```

Example 1: Mark as read a message depending on a sender and email body

However, other scripts had more complicated conditions. For instance, similar to our survey, we saw rules that performed actions only during certain time periods:

```
hour = datetime.now().hour
if hour in range(12,14)
    and "free food" in get_subject():
    delete()
```

Example 2: Remove irrelevant emails during a certain time range

Email modes. During the deployment, 24 unique modes were created by the participants. One user said that he didn't use standard filters in email clients because he couldn't activate them only for particular periods. In comparison, he found YouPS email modes useful for temporarily activating rules. Another user described the modes they created in YouPS as: "*I have a research mode (active from 9am-5pm) and sleep mode (active from 11pm to 8am). In my research mode, I mark all emails as read and move them to the TODO label. I do this so that I don't notice my new emails right away...and I can go through the new emails in my TODO folder periodically.*" One participant wrote a script to programmatically change their mode to "concentration mode" after the arrival of an important email:

```
urgentWords = ["important", "deadline"]
for w in urgentWords:
    if w in get_subject().lower()
        or w in get_content().lower():
        move("Important")
        set_mode("Concentration")
```

Example 3: Switch modes after receiving an email

Another user described how they could configure fine-grained filters using email modes: "*This [is] a way to write auto-replies that will be sent to co-workers, but not to family members, when going on vacation...Or, to snooze work-related email during the evening, but still allow family-related email...*"

Leveraging interaction history. Echoing our survey results, users incorporated interaction history to triage emails. Prior interactions implied that the email they were receiving from a person was likely to be important. One user said: "*If I've talked to someone via email, then their later messages are important to me*", while another said: "*You have different interaction periods with a person and those lead to different priorities.*". Conversely, users could automatically disregard messages from certain senders if there were too many messages within a short time period:

```
interaction = get_history(get_sender())
if interaction['received_emails'] > 5
    and get_sender() in mailing_list:
    mark_read(True)
```

Example 4: Disregard if too many emails within short time

(Non-)use of existing email client features. From interviews, we found that many users still manually process emails, even repetitive ones, despite the fact that current features within email clients can automate some of this activity. Similarly, during the deployment, more than half the rules written by participants could have been done already in their current client. We asked users why they didn't use

features in their current interface. Some participants said that it was difficult to write proper rules: "*I'd select the option to take emails from a certain address & marking them as read, but it didn't always work.*" This suggests that we need better ways to debug and test automations.

One surprise from YouPS was that many of our subjects (10 out of 12) *preferred* writing rules in Python to using their mail clients' rule-authoring interfaces. One said the reason they didn't like email interfaces for rule authoring is that they are cumbersome: "*Adding a new one often takes a lot of clicks and typing, even when there is an obvious pattern.*" In contrast, a second said "*(YouPS is) so much lower friction. I can copy-paste rules or even produce similar rules in a loop, and use familiar programming concepts to combine their logic or make exceptions.*"

Clearly, a programming language offers more flexibility than typical interfaces, but YouPS hides the complexity of talking to an IMAP server, so it may offer an attractive sweet-spot on the simplicity versus power tradeoff; this requires further investigation.

6 DISCUSSION

Our findings demonstrate that users would like more automation in their email management. The three distinct need-finding probes we carried out consistently identified certain common categories of email needs: capturing *richer data models* and internal and (time-varying) external *context*, using them for recipients to *manage attention* and for *senders* to reduce load on recipients, and *automated content processing* to, for example, aggregate replies to an invitation or extract attached photos into a relevant storage location.

Hacks that overload existing email features. As a way to manage attention, we noticed users coming up with hacks that repurpose existing email functionalities. For instance, several probes saw users marking emails unread as a way to remind themselves to revisit those emails. Several scripts from Probe 2 and 3 also tried to develop customized notifications by marking incoming emails as read and moving them to different folders to suppress the default notification.

This kind of overloading of existing features suggests that email systems could provide a richer data model for email users to manage attention. Is binary marking (e.g. marking as read/unread or starred) enough to imply different intensities of attention? Is labeling messages with priority effective given the eventual need for updates? One survey respondent said she often forgot to update her 'todo' labels, ending up with too many labels after a while. This suggests we need more dynamic or automated ways to mark emails, including ways to signal different and intersecting forms of priority.

Linking inboxes to other applications. We repeatedly encountered a desire to leverage internal and external context. Breaking with current email clients, this requires understanding users' inboxes beyond a single message and even further, beyond information stored within the inbox. Some of this context could be inferred rather than explicit, such as using number of unresponded emails or average response time to estimate a user's current load. However, context could also be determined explicitly, given the ability to link to other applications like calendars. As another example, some participants wanted to incorporate context such as current geolocation from their phone. This suggests that email systems should make it easier for users to leverage data from other applications when generating rules.

We also saw a need for complex content processing and interfacing with other applications, including use of email as middleware for other activities or to archive information to other places. Existing tools for task automation [26, 37] demonstrate one way to make it easier for email systems to interface with other systems, though some of these features may have greater adoption if integrated into email clients themselves.

Features from other social platforms. We found that email users wanted to incorporate features that are common on other social platforms. Indeed, prior work has pointed to ways that email usage, for instance within mailing lists, and social media usage, for instance on Facebook Groups, have become more similar [58]. This suggests additional ways that email systems could be adapted to suit how people conduct online discussion today. For instance, senders said they wanted to be more aware of their recipients, such as whether they were online or busy [41]. Users wanted more lightweight and casual ways of interacting as well as push notifications for certain content, blurring the lines between chat systems like Slack and email [59]. Users also wanted richer data models for their contacts in order to target messages, much like the user profiles that social media sites keep today. Finally, participants mentioned wanting community Q&A features like designating contacts to respond to a query, following or unfollowing threads, and the "average time to respond" metric available in tools like Piazza. They said seeing information about their own responsiveness would help increase productivity; we saw examples of this in Probe 2.

Customizing inbox presentation. Email has a broad spectrum of users from various backgrounds and contexts. A fixed interface for an email client cannot meet the diverse needs of email users. We noticed a desire to customize email interfaces in all three probes. This ranged from sorting emails in other than chronological order, highlighting important emails, and hiding some emails depending on time. Probes 1 and 2 also identified more structural changes to email clients,

permitting alternative ways of viewing threads or messages. Since changing the interface requires manipulating users' email client, we need to further investigate ways to allow email users to customize their interface.

7 LIMITATIONS

Two of our three studies explicitly focused on programmers due to the lack of existing non-programmer tools for automating email, while the other skewed towards engineers due to our sampled population. This raises questions of external validity. However, the needs we identified did not make reference to any programming-specific topics, and were similar between programmers and non-programmers.

So far, we have developed YouPS as a system for programmers. Its successful deployment validated our initial vocabulary of actions and data model drawn from our first probe, demonstrating that users' desired rules were easily and concisely described using this richer vocabulary. This offers the hope that new graphical interfaces (and machine learning tools) leveraging this richer vocabulary could permit even non-programmers to create their own email automations.

Because the questions in our survey were influenced by the design workshop, the survey responses may have been artificially steered to match the design workshop. At a minimum, however, the survey did reveal that many users shared the needs of those in the design workshop, even if we failed to identify some other needs. YouPS exposes a generic and obvious set of actions such as metadata access and email foldering; our subjects were not specifically constrained to address the same categories of needs as those found in the other studies, but they did so.

Although our studies revealed many needs that cannot be met by current clients, we cannot conclude what numerical proportion of needs are not being met. It is possible that users primarily focused on unmet needs, artificially amplifying that portion; alternatively, it is possible that familiarity with existing capabilities led users to focus more on needs that could already be met. But regardless of the proportion, we believe we have identified a number of interesting categories of needs that are not currently addressed.

Probe 2 identified scripts where programmers wrote code to address needs. One could therefore argue that these needs have been met by the extant code. However, of the 499 scripts, only 12 provided pre-packaged applications (e.g. browser extension or desktop application) that could be used in turnkey fashion by non-programmers. The remainder were either code libraries or scripts that other programmers would need to configure for their own environments by editing the source. It seems unlikely that exactly one person needed each script's solution, but the average number of non-author "watchers" of a project was 1.3, suggesting most of the projects are one-off and not widely used. Thus, it is still important to consider why more generally-accessible solutions do not exist, forcing programmers to write one-off scripts. YouPS aims to lower the bar for this.

8 FUTURE WORK

Our preliminary deployment of YouPS yielded evidence that our extended email vocabulary is suitable for scripting the rules users want. But we need a larger-scale deployment to gain confidence in this evidence. By deploying YouPS to more users over a larger duration, we hope to inspire participants to be more ambitious in their automations. Users' descriptions of things they would like to do with YouPS but cannot will reveal gaps in our vocabulary and ways to fill them. As the rules written in YouPS become large and complicated, we will look for common patterns in those rules capture them in new vocabulary that makes it possible to simplify those rules. We will also seek insight from prior tools for end-user programming [7, 31].

At the same time, we aim to extend the email automation capabilities of YouPS to non-programmers by creating GUIs for expressing rules that use the broader vocabulary. For initial guidance we can look to existing email clients' interfaces that permit non-programmers to create filters; these interfaces generally offer drop down menus of attributes and constraints that can be applied in filtering. We believe such an interface could be augmented with the new concepts identified in the work presented here. We will also consider alternatives, e.g., block programming but with support for reusing and remixing email rules [43], in a way that is usable for non-programmers.

9 CONCLUSION

We conducted a series of need-finding probes regarding email automation. First, we led a design workshop to identify categories of email needs, followed by a larger survey to deepen our understanding of the needs we identified. We performed a second probe by mining open source code on Github to see what needs have already been enacted by programmers. Finally, we conducted a deployment of a programmable email system YouPS which enables users to write custom email rules through a simple programmatic API wrapper of IMAP, as well as a study of users' interaction with the system over the course of a week. We found limitations in the implementation of current email clients that do not satisfy the complex desires of email users. The needs discovered in our studies will guide future designers and developers of email clients and inbox management systems.

10 ACKNOWLEDGMENTS

We thank our participants and reviewers, particularly our shepherd for their help improving the paper's structure. Soya Park is partly supported by the Kwanjeong fellowship.

REFERENCES

[1] Foundry 376. 2017. Mailspring. https://getmailspring.com.

[2] Gediminas Adomavicius and Alexander Tuzhilin. 2011. *Context-Aware Recommender Systems*. Springer US, Boston, MA, 217–253. https://doi.org/10.1007/978-0-387-85820-3_7

[3] Tarfah Alrashed, Ahmed Hassan Awadallah, and Susan Dumais. 2018. The Lifetime of Email Messages: A Large-Scale Analysis of Email Revisitation. In *Proceedings of the 2018 Conference on Human Information Interaction&Retrieval (CHIIR '18)*. ACM, New York, NY, USA, 120–129. https://doi.org/10.1145/3176349.3176398

[4] Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, and Ian Smith. 2003. Taking email to task: the design and evaluation of a task management centered email tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '03)*. ACM, New York, NY, USA, 345–352. https://doi.org/10.1145/642611.642672

[5] Frank Bentley, Nediyana Daskalova, and Nazanin Andalibi. 2017. If a person is emailing you, it just doesn't make sense: Exploring Changing Consumer Behaviors in Email. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 85–95. https://doi.org/10.1145/3025453.3025613

[6] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories (MSR '06)*. ACM, New York, NY, USA, 137–143. https://doi.org/10.1145/1137983.1138016

[7] Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. 2005. Automation and customization of rendered web pages. In *Proceedings of the 13th international conference on Intelligent user interfaces (IUI '05)*. ACM, New York, NY, USA, 163–172. http://doi.acm.org/10.1145/1095034.1095062

[8] boomerang. 2016. boomerang. http://www.boomeranggmail.com.

[9] Nathanial S. Borenstein. 1992. Computational mail as network infrastructure for computer-supported cooperative work. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work. (CSCW '92)*. ACM, New York, NY, USA, 67–74. https://doi.org/10.1145/143457.143463

[10] Horatiu Bota, Paul N. Bennett, Ahmed Hassan Awadallah, and Susan T. Dumais. 2017. Self-Es: the role of emails-to-self in personal information management. In *Proceedings of the 2017 Conference on Human Information Interaction&Retrieval (CHIIR '17)*. ACM, New York, NY, USA, 205–214. https://doi.org/10.1145/3020165.3020189

[11] Mary Czerwinski, Eric Horvitz, and Susan Wilhite. 2004. A diary study of task switching and interruptions. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '04)*. ACM, New York, NY, USA, 175–182. http://doi.acm.org/10.1145/985692.985715

[12] Laura A. Dabbish, Robert E. Kraut, Susan Fussell, and Sara Kiesler. 2005. Understanding email use: predicting action on a message. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*. ACM, New York, NY, USA, 691–700. http://doi.acm.org/10.1145/1054972.1055068

[13] Senad Dizdar. 2011. CloudHQ. https://linux.die.net/man/1/procmail.

[14] Marian Dork, Daniel Gruen, Carey Williamson, and Sheelagh Carpendale. 2010. A visual backchannel for large-scale events. In *IEEE transactions on visualization and computer graphics (Vis '10)*, Vol. 16. IEEE, 1129–1138.

[15] Mark Dredze, Tessa Lau, and Nicholas Kushmerick. 2006. Automatically classifying emails into activities. In *Proceedings of the 11th international conference on Intelligent user interfaces (IUI '06)*. ACM, New York, NY, USA, 70–77. http://doi.acm.org/10.1145/1111449.1111471

[16] Mark Dredze, Bill N. Schilit, and Peter Norvig. 2009. Suggesting Email View Filters for Triage and Search. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI '09)*. AAAI, Palo Alto, CA, USA, 1414–1419. https://www.aaai.org/ocs/index.php/IJCAI/IJCAI-09/paper/viewFile/488/909

[17] Mark Dredze, Hanna M. Wallach, Danny Puller, and Fernando Pereira. 2008. Automatically classifying emails into activities. In *Proceedings of the 13th international conference on Intelligent user interfaces (IUI '08)*. ACM, New York, NY, USA, 199–206. http://doi.acm.org/10.1145/1378773.1378800

[18] Mozilla Foundation. 2004. Mozilla Thunderbird. https://www.thunderbird.net/.

[19] Google. 2014. Google Inbox. https://inbox.google.com/.

[20] Sukeshini A. Grandhi and Lyndsey K. Lanagan-Leitzel. 2016. To Reply or To Reply All: Understanding Replying Behavior in Group Email Communication. In *Proceedings of the 2016 ACM conference on Computer-supported cooperative work. (CSCW '16)*. ACM, New York, NY, USA, 560–569. https://doi.org/10.1145/2818048.2819981

[21] Catherine Grevet, David Choi, Debra Kumar, and Eric Gilbert. 2014. Overload is overloaded: email in the age of Gmail. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. 793–802. https://doi.org/10.1145/2556288.2557013

[22] Daniel Gruen, Steven L. Rohall, Suzanne Minassian, Bernard Kerr, Paul Moody, Bob Stachel, Martin Wattenberg, and Eric Wilcox. 2004. Lessons from the reMail prototypes. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work (CSCW '04)*. ACM, New York, NY, USA, 152–161. http://doi.acm.org/10.1145/1031607.1031634

[23] Philip A. Guenther. 1990. procmail. https://linux.die.net/man/1/procmail.

[24] William C. Hill, James D. Hollan, Dave Wroblewski, and Tim McCandless. 1992. Edit wear and read wear. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*. ACM, New York, NY, USA, 3–9. https://doi.org/10.1145/142750.142751

[25] Edward Hung. 2001. Deduction of Procmail Recipes from Classified Emails. *CMSC724 Database Management Systems, individual research project report* (2001).

[26] IFTTT. 2010. IFTTT Gmail. https://ifttt.com/gmail.

[27] GrexIt Inc. 2011. hiver. https://hiverhq.com.

[28] Anjuli Kannan, Karol Kurach, Sujith Ravi, Tobias Kaufmann, Andrew Tomkins, Balint Miklos, Greg Corrado, Laszlo Lukacs, Marina Ganea, Peter Young, et al. 2016. Smart reply: Automated response suggestion for email. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 955–964.

[29] Shih-Wen Ke, Chris Bowerman, and Michael Oakes. 2006. Perc: A personal email classifier. *European Conference on Information Retrieval* (2006), 460–463. https://doi.org/10.1007/11735106_41

[30] Nicolas Kokkalis, Johannes Roith Chengdiao Fan, Michael S. Bernstein, and Scott Klemmer. 2017. MyriadHub: Efficiently Scaling Personalized Email Conversations with Valet Crowdsourcing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 73–84. https://doi.org/10.1145/3025453.3025954

[31] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 1719–1728. http://doi.acm.org/10.1145/1357054.1357323

[32] Wendy E Mackay. 1998. More than just a communication system: diversity in the use of electronic mail. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work (CSCW '98)*. ACM, New York, NY, USA, 344–353. http://doi.acm.org/10.1145/62266.62293

[33] Kaitlin Mahar, Amy X Zhang, and David Karger. 2018. Squadbox: A Tool to Combat Email Harassment Using Friendsourced Moderation. In

Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. ACM, 586–599.

[34] Mailbird. 2013. Mailbird. https://www.getmailbird.com.

[35] Gloria Mark, Shamsi T. Iqbal, Mary Czerwinski, Paul Johns, Akane Sano, and Yuliya Lutchyn. 2016. Email duration, batching and self-interruption: Patterns of email use on productivity and stress. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 1717–1728. https://doi.org/10.1145/2858036.2858262

[36] Luke McDowell, Oren Etzioni, Alon Halevy, and Henry Levy. 2004. Semantic email. In *Proceedings of the 13th international conference on World Wide Web (WWW '04)*. ACM, New York, NY, USA, 244–254. https://doi.org/10.1145/988672.988706

[37] Microsoft. 2018. Microsoft Flow. https://flow.microsoft.com/en-us/.

[38] Michael Muller and Daniel Gruen. 2002. Collaborating within not through email: Users reinvent a familiar technology.

[39] Michael J. Muller and Daniel M. Gruen. 2005. Working together inside an emailbox. In *Proceedings ECSCW 2005 (ECSCW '05)*. Springer, Dordrecht, 103–122. https://doi.org/10.1007/1-4020-4023-7_6

[40] Kanika Narang, Susan T. Dumais, Nick Craswell, Dan Liebling, and Qingyao Ai. 2017. Large-scale analysis of email search and organizational strategies. In *Proceedings of the 2017 Conference on Human Information Interaction&Retrieval (CHIIR '17)*. ACM, New York, NY, USA, 215–223. https://doi.org/10.1145/3020165.3020175

[41] Bonnie A. Nardi, Steve Whittaker, and Erin Bradner. 2000. Interaction and Outeraction: Instant Messaging in Action. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work (CSCW '00)*. ACM, New York, NY, USA, 79–88. http://doi.acm.org/10.1145/358916.358975

[42] Carman Neustaedter, A J. Bernheim Brush, Marc Smith, and Danyel Fisher. 2005. The Social Network and Relationship Finder: Social Sorting for Email Triage. *Proceedings of the 2005 Conference on Email and Anti-Spam (CEAS)*.

[43] Stephen Oney, Brad Myers, and Joel Brandt. 2014. InterState: a language and environment for expressing interface behavior. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 263–272. https://doi.org/10.1145/2642918.2647358

[44] CMU Cyrus Project. 2008. sieve. http://sieve.info.

[45] Steven L. Rohall, Daniel Gruen, Paul Moody, and Seymour Kellerman. 2001. Email visualizations to aid communications. In *Proceedings of InfoVis 2001 The IEEE Symposium on Information Visualization (InfoVis '01)*. IEEE, 15.

[46] Abigail J Sellen, Rachel Murphy, and Kate L Shaw. 2002. How knowledge workers use the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing System (CHI '02)*. ACM, New York, NY, USA, 227–234. http://doi.acm.org/10.1145/503376.503418

[47] Nikash Singh, Martin Tomitsch, and Mary Lou Maher. 2006. Understanding the management and need for awareness of temporal information in email. In *Proceedings of the 2006 international workshop on Mining software repositories (MSR '06)*. ACM, New York, NY, USA, 137–143. https://doi.org/10.1145/1137983.1138016

[48] Anselm Strauss and Juliet M Corbin. 1990. *Basics of qualitative research: Grounded theory procedures and techniques*. Sage Publications, Inc.

[49] Saiganesh Swaminathan, Raymond Fok, Fanglin Chen, Ting-Hao Kenneth Huang, Irene Lin, Rohan Jadvani, Walter S. Lasecki, and Jeffrey P. Bigham. 2017. WearMail: On-the-Go Access to Information in Your Email with a Privacy-Preserving Human Computation Workflow. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17)*. ACM, New York, NY, USA, 807–815. https://doi.org/10.1145/3126594.3126603

[50] Agnieszka Matysiak Szóstek. 2011. 'Dealing with My Emails': Latent user needs in email management. *Computers in Human Behavior* 27, 2 (2011), 723–729. https://doi.org/10.1016/j.chb.2010.09.019

[51] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger. 2004. The perfect search engine is not enough: a study of orienteering behavior in directed search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 415–422. https://doi.org/10.1145/985692.985745

[52] Joshua R Tyler and John C Tang. 2003. When can I expect an email response? A study of rhythms in email usage. In *ECSCW 2003*. Springer, 239–258.

[53] Stephen Voida, Elizabeth D. Mynatt, Blair MacIntyre, and Gregory M. Corso. 2002. Integrating virtual and physical context to support knowledge workers. *IEEE Pervasive Computing* 1 (2002), 73–79.

[54] Martin Wattenberg, Steven L. Rohall, Daniel Gruen, and Bernard Kerr. 2005. E-mail research: Targeting the enterprise. *Human-Computer Interaction* 20, 2 (2005), 139–162. https://doi.org/10.1207/s15327051hci2001&2_5

[55] Steve Whittaker, Victoria Bellotti, and Jacek Gwizdka. 2006. Email in personal information management. *Commun. ACM* 49, 1 (2006), 68–73. https://doi.org/10.1145/1107458.1107494

[56] Steve Whittaker and Candace Sidner. 1996. Email overload: exploring personal information management of email. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 793–802. https://doi.org/10.1145/238386.238530

[57] Shinjae Yoo, Yiming Yang, and Jaime Carbonell. 2011. Modeling personalized email prioritization: classification-based and regression-based approaches. In *Proceedings of the 20th ACM international conference on Information and knowledge management (CIKM '11)*. ACM, New York, NY, USA, 729–738. https://doi.org/10.1145/2063576.2063683

[58] Amy X. Zhang, Mark S. Ackerman, and David R. Karger. 2015. Mailing Lists: Why Are They Still Here, What's Wrong With Them, and How Can We Fix Them?. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 4009–4018. https://doi.org/10.1145/2702123.2702194

[59] Amy X. Zhang and Justin Cranshaw. 2018. Making Sense of Group Chat Through Collaborative Tagging and Summarization. In *Proceedings of the 2018 ACM conference on Computer supported cooperative work (CSCW '18)*. ACM, New York, NY, USA, 196–223. http://doi.acm.org/10.1145/3274465

[60] Qian Zhao, Paul Bennett, Adam Fourney, Anne Loomis Thompson, Shane Williams, Adam D. Troy, and Susan Dumais. 2018. Calendar-Aware Proactive Email Recommendation. In *Proceedings of the 41st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '18)*. ACM, New York, NY, USA, 655–664. http://doi.acm.org/10.1145/3209978.3210001