Exploring, Exposing, and Exploiting Emails to Include Human Factors in Software Engineering

Alberto Bacchelli
REVEAL @ Faculty of Informatics
University of Lugano
alberto.bacchelli@usi.ch

ABSTRACT

Researchers mine software repositories to support software maintenance and evolution. The analysis of the structured data, mainly source code and changes, has several benefits and offers precise results. This data, however, leaves communication in the background, and does not permit a deep investigation of the human factor, which is crucial in software engineering.

Software repositories also archive documents, such as emails or comments, that are used to exchange knowledge among people—we call it "people-centric information." By covering this data, we include the human factor in our analysis, yet its unstructured nature makes it currently sub-exploited.

Our work, by focusing on email communication and by implementing the necessary tools, investigates methods for exploring, exposing, and exploiting unstructured data. We believe it is possible to close the gap between development and communication, extract opinions, habits, and views of developers, and link implementation to its rationale; we see in a future where software analysis and development is routinely augmented with people-centric information.

Categories and Subject Descriptors

D.2.7 [**Distribution, Maintenance, and Enhancement**]: Enhancement

General Terms

Mining Software Repositories, Data Analysis, Human Factors

Keywords

Toolset, Unstructured Data, Email Communication

1. MOTIVATION

Human factors play a key role in software engineering processes. We write this inspired by three classics in the software engineering literature. The first, *How do committees invent?* [15], in 1968 presented the "Conway's Law," and told us that any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure. The second, *The Mythical Man-Month* [14], in 1975 made us aware that the *number of people* involved in a software project increases its complexity–exponentially. The third, *Peopleware* [29], in 1987 explained us that in more than five hundred failed software projects "there was not a single technological issue to explain the failure."

From the shoulders of these giants, we claim that by understanding and exploiting human factors we can improve software quality through better analysis, tools, and processes.

2. MINING SOFTWARE REPOSITORIES

By mining software repositories, researchers have produced valuable results. The focus has mainly been on repositories of *structured information*, *i.e.,* artifacts, such as source code, designed to be automatically parsed, *e.g.,* compilers.

For example, source code has been studied, analyzed, and measured to asses the quality of software systems' internal structure (*e.g.,* [26, 30]) or to predict the location of future software defects (*e.g.,* [8, 16, 32, 39, 43]); SCM repositories, which attracted the interest of researchers more recently [7, 44], are being studied to understand system evolution [17, 23], to identify poorly designed components [36], to devise new debugging methods [42], or to improve defect prediction (*e.g.,* [31, 34]); issue repositories are being mined to investigate the system evolution by studying its defects: Researchers devised defect prediction techniques studying historical defects (*e.g.,* [10, 24, 25]), or performed retrospective system analysis to understand the most problematic parts of a system [18].

But like any tool, structured data is not good for everything. There are some kinds of questions for which structured data answers little: In the context of our research—*i.e.,* understanding and exploiting human factors—structured data limit us to partial views. Let us imagine two developers working in the same project: We parse code changes to determine whether they worked on the same files in the same time period; we parse email metadata, to verify whether they also communicated in these days; but we do know whether they were really collaborating, whether they were aware of one another work, and whether they had according plans. To have answers, we should query them. Or we could study the *content* of their discussions or commit comments to get deeper insights. In fact, people write such documents to share *knowledge* with other people, thus they contain facts and qualitative data that answer new kinds of questions.

We believe that natural language (NL) documents—if correctly mined, measured, and made available—can integrate, consolidate, and complement the data extracted from structured sources, because they include human factors. The repositories that store artifacts with textual narrative, however, are still largely unexplored. The main reason lies in the difficulty of extracting significant information and quantitative data by parsing natural language (NL) text.

ICSE'11, May 21–28, 2011, Waikiki, Honolulu, HI, USA
ACM 978-1-4503-0445-0/11/05

2.1 Emails as Our Source of Information

Among the various kinds of artifacts written in NL and created while a software evolves (*e.g.,* design documents, meeting reports, forums, development and usage documentation, code comments, instant messages), we consider emails in our research, since we believe that they offer a valuable source of information. Mailing lists of software projects store the communications occurred among project members and users, and discussions span all the topics that could arise during software development and usage. Email content can range from low-level concerns (*e.g.,* refactoring, code formatting, documentation fixing) to high-level resolutions (*e.g.,* future planning, design decisions, API changes, significant refactoring). When considering open-source projects, "mailing lists are the bread and butter of project communication" [21]; also in co-located teams emails are a widely used communication mean [27].

By investigating email archives, we can conduct at least two kinds of analyses: *social analyses* and *technical analyses*. The former focuses on studying and interpreting the social phenomena, within a particular software project, or in the context of *software ecosystems* [28]; the latter extracts the data enclosed in mailing lists to tackle technical issues and extend system analysis.

Researchers conducted social analysis by using the *structured* email metadata (*e.g.,* author, date and time, threading): Bird *et al.* proposed techniques to mine email social networks, analyzed the correlation between email activity and source code activity [11, 12], and investigated social interactions among participants [13]. Ogawa *et al.* visualized social interaction among participants in open source projects [33]. Tang *et al.* proposed techniques for identifying the country origin of participants in open source mailing lists, and conducted a subsequent geographic analysis [40]. Shihab *et al.* performed an exploratory study on the role of mailing lists in open source projects, and showed that mailing list activity is related to source code activity, and mailing list discussions are good indicators of the types of source code changes happening in the project [37].

Only recently researchers have started analyzing the NL *content* of emails: Pattison *et al.* investigated the behavior of developers and users by studying the frequency with which terms of software entities are mentioned in emails, and correlating it with the number of system changes [35]. Baysal and Malton tried to correlate discussion in emails and source code [9]. In particular, they searched for a correlation between discussions and software releases, by applying data mining and Natural Language Processing techniques.

Even though textual narrative poses a serious challenge in exploiting NL content of emails, we believe that "we should not be dissuaded from our duty by the existence of textual narrative in artifacts" [20], because their content can broad our view on human factors and help us to improve system development and analysis.

3. APPROACH AND FIRST STEPS

Currently, we can see two approaches for exploiting the information stored in email archives: (1) enriching what we obtain from structured data (*e.g.,* source code) by complementing it with the information mined from mailing lists (as in Figure 1); (2) exploiting only mailing list data to provide complementary views and analyses.

Both the approaches require being able to import emails from the archives in which they reside, model their data, and store them in an easily accessible persistent format. Thus, as a first step, we created and continuously improve *Miler*, a toolset to explore email data pertaining to software systems [4].

Open source systems as case studies: To conduct our data exploration and assess our techniques and findings, we are performing case studies. Currently, we have only focused on email archives of

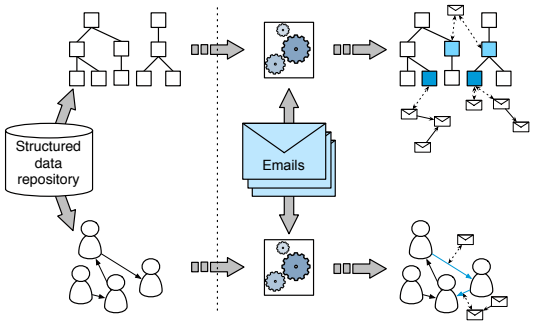

Figure 1: Augmenting structured information with email data

open source systems. This offers us great variety of data and projects to study, and allows us to share the datasets we use in our research to improve verifiability and reproducibility of our results. At the same time, this limits the generalizability of our findings. Even though we strive to build techniques that do not rely on features specific to open source systems, we cannot claim they will equally work in closed source and industrial settings. For this reason, we plan to conduct a research to understand how industrial practitioners communicate and whether our findings can be adapted to their context.

Recovering traceability links: In the first period of our research, we have mainly approached the problem of augmenting a system model generated from the source code. We think that emails can provide new data and metrics to enrich the already available information. As an example, emails pertaining to certain entities of the source code can integrate, or supply, incomplete *documentation*. Before being able to extract this information, emails must be *linked* to the discussed source code artifacts: Messages often reference other data sources, such as source code, but there is no actual linking to referenced artifacts. Moreover there is no link from code to emails.

We have tackled the recovering of traceability links between emails and source code devising lightweight techniques, based on text matching [3]. The comparison of these techniques to more sophisticated Information Retrieval (IR) methods showed us that the nature of email content favors our simpler approach [6].

Defects and email popularity: Restored the traceability link, the system model can be enriched with data extracted from email archives (second half of Figure 1). As a first step, we tested our belief that among the most discussed entities we also find the most defective. We devised metrics to seize the "popularity" of source code entities in the mailing list discussions, and we used them to perform defect prediction for object-oriented (OO) systems. We compared their predictive power to that of metrics obtained through structured data, and we got results similar to OO metrics, but inferior to change metrics. The most interesting result is that the *union* of metrics extracted from repositories with different form of data, *i.e.,* emails and change history, improves the *overall* predictive power [1]. This suggests that email data *complements* other forms of data.

Email communication in the IDE: Email data contains valuable information on the discussed software and that can be explored to obtain qualitative knowledge. For this reason, we implemented *Remail*, an Eclipse plugin, to make email data available in the place where developers spend most of their time—IDEs [27]. Remail includes our class-to-emails traceability techniques, and we used it to verify that tasks related to program comprehension and software development are enhanced by having email data at disposal [5].

Source code extraction: Since developers rely on personal mental models of systems they are developing [27], we think that we

could use development emails to reconstruct different models and views of a system, identifying inconsistencies, unseen features, and drifting evolutions. This requires extracting a system model from email archives alone. We are working on this aspect, which involves the extraction and reconstruction of structured data embedded in natural language content. As a first step, we devised text-matching based techniques able to separate source code from NL [2].

4. NEXT STEPS

We foresee next steps we could follow to mature a more solid contribution in using email data to improve software engineering.

Classification of emails and links: In our work on Remail, we verified its usefulness for program comprehension: By reading emails related to the classes at hand, we could improve our specific and general program knowledge. At the same time, this work allowed us to understand how some of the emails related to an entity have are more significance than others. In emails, entities can be mentioned in code snippets, patches, stack traces, or in NL sentences. We noticed that, in most of the cases, entities mentioned in NL sentences give us true *qualitative* information: Stack traces can be a list of classes with no special meaning, but a class included in a broader NL sentence is often well contextualized and explained. For this reason, in our next step, we want to find methods for classifying the different parts that compose emails. By having this information, we would be able to answer quantitative questions about mailing list content and usage (*e.g.,* are developers sharing code? Or, do users report stack traces in mailing list?), and we would be able to give a classification to our traceability links. Links would be associated with a tag distinguishing the text in which the entity is referenced, *e.g.,* NL or stack trace. We believe that links with NL tags might be useful for qualitative analysis, while links with other tags might be useful for quantitative analysis.

Chat coupling: Change coupling [22] detects evolutionary and implicit dependencies among artifacts of a system, by analyzing the history of their co-changes. We believe that we might analogously relate the artifacts that are often discussed together: If classes are mentioned in the same discourse, they could be logically connected. This information could reinforce the same structure we find in the source code, could confirm change coupling information, or could give insights on unexpected implicit dependencies. After the aforementioned link classification step, that would strengthen the information we provide with links, we plan to conduct a case study to test the importance of this metric for software engineering.

Opinion metrics: Our work on popularity, which relates classes popularity and defects, relies on simple quantitative data. Popularity counts the number of emails discussing about a class, the number of threads, the number of authors, *etc.*, but it does not seize *what* these authors in these emails say. We plan to improve our work by performing analysis of the *opinions* and *sentiments*, including methods from the IR and NL processing fields (*e.g.,* [19, 41]). Our target is understanding whether email authors are sharing an opinion about a source code entity: We expect negative opinions to better correlate with defects than mere number of emails, or give us more information about where we can improve a software system. This work requires a survey of the state of the art in opinion mining and sentiment analysis, in order to find the most appropriate approach for our domain. From our first analysis, we realized that it can be conducted only after the aforementioned classification step, as these techniques can only be applied to clean NL text.

Recovering structured knowledge: As previously mentioned, emails enclose structured information in the form of source code snippets, patches, stack traces, *etc.* These information can be extracted for our task of reconstructing different models and histories of a system, from the alternative point of view of emails. We are implementing an approach based on *island parsing* [38], which is providing extremely accurate results in extracting the whole structured information embedded in NL text. Currently, we are able to reconstruct an entire alternative model of a system by simply mining its mailing list. Indeed, it does not cover the whole system model (this can be done parsing the source code of any release), yet the contribution of this model is the novel perspective it gives on a system. Only part of the entities are mentioned, only some methods are discussed, only a portion of the relations are considered—it might open a brand new view on what is important or central in a system.

Events and trends in emails: The amount of emails generated around a software system can form a very extensive amount of information. For example, in the Linux kernel mailing list, developers and users currently exchange more than 10,000 messages a month. Finding the most interesting or relevant information in this amount of data can be a daunting task, especially for not experts of the system. We believe we can adopt methods from the IR research field to help us to find the way to the most important details enclosed in development mailing lists. More precisely, we are considering that the study of *unexpected events* and *emerging trends* might conduct us to interesting results. Our idea is that a moment in time in which we discover an unexpected event (*e.g.,* something almost never discussed appears) can be symptom of problems in the system or in the development team; similarly, an emerging trend might be a prelude of a change in the system. We believe that we can study these trends and events from an historical perspective to guide us to emails that could explain how and why a system reached a certain status. From a development point of view, we can keep track of emerging trends and unexpected events in projects we rely on. For example, if we use open source libraries, our analysis could monitor their mailing lists and alert us when we should be aware of upcoming changes, without the need of constantly reading the exchanged emails.

5. CONCLUSION

Our work aims at including human factors in software engineering research, by exploring, exposing, and exploiting email communication. In this paper, we presented what is the motivation that founds our work, we explained the approach we are currently following, and we enumerated the research paths that we would like to follow.

For exploiting mailing list data, we see two directions, which we are taking in parallel: Using the unstructured and qualitative information in mailing lists to *enrich* models obtained through the analysis of structured data; and considering mailing lists as an *alternative* data source that give additional and complementary information, which is independent from other forms of data.

In the former direction, we developed methods for restoring traceability links among emails and classes, we devised a new set of "popularity" metrics, and we included email information in the IDE. In the latter direction, we are currently tackling the issue of extracting the structured data embedded in the natural language of emails. We reached interesting *classification* results, by using lexical methods, currently, we are also implementing more sophisticated techniques based on island parsing that also allows us to *understand* and *expose* the meaning of extract structured information from emails.

As future steps, we believe that, for achieving a more solid contribution, we must give more evidence of the usefulness of unstructured data in emails. Here we proposed five different topics that we plan to investigate our future research: Better classification of emails, to recognize that even their content has a *structure*, which can be used for exposing more precise information; the extraction of *chat coupling*, a metric to measure the dependencies of code entities in mailing list discussions, and of *opinions* and *sentiments* expressed

by email authors; a more precise analysis of the structured data in emails for alternative system modeling; and an analysis of the evolution of a system, based on the study of *emerging trends* and *unexpected events* in the continuos stream of email messages.

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

6. REFERENCES

[1] A. Bacchelli, M. D'Ambros, and M. Lanza. Are popular classes more defect prone? In *Proc. of FASE 2010 (13th Int'l Conf. on Fundamental Approaches to Software Engineering)*, pages 59–73, 2010.

[2] A. Bacchelli, M. D'Ambros, and M. Lanza. Extracting source code from e-mails. In *Proc. of ICPC 2010 (18th IEEE Int'l Conf. on Program Comprehension)*, pages 24–33, 2010.

[3] A. Bacchelli, M. Lanza, and M. D'Ambros. Miler - a tool infrastructure to analyze mailing lists. In *Proc. of FAMOOSr 2009 (3rd Int'l Workshop on FAMIX and Moose in Reengineering)*, 2009.

[4] A. Bacchelli, M. Lanza, and M. D'Ambros. Miler: A toolset for exploring email data. In *Proc. of ICSE 2011*, page tbd, 2011.

[5] A. Bacchelli, M. Lanza, and V. Humpa. RTFM (Read The Factual Mails) –Augmenting program comprehension with remail. In *Proc. of CSMR 2011 (15th IEEE European Conf. on Software Maintenance and Reengineering)*, 2011.

[6] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *Proc. of ICSE 2010 (32nd Int'l Conf. on Software Engineering)*, pages 375–384. ACM, 2010.

[7] T. Ball, J.-M. Kim, A. Porter, and H. P. Siy. If your version control system could talk... In *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*, 1997.

[8] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996.

[9] O. Baysal and A. J. Malton. Correlating social interactions to release history during software evolution. In *Proc. of MSR 2007 (4th Int'l Workshop on Mining Software Repositories)*, page 7. IEEE CS, 2007.

[10] A. Bernstein, J. Ekanayake, and M. Pinzger. Improving defect prediction using temporal features and non linear models. In *Proc. of IWPSE 2005 (Int'l Workshop on Principles of Software Evolution)*, pages 11–18. IEEE CS Press, 2005.

[11] C. Bird, A. Gourley, P. T. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proc. of MSR 2006*, pages 137–143, 2006.

[12] C. Bird, A. Gourley, P. T. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks in Postgres. In *Proc. of MSR 2006*, pages 185–186, 2006.

[13] C. Bird, D. S. Pattison, R. M. D'Souza, V. Filkov, and P. T. Devanbu. Latent social structure in open source projects. In *SIGSOFT FSE*, pages 24–35, 2008.

[14] F. Brooks. *The Mythical Man-Month*. Addison-Wesley, 1995.

[15] M. E. Conway. How do committees invent? *Datamation*, 14(4):28–31, 1968.

[16] M. D'Ambros, A. Bacchelli, and M. Lanza. On the impact of design flaws on software defects. In *Proc. of QSIC 2010 (Int'l Conf. on Quality Software)*, pages 23–31. IEEE, 2010.

[17] M. D'Ambros, H. Gall, M. Lanza, and M. Pinzger. Analyzing software repositories to understand software evolution. In *Software Evolution*, pages 37–67. Springer, 2008.

[18] M. D'Ambros and M. Lanza. Software bugs and evolution: A visual approach to uncover their relationship. In *Proc. of CSMR 2006*, pages 227–236. IEEE CS Press, 2006.

[19] K. Dave, S. Lawrence, and D. M. Pennock. Mining the peanut gallery: opinion extraction and semantic classification of product reviews. In *Proc. of WWW 2003 (12th international conference on World Wide Web)*, pages 519–528. ACM, 2003.

[20] A. Dekhtyar, J. H. Hayes, and T. Menzies. Text is software too. In *Proc. of MSR 2004*, pages 22–26, 2004.

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

[22] H. Gall, M. Jazayeri, and J. Krajewski. CVS release history data for detecting logical couplings. In *Proc. of IWPSE 2003*, pages 13–. IEEE CS, 2003.

[23] T. Gîrba. *Modeling History to Understand Software Evolution*. PhD thesis, University of Berne, November 2005.

[24] A. E. Hassan and R. C. Holt. The top ten list: Dynamic fault prediction. In *Proc. of ICSM 2005 (21st IEEE Int'l Conf. on Software Maintenance)*, pages 263–272. IEEE CS, 2005.

[25] S. Kim, T. Zimmermann, J. Whitehead, and A. Zeller. Predicting faults from cached history. In *Proc. of ICSE 2007*, pages 489–498. ACM, 2007.

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

[27] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *Proc. of ICSE 2006 (28th ACM Int'l Conference on Software Engineering)*, pages 492–501. ACM, 2006.

[28] M. Lungu. *Reverse Engineering Software Ecosystems*. PhD thesis, University of Lugano, Switzerland, Oct. 2009.

[29] T. D. Marco. *Peopleware - Productive Projects and Teams*. Dorset House, 1999.

[30] R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *Proc. of ICSM 2004 (20th IEEE Int'l Conf. on Software Maintenance)*, pages 350–359. IEEE, 2004.

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

[32] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proc. of the ICSE 2006*, pages 452–461. ACM, 2006.

[33] M. Ogawa, K.-L. Ma, C. Bird, P. T. Devanbu, and A. Gourley. Visualizing Social Interaction in Open Source Software Projects. In *Proc. of APVIS 2007 (6th Int'l Asia-Pacific Symposium on Visualization)*, pages 25–32, 2007.

[34] T. J. Ostrand, E. J. Weyuker, and R. M. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005.

[35] D. S. Pattison, C. Bird, and P. T. Devanbu. Talk and work: a preliminary report. In *MSR*, pages 113–116, 2008.

[36] D. Ratiu, S. Ducasse, T. Gîrba, and R. Marinescu. Using history information to improve design flaws detection. In *Proc. of CSMR 2004*, page 223. IEEE CS, 2004.

[37] E. Shihab, Z. M. Jiang, and A. E. Hassan. Studying the use of developer IRC meetings in open source projects. In *Proc. of ICSM 2009*, pages 147–156. IEEE CS, 2009.

[38] O. Stock, R. Falcone, and P. Insinnamo. Island parsing and bidirectional charts. In *Proc. of the 12th Conf. on Computational Linguistics*, pages 636–641, 1988.

[39] R. Subramanyam and M. S. Krishnan. Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering*, 29(4):297–310, 2003.

[40] R. Tang, A. E. Hassan, and Y. Zou. Techniques for identifying the country origin of mailing list participants. In *Proc. of WCRE 2009*, pages 36–40. IEEE CS Press, 2009.

[41] J. Yi, T. Nasukawa, R. Bunescu, and W. Niblack. Sentiment analyzer: Extracting sentiments about a given topic using natural language processing techniques. In *Proc. of ICDM 2003 (3rd IEEE Int'l Conf. on Data Mining)*, pages 427–. IEEE, 2003.

[42] A. Zeller. Yesterday, my program worked. today, it does not. why? *ACM SIGSOFT Software Engineering Notes*, 24(6):253–267, 1999.

[43] T. Zimmermann and N. Nagappan. Predicting defects using network analysis on dependency graphs. In *Proc. of ICSE 2008*, pages 531–540. ACM, 2008.

[44] T. Zimmermann, P. Weissgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, 2005.