RTFM (Read The Factual Mails) — Augmenting Program Comprehension with Remail

Alberto Bacchelli, Michele Lanza, Vitezslav Humpa
REVEAL @ Faculty of Informatics - University of Lugano, Switzerland

***Abstract*—The advent of globalization has led to the adoption of distributed software development as a common practice. One of its drawbacks -the absence of impromptu meetings- is tackled with other communication means, such as emails, instant messaging, or forums. Mailing lists have proven to be effective for enabling developers' collaboration and coordination: Being asynchronous, emails can evade timezone barriers; being public, mailing lists maintain developers' awareness; being recorded, email archives offer information on system evolution.**

Emails can provide information about a task, clarify implementation details, or reveal hidden connections among entities, always within the clear context of a discussion. As a result, we argue that emails might help program comprehension.

We devised *Remail*, an Eclipse plugin to integrate email communication in the IDE. It allows developers to seamlessly handle code entities and emails concerning the source code. Discussions relevant to chosen entities can be retrieved easily, thus providing an updated and effective form of complementary documentation. We present design and implementation of Remail, and illustrate, through a number of scenarios, how it can augment program comprehension.

I. Introduction

Developers spend most of their programming time on software maintenance and program comprehension. Software maintenance is estimated to impact between 85% and 90% of the global cost of a software system [6], [21]; while program comprehension takes up 50-60% of this maintenance time [5].

Clear, comprehensive, and updated software documentation would be an effective approach to reduce time spent in program comprehension. Nevertheless, industrial developers report how documentation is commonly inadequate, outdated, and hard to retrieve or link to actual source code entities [14]. Open source development projects are similarly affected by issues related to documentation [8].

In small co-located development teams, unplanned face-to-face meetings are the favorite form of communication when developers face program comprehension problems [14]. Developers who need to understand source code entities (*e.g.,* to know the design rationale behind a certain implementation –the most common information need for a developer [12]), and cannot find the appropriate documentation, simply query other programmers. This solution, besides disrupting developers' attention and retaining knowledge by a few developers, is inapplicable to large or distributed development projects.

Developers, thus, replace face-to-face meetings with electronic communication means. Instant messaging, wikis, forums are viable options, but the decisive role is played by emails, indeed: "Mailing lists are the bread and butter of project communications" [7]. Emails are asynchronous, thus evade time zone barriers and do not disrupt developers' attention; mailing lists broadcast discussions, announcement, and decisions to all the participants, thus maintaining developers' awareness; emails are not bound to specific abstraction levels (as opposed to commit messages, design documents, or code comments), thus they can be used to discuss issues ranging from low-level decisions (e.g., implementation, bug fixing) up to high-level considerations (e.g., design rationales); mailing lists archive messages, thus offering a historical perspective.

Development mailing lists are places in which developers discuss and share knowledge. Gleixner, main maintainer of the Real-Time Linux Kernel, explains how "the Linux kernel mailing list archives provide a huge choice of technical discussions" and reveals that developers are "too lazy or too busy" to write documentation, since they "believe that it is all documented in [. . .] the maze of the Linux kernel mailing list archives which are freely available for the interested reader." [8]

Considering the breadth of the information that is to be found in mailing lists (*e.g.,* information about how to perform a specific development task, clarification on certain implementation details, explanation of high-level design decisions), and that readers are able to always verify the *context* of ongoing discussions and decide whether the retrieved information applies to their situation, we argue that emails can be used to help program comprehension tasks.

We present Remail, a plugin for Eclipse, which integrates email archives in the IDE. Remail is a recommendation system for emails: It allows developers to easily retrieve discussions related to the chosen code entities. The practitioner can read and learn from previous discussions occurred among programmers, thus accessing an updated and effective form of complementary documentation. Using Remail the interaction with the emails is within the development environment, *i.e.,* the programmer is not forced to frequent context switches and can retain the current development situation.

Structure of the paper: In Section II we present Remail, our recommender for emails and, in Section III, we illustrate how it can augment program comprehension. In Section V we present the related work, and we conclude in Section VI.

II. REmail: Recommending Emails

When introducing the concept of open source projects, Fogel invites to use mailing lists "as much as possible, and as conspicuously as possible", since "searching in them [for answers to technical questions] can produce answers" [7]. Nevertheless, two critical issues hinder both a conspicuous usage and the effectiveness of mailing lists in supporting program comprehension and software development:

1) Mailing lists store very large amounts of messages: The archive of Linux counts more than one million messages, the mailing lists of smaller but active projects count tens of thousands of emails. Finding, in such archives, the most relevant information concerning a specific code entity is a non-trivial task, and important discussions could be missed [10].
2) Development mailing lists discuss topics related to project development and are continuously read and written by developers. Developers spend most of their programming, designing, and understanding time within IDEs [14]. However, no matter how much related emails are to software development, programs that are *external* to IDEs manage them, sometimes even in a web browser. Therefore emails are completely *disconnected from the development environment*.

We devised Remail to tackle both issues. Remail recommends the emails that are related to specified code artifacts, by using the lightweight techniques we devised and thoroughly evaluated for this task [1], [3]. This reduces the amount of messages to be read by orders of magnitude, and lets practitioners focus on the emails related to their tasks. In addition, Remail is a modular plugin for the Eclipse IDE, thus among other benefits, it allows developers to (i) simultaneously inspect code and content of messages, easily (ii) prompt recovered traceability links between code and emails, and (iii) minimize the disruptive context switches necessary to access email data while programming.

Eclipse has been our target IDE because of its modular and pluggable structure, its significant amount of users (who can benefit from Remail, and provide feedback for further improvement), and its support for multiple languages. The current implementation of Remail is targeted to java systems, however our lightweight linking techniques have proven to be effective for a number of other languages (*e.g.,* ECMAScript, PHP) [3]; the multi-language support of Eclipse paves the way to expand Remail to other languages with a minor effort.

Since Remail can be considered a recommendation system, we detail it by following the division described by Robillard *et al.* [19]: A recommender system involves: a *data-collection mechanism* to store development-process data and artifacts in a model, a *recommendation engine* to analyze the data and generate recommendation, and a *user interface* to trigger recommendations and present the results.

A. Data-collection Mechanism

Figure 1 shows the infrastructure we devised for handling the data in Remail. The section *Available email archives* reports the two formats from which Remail collects data.

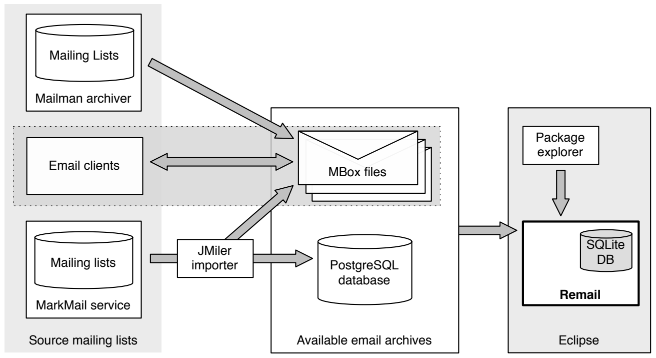


Figure 1. Data-handling infrastructure

PostgreSQL database: This solution stores emails into a running instance of a PostgreSQL server, according to a specific meta-model we devised (the interested reader can refer to Remail's implementation[1] for further details). Since Remail's recommendation engine mainly uses regular expression for finding relevant emails, the database solution offers the best performances: PostgreSQL provides a highly optimized and effective platform for regular expression based queries. In addition, the database solution allows multiple connections, thus enabling multiple developers to work simultaneously using a single email archive. This can be an advantage when rating email relevance: Whenever developers retrieve an irrelevant email (traceability links cannot be always perfect), they can rate it as *non-interesting*, and all the team would benefit from its removal.

MBox files: This solution provides messages in plain text files, according to the RFC 2822[2] Internet Message format. The MBox solution is lightweight in terms of requirements and can take advantage of established email clients. While the database approach needs a running instance of PostgreSQL server, MBox files can be directly accessed on the filesystem. In addition, a number of popular email clients (*e.g.,* Mozilla Thunderbird and related clients, Apple Mail, Eudora) use the MBox format to store the messages on which they operate. This is a decisive benefit offered by this approach: Practitioners are able to immediately take advantage of the emails already archived by their email clients, just by pointing Remail to the right folders. From that moment, simply and transparently, new messages in the email client will be also available in Remail, modified or deleted emails in the client will be also accordingly updated in Remail.

New developers, analysts, or researchers can be interested in accessing archives of mailing lists to which they are not

[1]http://code.google.com/p/r-email/
[2]http://tools.ietf.org/html/rfc2822

yet subscribed. The first section of Figure 1, called *Source mailing lists*, shows the three different sources for emails that can be used for this purpose. In the top, we find the mailing lists managed by GNU Mailman[3], which uses the MBox format. Most open source projects use Mailman as their list management system and also publicly offer the email archives. As they are already in MBox format, they can be immediately used by Remail.

In the bottom part of the first section, we find mailing lists managed by the MarkMail[4], a free online service for searching within constantly updated mailing list archives. It stores and consistently displays more than 8,000 mailing lists, mainly taken from open source software projects. We implemented *JMiler*, an importer that crawls the MarkMail website, extracts all the emails from the selected mailing lists, and populates PostgreSQL databases or MBox files.

Finally, the aforementioned solution of using a pre-existing email client is also presented in the first section of Figure 1. We note how, by using the MBox files approach, the messages imported via Mailman, or using the JMiler importer, can be used not only by Remail, but also transparently by the email clients, which will be able to directly operate on them.

B. Recommendation Engine

The recommendation engine analyzes the data offered by the *Available email archives* layer and generates the recommendation for the provided context. Users specify the context directly from the Eclipse environment: They select the packages and classes in which they are interested and trigger the engine. The recommendation consists in presenting emails that discuss the chosen classes, as they might be useful for program comprehension and augmenting awareness.

Retrieving the most appropriate emails, given a class, is a non-trivial task. We previously assessed that this task can be tackled with lightweight text matching techniques [1], which achieve notable results also compared to more sophisticated information retrieval methods [3]. In our previous work, we devised six techniques that can provide all the links, from a chosen entity to tens of thousands of emails, in a few seconds. These are at the basis of our current Remail recommendation engine, the interested reader can refer to our previous work for further details on the evaluation and the rationale of our techniques, and can refer to the Remail source code for the actual implementation.

Since our lightweight methods offer different trade-offs between precision and recall, users could find some of them to be more appropriate for their needs. For this reason, we included all the methods in the Remail engine and let the user configure the preferred option.

C. User Interface

Figure 2 presents the Remail plugin as seen during a development session in the Eclipse IDE.

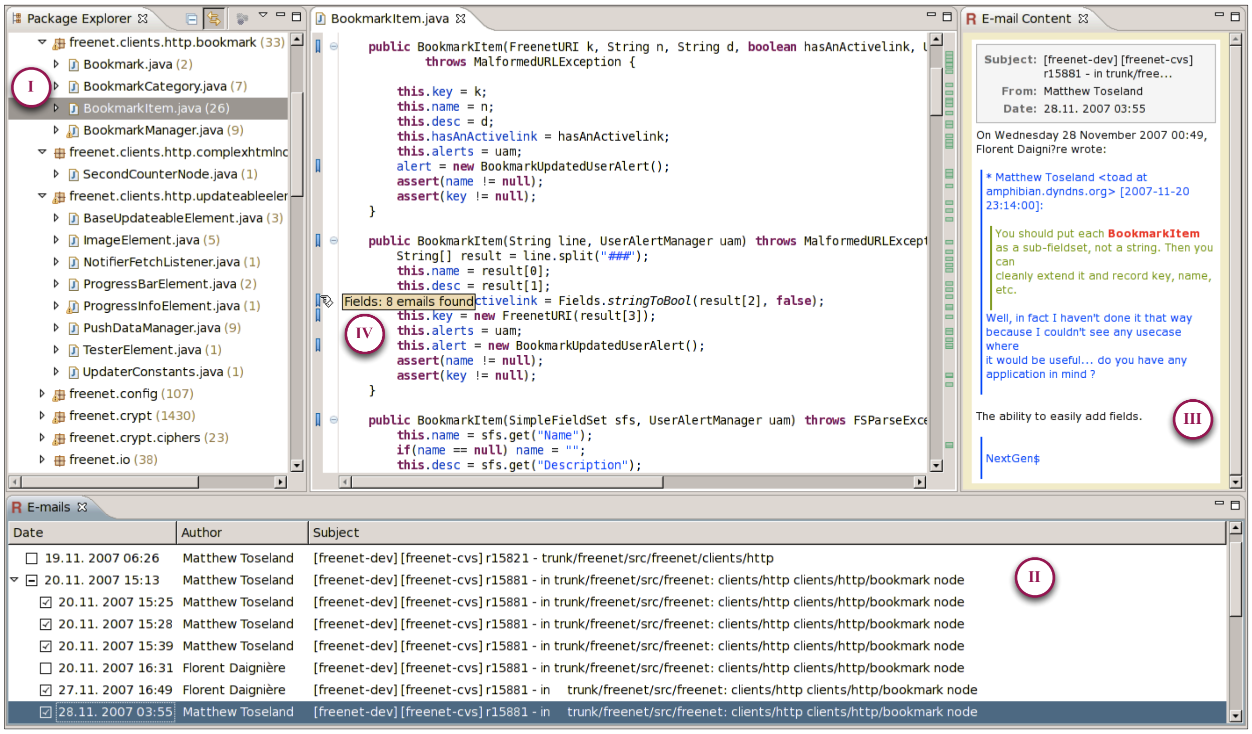
[3]The GNU Mailing List Manager, http://www.gnu.org/software/mailman
[4]http://www.markmail.org

Package Explorer: This is the entry point for interacting with Remail and triggering its recommendation engine (Figure 2, Point I). The first time Remail is used, the developer selects the classes and packages of interest and starts the search by clicking *Remail search* in the popup menu. By selecting a package, the search will be recursively performed in all the subclasses. Once the process is completed, next to each chosen entity in the Package Explorer, the user sees a number, which shows how many “hits” each entity has within the mailing list, thus effectively measuring the *popularity* of the entities [2]. As we discuss in Section III, the popularity can be used as an entry point to study an unknown system.

Emails View: Once a search has been performed, the user can click on any indexed class in the package explorer and the *Emails View* (Figure 2, Point II) will be updated accordingly. The visualization used in this panel conveys multiple details: (1) The messages are sorted by time, (2) the three columns (namely *date*, *author*, and *subject*) chunk the main metadata, and (3) the nested tree layout preserves the discussion *threads*. This view is similar to the one presented in common email clients: It scales to a vast number of emails and gives a temporal dimension. The latter feature is interesting from a program comprehension point of view: Even though the source code is updated, it is possible to walk back in time, by reading the emails discussing a class in the past. In mailing lists emails are organized in threads: Whenever there is a reply on an email, subsequent emails are handled in discussion threads by email clients. Remail supports threads in the *Available email archives* layer, in the *Emails View*, and in the *Email Content View*. This enhances email readability and the discussion context is more explicit.

Email Content View: Point III in Figure 2 details the panel that is opened when a specific email is selected in the *Emails View*. Its content is also visual: A box presents the metadata, different colors and bars distinguish the different quotation levels, and a bold red typeface highlights the name of the class for which the email was recommended.

Editor support: Most time that developers spend using IDEs is focused on the editor, where they actually work with the source code. They might want to maximize the editor to completely fill the screen. By doing so, the views of Remail are not visible. Therefore, we enhance the Editor itself to provide support in this situation. *Markers* signal general point of interest in any of the resource files. We have used the bookmark markers to provide information about all the class names visible in the source code editor, to which some emails have been linked. A toolbar button triggers markers, so that user can decide whether to show them. In Figure 2 Point IV, the pointer hovers a marker in a line of code that uses the class *Fields*, and Remail informs us about the existence of eight emails concerning this entity.



freenet.clients.http.bookmark (33)
Bookmark.java (2)
BookmarkCategory.java (7)
BookmarkItem.java (26)
BookmarkManager.java (9)
freenet.clients.http.complexhtmlnc
SecondCounterNode.java (1)
freenet.clients.http.updateableeler
BaseUpdateableElement.java (3)
ImageElement.java (5)
NotifierFetchListener.java (1)
ProgressBarElement.java (2)
ProgressInfoElement.java (1)
PushDataManager.java (9)
TesterElement.java (1)
UpdaterConstants.java (1)
freenet.config (107)
freenet.crypt (1430)
freenet.crypt.ciphers (23)
freenet.io (38)

```
public BookmarkItem(FreenetURI k, String n, String d, boolean hasAnActivelink, U
        throws MalformedURLException {

    this.key = k;
    this.name = n;
    this.desc = d;
    this.hasAnActivelink = hasAnActivelink;
    this.alerts = uam;
    alert = new BookmarkUpdatedUserAlert();
    assert(name != null);
    assert(key != null);
}

public BookmarkItem(String line, UserAlertManager uam) throws MalformedURLExcept
    String[] result = line.split("###");
    this.name = result[0];
    this.desc = result[1];
Fields: 8 emails foundctivelink = Fields.stringToBool(result[2], false);
    this.key = new FreenetURI(result[3]);
    this.alerts = uam;
    this.alert = new BookmarkUpdatedUserAlert();
    assert(name != null);
    assert(key != null);
}

public BookmarkItem(SimpleFieldSet sfs, UserAlertManager uam) throws FSParseExce
    this.name = sfs.get("Name");
    if(name == null) name = "";
    this.desc = sfs.get("Description");
```

Subject: [freenet-dev] [freenet-cvs] r15881 - in trunk/free...
From: Matthew Toseland
Date: 28.11. 2007 03:55

On Wednesday 28 November 2007 00:49, Florent Daigni?re wrote:

> * Matthew Toseland <toad at amphibian.dyndns.org> [2007-11-20 23:14:00]:
>
>> You should put each **BookmarkItem** as a sub-fieldset, not a string. Then you can
>> cleanly extend it and record key, name, etc.
>
> Well, in fact I haven't done it that way because I couldn't see any usecase where
> it would be useful... do you have any application in mind ?

The ability to easily add fields.

> NextGen$

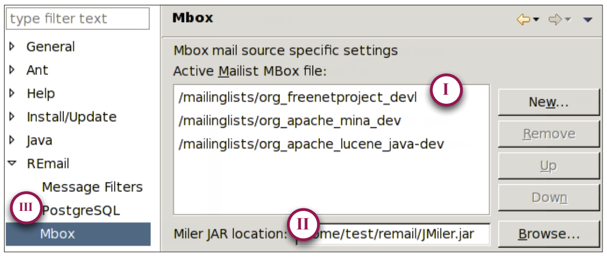
Date	Author	Subject
☐ 19.11. 2007 06:26	Matthew Toseland	[freenet-dev] [freenet-cvs] r15821 - trunk/freenet/src/freenet/clients/http
⊟ 20.11. 2007 15:13	Matthew Toseland	[freenet-dev] [freenet-cvs] r15881 - in trunk/freenet/src/freenet: clients/http clients/http/bookmark node
☑ 20.11. 2007 15:25	Matthew Toseland	[freenet-dev] [freenet-cvs] r15881 - in trunk/freenet/src/freenet: clients/http clients/http/bookmark node
☑ 20.11. 2007 15:28	Matthew Toseland	[freenet-dev] [freenet-cvs] r15881 - in trunk/freenet/src/freenet: clients/http clients/http/bookmark node
☑ 20.11. 2007 15:39	Matthew Toseland	[freenet-dev] [freenet-cvs] r15881 - in trunk/freenet/src/freenet: clients/http clients/http/bookmark node
☐ 20.11. 2007 16:31	Florent Daignière	[freenet-dev] [freenet-cvs] r15881 - in trunk/freenet/src/freenet: clients/http clients/http/bookmark node
☑ 27.11. 2007 16:49	Florent Daignière	[freenet-dev] [freenet-cvs] r15881 - in trunk/freenet/src/freenet: clients/http clients/http/bookmark node
☑ 28.11. 2007 03:55	Matthew Toseland	[freenet-dev] [freenet-cvs] r15881 - in trunk/freenet/src/freenet: clients/http clients/http/bookmark node

Figure 2. The Remail Plugin

D. *Implementation details*

Having a working implementation of Remail allowed us not only to run the case study we present in Section III, but also to face difficulties posed by a real-world context, such as performance and usability issues.

Performances and indexing: When using MBox files as data source, performance must be taken into account. Remail relies on fast and lightweight techniques, which find relationships between a source code entity and thousands of emails in seconds. As a consequence, it is feasible to trigger a new searching process on the MBox files every time users want to inspect emails related to a class. The advantage is that links are always updated, even when archive files are externally modified (*e.g.,* by email clients). We found this approach to be inappropriate when dealing with more than one class (*e.g.,* with a package) since the linking time grows linearly with the number of entities to be linked. For this reason, Remail requires a mechanism to store links and additional information. Since we want the MBox files to be correctly used by other applications, they cannot be modified to store extraneous information. For this reason, we rely on an internal lightweight database (see Figure 1): The first time Remail is used on a class, the results is stored in this database which, from that moment, is used to present results instantaneously. The user can trigger the cache update whenever necessary. Finally, since the first linking of a complete project can take minutes, we give feedback to the user about the process with a progress bar. The PostgreSQL solution already saves the links in a specific table, thus does not requires the internal Remail cache.



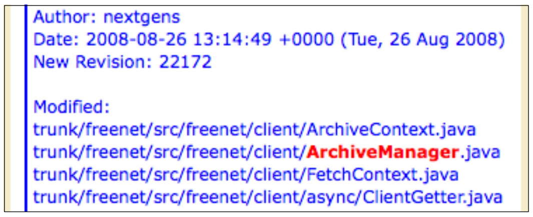
Mbox

Mbox mail source specific settings
Active Mailist MBox file:
/mailinglists/org_freenetproject_devl
/mailinglists/org_apache_mina_dev
/mailinglists/org_apache_lucene_java-dev

Miler JAR location: me/test/remail/JMiler.jar

Figure 3. A data source configuration phase

Source configuration: Remail offers an interactive configuration manager. It allows users to configure the preferred code-to-email linking technique, the email archives to be used as sources (either a PostgreSQL database or MBox files), and global message filtering. Point I in Figure 3 details the procedure for specifying MBox files. Users can indicate the files already operated by their email clients, independent MBox files, and empty files. For the last type of files, also the location of JMiler is to be provided (Point II): Given a correct syntax (*e.g.,* org_freenetproject_devl) and an existing mailing list, JMiler crawls MarkMail and populates the file with the retrieved data. By clicking on *PostgreSQL* (Point III), a similar interface is shown to describe the configuration for a database source.

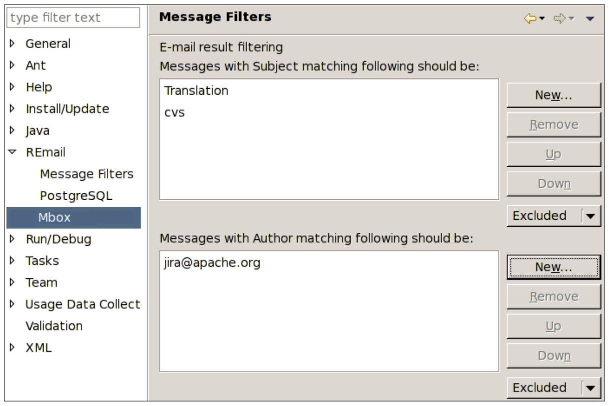


```
Author: nextgens
Date: 2008-08-26 13:14:49 +0000 (Tue, 26 Aug 2008)
New Revision: 22172

Modified:
trunk/freenet/src/freenet/client/ArchiveContext.java
trunk/freenet/src/freenet/client/ArchiveManager.java
trunk/freenet/src/freenet/client/FetchContext.java
trunk/freenet/src/freenet/client/async/ClientGetter.java
```

Figure 4. Excerpt of a related, however irrelevant, linked email

Global email filtering: During our case studies we obtained a considerable number of emails, that indeed referred to a class in question, but were irrelevant in a program comprehension context. The vast majority of such irrelevant messages are automatically generated and sent by issue repository systems, or by version control systems for detailing commits (*e.g.,* see Figure 4). These emails include listings of all classes that are part of each commit or defect report, which is hardly relevant for their understanding.



Message Filters

E-mail result filtering

Messages with Subject matching following should be:

Translation
cvs

Excluded

Messages with Author matching following should be:

jira@apache.org

Excluded

Figure 5. Email filtering configuration

Remail includes a message filtering feature to reject messages based on subject and author fields. The filtering configuration panel is shown in Figure 5. This feature proved to be helpful during our case study: The emails posted to the mailing lists by the version control systems have a special subject and all the emails posted by the issue repository system have the same sender, thus, by creating filters, we had been able remove all unnecessary emails. The filtered emails are removed from the linking results and do not appear in *Email View*, nor they are counted as "hits". The filtered links are not removed from the caching mechanism, thus we can try different filters without triggering the search again.

Email rating: In Point II of Figure 2 a checkbox close to each email allows users to give a binary rating to emails. Whenever users find an email to be not relevant (*e.g.,* wrongly retrieved by the recommendation engine), they can uncheck the box and the email is not presented again. This information is stored in a table in the case of PostgreSQL, while in the internal cache when using MBox files.

III. Program comprehension through emails

We present scenarios to illustrate the benefits of the program comprehension support that Remail offers by recommending emails. We explore two open source projects, Apache Mina[5] and Freenet[6], from unrelated domains, with different size, and emerging from distinct communities. Table I contains a numerical overview of the two systems.

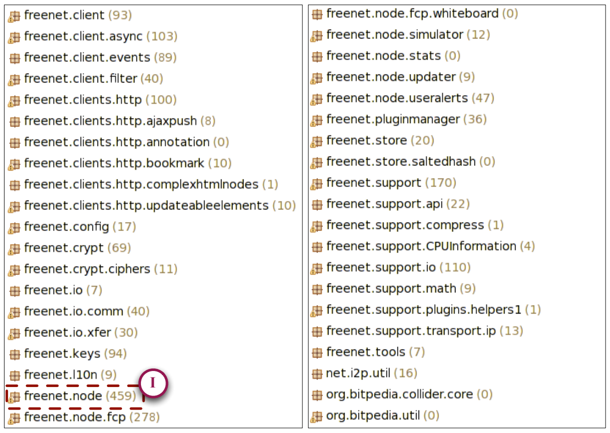
Project	Classes	Mailing List	Messages
Freenet	1,424	org.freenetproject.devl	22,859
Apache Mina	408	org.apache.mina.dev	21,079

Table I
Investigated software systems

A. Entry points from class popularity in emails

When starting a program comprehension effort, a crucial issue is knowing where to begin the investigation. Email data provides both qualitative and quantitative information for this purpose. The augmented Package Explorer view, which shows decorations with the number of emails related to the chosen packages and classes, gives hints on the "popularity" of entities in the mailing list, in quantitative terms. We argue that this value might be high in classes that implement the core functionalities of a system, thus it might be used for recommending *entry points* for program comprehension.

This "secondary" recommendation, based on popularity, can be easily *contextualized* and evaluated by the practitioner, thanks to the qualitative aspect of emails. In other words, this popularity is not a value coming out of the blue, but, on the contrary, it is supported by the content of the emails: By skimming the messages' text, one can decide whether a popular class is worth understanding for the task at hand.
Freenet: Figure 6 reports the popularity of Freenet packages, as shown by Remail's Package Explorer decorator.



```
freenet.client (93)
freenet.client.async (103)
freenet.client.events (89)
freenet.client.filter (40)
freenet.clients.http (100)
freenet.clients.http.ajaxpush (8)
freenet.clients.http.annotation (0)
freenet.clients.http.bookmark (10)
freenet.clients.http.complexhtmlnodes (1)
freenet.clients.http.updateableelements (10)
freenet.config (17)
freenet.crypt (69)
freenet.crypt.ciphers (11)
freenet.io (7)
freenet.io.comm (40)
freenet.io.xfer (30)
freenet.keys (94)
freenet.l10n (9)
freenet.node (459)
freenet.node.fcp (278)
freenet.node.fcp.whiteboard (0)
freenet.node.simulator (12)
freenet.node.stats (0)
freenet.node.updater (9)
freenet.node.useralerts (47)
freenet.pluginmanager (36)
freenet.store (20)
freenet.store.saltedhash (0)
freenet.support (170)
freenet.support.api (22)
freenet.support.compress (1)
freenet.support.CPUInformation (4)
freenet.support.io (110)
freenet.support.math (9)
freenet.support.plugins.helpers1 (1)
freenet.support.transport.ip (13)
freenet.tools (7)
net.i2p.util (16)
org.bitpedia.collider.core (0)
org.bitpedia.util (0)
```

Figure 6. Excerpt from Package Explorer: Freenet packages with popularity

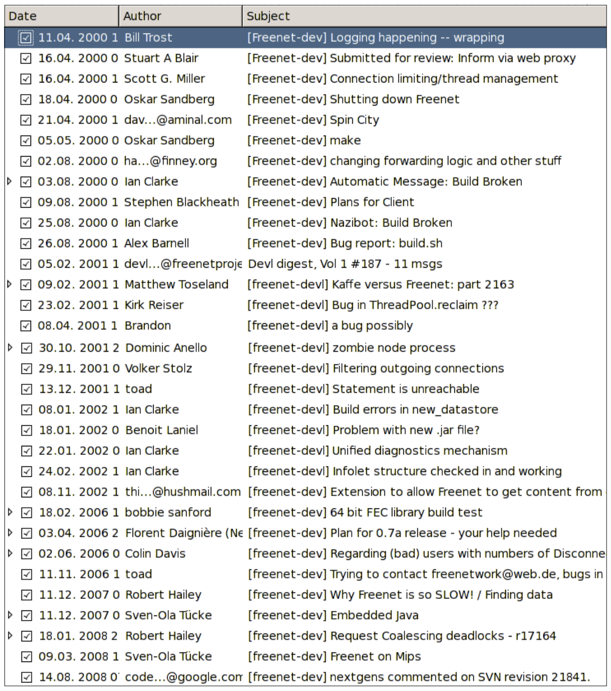
It shows that the most popular package is *freenet.node* (Point I), with classes discussed in more than 450 emails.

[5]http://mina.apache.org/
[6]http://freenetproject.org/

The second most discussed package is *freenet.node.fcp*, with slightly more than 250 emails, while the other packages are significantly less popular.

We investigate the most popular package: In the *node* package, developers mainly discuss four entities: classes *Node* (74 emails), *PacketSender* (61), and *PeerNode* (67), and interface *RequestClient* (98). With a brief analysis of the emails for the interface, we see that it was popular during the first phases of Freenet development, but afterwards its importance gradually faded. We focus on the other classes that are still currently discussed:

Node: By looking at the distribution of the emails over time, via the *Emails View* panel, we see (Figure 7) that the large class *Node* has been very popular since the inception of the project (*i.e.,* year 2000). By inspecting the code, we discover not only that it includes the main method from which the Freenet system bootstraps, but also that it models the node run by the user in the network. Since Freenet is a peer-to-peer system, the user node has a crucial role for the whole application and is essential for comprehending how the software functions as a whole.

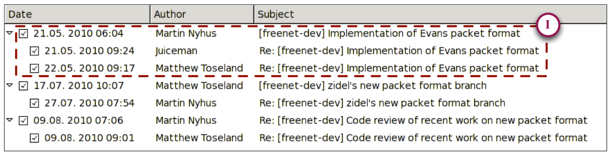


Date	Author	Subject
☑ 11.04. 2000 1	Bill Trost	[Freenet-dev] Logging happening -- wrapping
☑ 16.04. 2000 0	Stuart A Blair	[Freenet-dev] Submitted for review: Inform via web proxy
☑ 16.04. 2000 1	Scott G. Miller	[Freenet-dev] Connection limiting/thread management
☑ 18.04. 2000 0	Oskar Sandberg	[Freenet-dev] Shutting down Freenet
☑ 21.04. 2000 1	dav…@aminal.com	[Freenet-dev] Spin City
☑ 05.05. 2000 0	Oskar Sandberg	[Freenet-dev] make
☑ 02.08. 2000 0	ha…@finney.org	[Freenet-dev] changing forwarding logic and other stuff
▷ ☑ 03.08. 2000 0	Ian Clarke	[Freenet-dev] Automatic Message: Build Broken
☑ 09.08. 2000 1	Stephen Blackheath	[Freenet-dev] Plans for Client
☑ 25.08. 2000 0	Ian Clarke	[Freenet-dev] Nazibot: Build Broken
☑ 26.08. 2000 1	Alex Barnell	[Freenet-dev] Bug report: build.sh
☑ 05.02. 2001 1	devl…@freenetproje	Devl digest, Vol 1 #187 - 11 msgs
▷ ☑ 09.02. 2001 1	Matthew Toseland	[freenet-devl] Kaffe versus Freenet: part 2163
☑ 23.02. 2001 1	Kirk Reiser	[freenet-devl] Bug in ThreadPool.reclaim ???
☑ 08.04. 2001 1	Brandon	[freenet-devl] a bug possibly
▷ ☑ 30.10. 2001 2	Dominic Anello	[freenet-devl] zombie node process
☑ 29.11. 2001 0	Volker Stolz	[freenet-devl] Filtering outgoing connections
☑ 13.12. 2001 1	toad	[freenet-devl] Statement is unreachable
☑ 08.01. 2002 1	Ian Clarke	[freenet-devl] Build errors in new_datastore
☑ 18.01. 2002 0	Benoit Laniel	[freenet-devl] Problem with new .jar file?
☑ 22.01. 2002 0	Ian Clarke	[freenet-devl] Unified diagnostics mechanism
☑ 24.02. 2002 1	Ian Clarke	[freenet-devl] Infolet structure checked in and working
☑ 08.11. 2002 1	thi…@hushmail.com	[freenet-dev] Extension to allow Freenet to get content from
▷ ☑ 18.02. 2006 1	bobbie sanford	[freenet-dev] 64 bit FEC library build test
▷ ☑ 03.04. 2006 2	Florent Daignière (Ne	[freenet-dev] Plan for 0.7a release - your help needed
▷ ☑ 02.06. 2006 0	Colin Davis	[freenet-dev] Regarding (bad) users with numbers of Disconne
☑ 11.11. 2006 1	toad	[freenet-dev] Trying to contact freenetwork@web.de, bugs in
☑ 11.12. 2007 0	Robert Hailey	[freenet-dev] Why Freenet is so SLOW! / Finding data
▷ ☑ 11.12. 2007 0	Sven-Ola Tücke	[freenet-dev] Embedded Java
▷ ☑ 18.01. 2008 2	Robert Hailey	[freenet-dev] Request Coalescing deadlocks - r17164
☑ 09.03. 2008 1	Sven-Ola Tücke	[freenet-dev] Freenet on Mips
☑ 14.08. 2008 0'	code…@google.com	[freenet-dev] nextgens commented on SVN revision 21841.

Figure 7. Excerpt from Emails View: emails recommended for class *Node*

PacketSender: This class implements packet sending through the Freenet network. It has a general importance in the system, and by reading one recommended email, we understand that it is *critical* for a developer who must deal with message handling: "Are you interested in implementing message priorities? MessageItem and PacketSender are the most relevant classes." This message also reveals a hidden coupling, not detectable by static analysis, with *MessageItem*.

PeerNode: The opening code comment of *PeerNode* states that it "represents a peer we are connected to." Therefore, it plays a central role in the Freenet functioning and is another important entry point for program comprehension. Moreover, by reading among the most recent email threads recommended by Remail (Figure 8, Point I), we discover additional information that could not have been learned by solely investigating the code. *PeerNode* is responsible for implementing the Freenet Network Protocol (FNP) – the communication protocol used in Freenet. A developer who must change this protocol is required to "move all the FNP related code from PeerNode to a new class, and have PeerNode use the old code through this class. The new code can then be added without touching FNP, and PeerNode [can] choose which format to use for each peer." By reading the same thread, the developer interested in modifying FNP would also discover the other two classes responsible for the implementation of FNP: *PacketTracker* and *SessionKey*.



Date	Author	Subject
▽ ☑ 21.05. 2010 06:04	Martin Nyhus	[freenet-dev] Implementation of Evans packet format
☑ 21.05. 2010 09:24	Juiceman	Re: [freenet-dev] Implementation of Evans packet format
☑ 22.05. 2010 09:17	Matthew Toseland	Re: [freenet-dev] Implementation of Evans packet format
▽ ☑ 17.07. 2010 10:07	Matthew Toseland	[freenet-dev] zidel's new packet format branch
☑ 27.07. 2010 07:54	Martin Nyhus	Re: [freenet-dev] zidel's new packet format branch
▽ ☑ 09.08. 2010 07:06	Martin Nyhus	Re: [freenet-dev] Code review of recent work on new packet format
☑ 09.08. 2010 09:01	Matthew Toseland	Re: [freenet-dev] Code review of recent work on new packet format

Figure 8. Emails View: recent threads recommended for *PeerNode*

To further evaluate the importance of these three classes in the system, we analyzed them in terms of *Design Flaws* [13]. The detection strategies we use (see [15]) diagnose all the three classes as affected by the *Behavioral God Class* [18] design flaw, *i.e.,* they tend to incorporate a disproportionately large amount of intelligence. This reinforces the hypothesis that they represent an important entry point for program comprehension. Additionally, all the classes presents other design flaws such as *Brain Methods*, *Intensive Coupling*, and *Shotgun Surgery*. Due to space constraints we do not analyze each design flaw, but refer the interested reader to [13].

Mina: MINA is an application framework for supporting the development of network applications. Within the project documentation, developers offer a decomposition of the system to introduce newcomers to its architecture. Figure 9 replicates this decomposition.

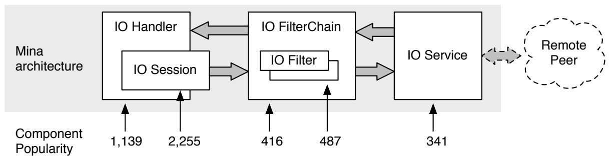


Figure 9. Mina architecture: Main components and their popularity

With this decomposition in place, we compare it to the popularity of its component. For each component (*e.g., IO Handler*), we report the popularity of the corresponding class (*e.g., core.service.IOHandler*). Only in the case of *IO Filter*

Chain, we sum the popularity of the interface *IOFilterChain* and of its sole implementer *DefaultIOFilterChain*.

The popularity values depicted in the picture show how the most discussed components clearly match the architecture proposed by the developers. In fact, only a few other classes reach such a popularity score: *IOBuffer* (541 emails), the replacement of the JAVA library class ByteBuffer, necessary for writing on any *IO Session*; *IOAcceptor* (390 emails) and *IOConnector* (347) used for starting a server and a client, respectively; and *ProtocolCodecFilter* (1,010), a specialized IO filter also detailed in the official decomposition. In other words, we could have extracted an almost equivalent decomposition simply by using quantitative data from emails.

On an unrelated note, we see how the number of emails talking about the most popular classes is rather higher compared to Freenet, even though the respective mailing lists archive a similar total number of messages. This reflects the different programming and communication habits between the two development communities.

B. *Software Evolution Analysis*

Version control (or software configuration management–SCM) systems offer historical information on the evolution of the source code. They can be used to track *changes* in source code artifact in order to detect whether a class is stable or always morphing. For example, researchers have successfully used *change metrics* to predict defects [4], [17].

We argue that, in system in which developers mainly communicate in mailing lists (such as open source projects, and distributed teams), emails might be used to *complement* known information in order to better understand the relevance of changes. The rationale is that classes that are not discussed in the development mailing lists are likely to be more stable and less prone to major modifications, since a substantial change would require an agreement of the development team.

To investigate our hypothesis, we analyze the package *util* in the Mina project. It presents 98 distinct commits in fifty months, and discussions in the mailing list. It contains 17 classes, of which we manually inspected the complete history in the SCM system and all the emails recommended. We analyze the changes that occurred since the inception of the mailing list, in 2006.

Figure 10 visualizes the results of our analysis. We divide the time in one-month slots and fill them with the occurred events. Grey boxes represent related emails, the other boxes represent commits. Light blue boxes represent class addition, white boxes represent minimal changes (*e.g.,* author renaming, license change, reformatting), and red boxes represent relevant changes that modified a class' behavior.

The first thing we note is the small number of relevant changes: Only 6 relevant changes over 98 commits. From our hypothesis, we expected this behavior, since the mailing list is silent on most of the classes. We note how there is no relevant change in classes not mentioned in the mailing

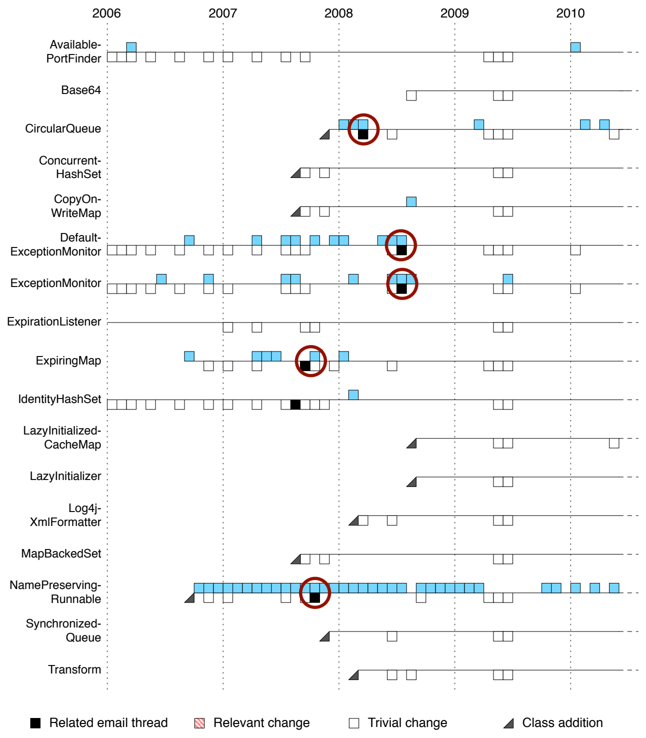


Figure 10. Package *org.apache.mina.util*: changes and discussions

list. Concerning the relevant changes, we see (circled in Figure 10) how these happen close in time to discussion related to the class. The only exception is *IdentityHashSet*, where the related discussion happens after many months.

An analogous pattern is present in the Freenet system. For example, the package *crypt* contains classes that used to be discussed in the past of the project, but that have been almost no discussed for two years. By analyzing the related commits, we verify that they contain no relevant change.

From this scenario, we see how historical information provided by emails might help in understanding where the current, active, and relevant development is focused.

C. *Expert finding*

Program comprehension involves keeping up with who on a distributed team is expert about specific code entities. Given the complexity and the amount of changes in software, this is a non-trivial task. Researchers have proposed a number of approaches to recommend experts (*e.g.,* [16], [20], [22]), most of them based on authorship of code: The person committing changes to an artifact has expertise in it. Emails recommended by Remail also report the author (see Point II and III of Figure 2). We argue that this information can be used to extract both quantitative and qualitative information about the expertise on discussed entities. As an example, we see how Remail can be used to find an expert of class *BookmarkItem*.

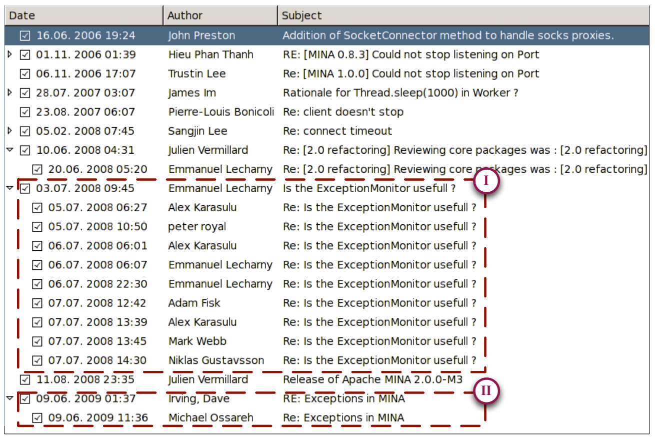
We first selected the entity itself (cf. Figure 2, Point I), which has 26 related emails. Then, in the *Emails View* (Point

II), we already see how Toseland wrote several emails on this entity. This quantitative information suggests us his potential class expertise. To confirm this belief, we select one of the emails he wrote and we read it in *Email Content View* (Point III). Thanks to the different colors used to distinguish the quotation levels, we clearly read that Toseland, first, indicated how *BookmarkItem* must be aggregated: "you should put each BookmarkItem as a sub-fieldset, not a string"; then, gave the rationale behind this behavior: "[to have] the ability to easily add fields." Without using SCM system data, but simply with Remail, we easily found an expert of this entity.

As another example, we see how Remail *complements* the expertise information we find in the SCM system. We analyze the Mina class *ExceptionMonitor*. Table II reports the SCM system commits involving this class, by date and author. Only four developers committed on this class: They are the exclusive experts from a SCM system point of view.

Revision	Date	Author
995,776	Oct 2010	elecharny
900,040	Jan 2010	elecharny
783,334	Jun 2009	elecharny
774,593	May 2009	elecharny
678,335	Jul 2008	mwebb
671,827	Jun 2008	jvermillard
576,217	Sep 2007	trustin
565,669	Aug 2007	trustin
555,855	Jul 2007	trustin
497,314	Jan 2007	trustin

Table II
CODE COMMITS INVOLVING *org.apache.mina.util.ExceptionMonitor*



Date	Author	Subject
16.06. 2006 19:24	John Preston	Addition of SocketConnector method to handle socks proxies.
01.11. 2006 01:39	Hieu Phan Thanh	RE: [MINA 0.8.3] Could not stop listening on Port
06.11. 2006 17:07	Trustin Lee	Re: [MINA 1.0.0] Could not stop listening on Port
28.07. 2007 03:07	James Im	Rationale for Thread.sleep(1000) in Worker ?
23.08. 2007 06:07	Pierre-Louis Bonicoli	Re: client doesn't stop
05.02. 2008 07:45	Sangjin Lee	Re: connect timeout
10.06. 2008 04:31	Julien Vermillard	Re: [2.0 refactoring] Reviewing core packages was : [2.0 refactoring]
20.06. 2008 05:20	Emmanuel Lecharny	Re: [2.0 refactoring] Reviewing core packages was : [2.0 refactoring]
03.07. 2008 09:45	Emmanuel Lecharny	Is the ExceptionMonitor usefull ?
05.07. 2008 06:27	Alex Karasulu	Re: Is the ExceptionMonitor usefull ?
05.07. 2008 10:50	peter royal	Re: Is the ExceptionMonitor usefull ?
06.07. 2008 06:01	Alex Karasulu	Re: Is the ExceptionMonitor usefull ?
06.07. 2008 06:07	Emmanuel Lecharny	Re: Is the ExceptionMonitor usefull ?
06.07. 2008 22:30	Emmanuel Lecharny	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 12:42	Adam Fisk	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 13:39	Alex Karasulu	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 13:45	Mark Webb	Re: Is the ExceptionMonitor usefull ?
07.07. 2008 14:30	Niklas Gustavsson	Re: Is the ExceptionMonitor usefull ?
11.08. 2008 23:35	Julien Vermillard	Release of Apache MINA 2.0.0-M3
09.06. 2009 01:37	Irving, Dave	RE: Exceptions in MINA
09.06. 2009 11:36	Michael Ossareh	Re: Exceptions in MINA

Figure 11. Emails recommendend for *mina.util.ExceptionMonitor*

Figure 11 reports the emails related to *ExceptionMonitor*, as recommended by Remail. The aforementioned committers were all involved in discussions, in a moment in time close to their activity on the class. For example, we see that Trustin Lee (trustin, in commits) wrote emails in 2006 and worked on the class up to 2007. In 2008, Mark Webb and Julien Vermillard where both committing and discussing the class. Currently Emmanuel Lecharny is the sole committer.

By reading the thread in Point I, we learn that other developers designed the class, but are now no longer involved in the implementation (*e.g.,* Karasulu: "I think this was something Trustin and I talked about while experimenting with Monitors versus logging. [This class] was a bad idea then and I think it is a bad idea now."). At the same time, by reading Point II, we see that Dave Irving is knowledgeable about the functioning of the class: "an ExceptionMonitor instance [...] doesn't swallow exceptions (or captures them for relaying back to your test if you're not running off the main thread)." By using only the SCM system data, we would have not been able to discover his expertise.

D. Recovering Additional Information

Official documentation, *e.g.,* design documents, is regarded by developers as "write-only media", as it is difficult and cumbersome to evolve along with the rest of the systems [14]. Moreover, not all the developers have the rights to modify it. On the contrary, emails are easier to write, due to their less formal nature, and can be written by non-developers. We claim that recommended emails might contain *complementary* information not available in code comments, or official documentation, thus helping program comprehension.

We present the example of the class *CircularQueue* (CQ), in the Mina system, which provides anecdotal evidence of our hypothesis. The SCM system (as shown in Figure 10) reports five commits on this class in the last months, thus underlying a sort of evolution. However, from a previous scenario we know that these changes are not relevant and the class is almost frozen. For this reason, one would wonder its real purpose, which components use this class, and why it is stable. The only information provided by the class comments and documentation is that CQ implements an "unbounded circular queue based on array." However, we can read a recent thread titled "About CircularQueue", among those recommended by Remail as pertaining to CQ, to gain additional information. From the emails we discover that this class has a logical connection to *ConcurrentLinkedQueue* (CLQ). Even though the latter "performs bad comparing to synchronized [CQ] when the number of accessing threads are very small", developers decided to remove "all references to the non-thread-safe [CQ] data structure, and replace it with a [CLQ]." The reason is that "not only [CLQ] is a comparable data structure, but it's also thread safe, and tested." Eventually, we read that developers are considering to "remove the [CQ] data structure from the code base" as it should only be used by the core and not by framework users.

Thanks to this additional information, we learned what the most important issues of the class are (*i.e.,* not-thread-safe and not well tested), which components use it (*i.e.,* only internal core classes), and why it has not been changed significantly lately (*i.e.,* it is probably going to be removed).

IV. Discussion

Limitations: The main limitations of our approach are caused by the fact that it was validated only on two open-source systems, and that the evidence is so far anecdotal.

For example, concerning the entry point analysis (cf. Section III-A), in the studied systems we found that classes are popular because they form the system core. Nevertheless, artifacts in other systems might be popular in emails for other reasons (*e.g.,* because of their size, or because they are subject to frequent changes); it would be necessary to analyze whether these classes can also be considered as valid entry points for comprehending a system.

Future work: We plan to perform a systematic evaluation of how and whether emails can aid program comprehension tasks. For example, we plan to involve open-source developers to evaluate Remail and learn from their opinion, and to conduct a controlled experiment to compare Remail with other tools that improve program comprehension.

Email archives are not the panacea: With massive amount of data—even if useful—also comes *information overload*. Our linking techniques are able to find emails related to the chosen artifact, thus filtering much non-relevant information, but large systems with a long history can archive thousands of emails concerning a single artifact. In our study, we found that the information about the email date, the aggregation in threads, and visibility of the email subject at a glance, are functional features to find searched information; however, we plan to improve data mining by including new filters on topics and emerging trends, and by linking to code changes (*e.g.,* to see emails close in time to the specified commit).

Finally, we plan to conduct an empirical study on the correlation between email popularity and design flaws. In our study, we found that Freenet flaws are related to popularity, but we want to investigate whether this relation is stable in other systems.

V. Related Work

This work builds on the lessons learned by presenting and discussing our prototype of Remail at the SUITE 2010 Workshop [2]. Since then, thanks to the positive feedback and suggestions received, we extensively expanded our work. We improved the overall stability of Remail, added new visualizations (such as the popularity decoration in the Package Explorer) and ameliorated existing ones (*e.g.,* by including threads in the *Emails View*), and by adding MBox files as an alternative, simple, and transparent data source. In addition, in the present work, we illustrated how Remail can be used to augment program comprehension.

An approach similar to ours, which includes external artifacts in the development environment, has been devised by Holmes and Begel [9]. They presented *Deep Intellisense*, a Visual Studio IDE plugin that links a number of artifacts (*e.g.,* bug reports, e-mails, code changes) to source code entities. It features three views: a structural view of the artifact chosen, a view to represent people related to that artifact, and a view to display all the related historical information (*e.g.,* checkins, or bug reports). As opposed to our approach, Deep Intellisense always relies on external applications to handle the visualization of different artifacts: For example, when a user clicks on a bug, the native viewer is open. By implementing Remail, we strive to give the users views that are consistent with the development environment, in order to avoid unnecessary context switches, and to allow more interaction. For example, as our future work, we plan to allow the developer to click on the content of an email to trigger appropriate events in the IDE. This would not be possible by relying on external viewers. In addition, Deep Intellisense needs a specialized database with all the information modeled, in order to use its implicit query system [24]. By using MBox files, we removed the burden of such complex infrastructure from users' shoulders, thus lowering the bar for adopting Remail in everyday development.

Hipkat offers an integrated approach to access information stored in project archives [23]. Similarly to our approach, it is an Eclipse plugin. The main difference resides on fact that Hipkat requires an external server process to monitor sources, to store them in a database, to identify links, and to reply to requests sent by the client. While, in Remail, all these operations are conducted in the client, since we strived to implement our approach as fast, lightweight, and unobtrusive. In addition, the users of Remail do not need to specify any query for retrieving the relevant data, they simply choose the entities, and emails are automatically recommended.

IBM Jazz[7] offers a framework, built on top of Eclipse, to support collaborative software development. It features IM communication in the IDE, and uses the concept of "work items" to track and coordinate development tasks and workflows. Each work item is connected to other artifacts (*e.g.,* builds, defect reports, change sets, or source code entities). Such work items have analogies with emails, even though the latter might have a broader perspective. While Jazz advises the use of a brand new technology, we decided on harnessing the power of emails, a pre-existing popular communication means successfully adopted by developers of a vast majority of software projects. This allows developers to take advantage of mailing lists that archive years of relevant information about the development of software systems.

Mylyn offers the concept of "task context" that focuses on automatically link all relevant artifacts to the task-at-hand [11]. As for Remail, this system helps in reducing information overload and easing the sharing of expertise. However, as for the case of Jazz, it also requires an additional technology to be used, thus it does not provide access to differently archived data and requires a higher learning effort by developers. At the same time, Remail and Mylyn can coexist in the same IDE and provide complementary data.

[7]http://jazz.net/

VI. Conclusion

We presented a new approach for program comprehension based on email information. We implemented Remail, an Eclipse plugin to integrate email communication. It enables the connection between code artifacts and emails, within the programming environment. By using code-to-emails lightweight linking techniques, Remail allows the user to easily retrieve discussion relevant to the chosen entities. Remail revolves around two aspects:

1) *Simplicity:* Remail allows developers to take advantage of email archives already present in their common email clients and to easily import new archives via Markmail or Mailman. Also, it lets practitioners find and read relevant emails, for a chosen entity, with one click.
2) *Integration:* Remail smoothly integrates with email clients: When Remail uses an archive of a client, it will not interfere with its functioning, but, on the contrary, will take advantage of the updates performed by the user from the client, by updating its own data. In addition, new archives imported by Remail can immediately also be used by email clients. Also, Remail integrates with the IDE: It has internal views to avoid context switches and ease concurrent code and email inspection.

The main contribution of Remail is disclosing both *qualitative and quantitative* information provided by email archives, so that it can be used during software development to improve program comprehension. We have shown how the email information, as displayed by Remail, helps to find entry points in an unknown system, understand software evolution, identify experts, and complement missing documentation. The main strength of Remail resides in the fact that it recommends emails–discussions in a context. The context is vital for users to verify the value of a recommendation.

Acknowledgements: The Swiss National Science foundation's support for the project "SOSYA" (SNF Project No. 132175).

References

[1] A. Bacchelli, M. D'Ambros, M. Lanza, and R. Robbes. Benchmarking lightweight techniques to link e-mails and source code. In *Proceedings of WCRE 2009 (16th IEEE Working Conference on Reverse Engineering)*, pages 205–214. IEEE CS, 2009.

[2] A. Bacchelli, M. Lanza, and V. Humpa. Towards integrating e-mail communication in the IDE. In *Proceedings of SUITE 2010 (2nd International Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation)*, pages 1–4, 2010.

[3] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *Proceedings of ICSE 2010 (32nd International Conference on Software Engineering)*, pages 375–384. ACM, 2010.

[4] A. Bernstein, J. Ekanayake, and M. Pinzger. Improving defect prediction using temporal features and non linear models. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 11–18. IEEE CS, 2007.

[5] T. A. Corbi. Program understanding: challenge for the 1990's. *IBM System Journal*, 28(2):294306, 1989.

[6] L. Erlikh. Leveraging legacy system dollars for e-business. *IT Professional*, 2(3):17–23, 2000.

[7] K. Fogel. *Producing Open Source Software*. O'Reilly, 2005.

[8] T. Gleixner. The realtime preemption patch: Pragmatic ignorance or a chance to collaborate? In *Keynote of ECRTS 2010 (22nd Euromicro Conference on Real-Time Systems)*, 2010. http://lwn.net/Articles/397422/.

[9] R. Holmes and A. Begel. Deep Intellisense: a tool for rehydrating evaporated information. In *Proceedings of MSR 2008 (5th IEEE Working Conference on Mining Software Repositories)*, pages 23–26. ACM, 2008.

[10] W. M. Ibrahim, N. Bettenburg, E. Shihab, B. Adams, and A. E. Hassan. Should I contribute to this discussion? In *Proceedings of MSR 2010*, pages 181–190. IEEE, 2010.

[11] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for IDEs. In *Proceedings AOSD 2005 (4th international conference on Aspect-oriented software development)*, pages 159–168. ACM, 2005.

[12] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proceedings of ICSE 2007*, pages 344–353. IEEE CS, 2007.

[13] M. Lanza and R. Marinescu. *Object-Oriented Metrics in Practice*. Springer-Verlag, 2006.

[14] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *Proceedings of ICSE 2006*, pages 492–501. ACM, 2006.

[15] R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *Proceedings of ICSM 2004 (20th IEEE International Conference on Software Maintenance)*, pages 350–359. IEEE Computer Society Press, 2004.

[16] A. Mockus and J. D. Herbsleb. Expertise Browser: a quantitative approach to identifying expertise. In *Proceedings of ICSE 2002*, pages 503–512, 2002.

[17] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of ICSE 2008*, pages 181–190, 2008.

[18] A. Riel. *Object-Oriented Design Heuristics*. Addison-Wesley, 1996.

[19] M. P. Robillard, R. J. Walker, and T. Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2010.

[20] D. Schuler and T. Zimmermann. Mining usage expertise from version archives. In *Proceedings of MSR 2008*, pages 121–124. ACM, 2008.

[21] R. C. Seacord, D. Plakosh, , and G. A. Lewis. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley, 2003.

[22] G. C. M. Thomas Fritz, Jingwen Ou and E. Murphy-Hill. A degree-of-knowledge model to capture source code familiarity. In *Proceedings of ICSE 2010*, pages 385–394. IEEE CS, 2010.

[23] D. Čubranić and G. C. Murphy. Hipikat: recommending pertinent software development artifacts. In *Proceedings of ICSE 2003*, pages 408–418, 2003.

[24] G. Venolia. Textual allusions to artifacts in software-related repositories. In *Proceedings of MSR 2006*, pages 151–154. ACM, 2006.