Software is Data Too

Andrian Marcus
Wayne State University
Department of Computer Science
5057 Woodward Ave., Detroit, MI 48202, USA
+1 313 577 5408
amarcus@wayne.edu

Timothy Menzies
West Virginia University
Lane Dept. of Comp. Science & Electrical Engineering
Morgantown, WV 26506, USA
+1 304 293 9127
tim@menzies.us

ABSTRACT
Software systems are designed and engineered to process data. However, software is data too. The size and variety of today's software artifacts and the multitude of stakeholder activities result in so much data that individuals can no longer reason about all of it. We argue in this position paper that data mining, statistical analysis, machine learning, information retrieval, data integration, etc., are necessary solutions to deal with software data. New research is needed to adapt existing algorithms and tools for software engineering data and processes, and new ones will have to be created.

In order for this type of research to succeed, it should be supported with new approaches to empirical work, where data and results are shared globally among researchers and practitioners. Software engineering researchers can get inspired by other fields, such as, bioinformatics, where results of mining and analyzing biological data are often stored in databases shared across the world.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement - *Restructuring, reverse engineering, and reengineering*; H.2.8. [**Database Management**]: Database Applications – *Data mining* H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval;

General Terms
Management, Algorithms, Performance, Experimentation

Keywords
Data mining, machine learning, information retrieval, statistical analysis, software engineering, empirical research.

1. HOW MUCH DATA IS IN SOFTWARE?
Software development no longer produces only source code and external documentation. The global nature of today's development processes and the complexity and size of most software systems result in staggering amounts of data. How much data are we talking about? Source code contains millions of lines of code and comments. Versioning systems keep copies of the source code that evolved over decades. Analysis data (static and dynamic) populate databases in the range of terabytes. Test data often exceeds the size of source code. Bug tracking systems store information not only about defects and their fixes, but also large amounts of text resulting from developer discussions about these bugs. In distributed development environments, e-mail communications between developers are also stored. Usage and run time data of deployed software is collected in huge databases. Developer activities (inside and outside a development environment) are often monitored and stored as well. Process and management information is routinely stored and analyzed. External documentation, which ranges from requirements to user manuals are part of any software system. These are only the more common types of data that is generated, stored, and used during the life of a software system. There is more than that and the picture is pretty clear. The amount of data is so large that people can no longer reason about it without specialized, computer assisted, tool support. Key in dealing with so much data is to extract what is important to different stakeholders and tasks. Building software is no longer just an engineering problem; it is also an information management problem.

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.

2. HOW TO DEAL WITH THE DATA?
Analysis and management of the software data is an activity that software engineers are not trained to do. We have to look for solutions outside software engineering, adopt them, and make them our own. These solutions can come from data mining, information retrieval, machine learning, statistical analysis, etc.

This is not the first time software engineers are looking at such solutions. It has been going on for about two decades, in a form or another. For example, data mining techniques have been proposed by many researchers to extract what is relevant to the stakeholders (i.e., developers, managers, testers, etc.) and help them understand data about a software system, its development process, and to make predictions about its future quality, cost, and evolution. Many traditional software engineering tasks and newer research areas already rely heavily on the use of data mining techniques, machine learning, statistical analysis, etc. The newer approaches include: search based software engineering (SBSE - http://www.sebase.org/), mining software repositories (MSR - http://www.msrconf.org/), recommendation systems in software engineering (RSSE - http://sites.google.com/site/rsseresearch), predictor models in software engineering (PROMISE - http://promisedata.org/), etc. These areas of research already have dedicated conferences or workshops and their communities are growing. There are many software engineering tasks that are supported with such techniques, used to analyzed software data. These tasks include: defect prediction, effort estimation, impact analysis, bug triage, bug assignment, source code searching,

requirements analysis, traceability link recovery between software artifacts, bug detection, debugging, etc., to name a few.

Software engineering is not the first field to face the challenge of dealing with so much data, as the research and industry develops. For example, the generation and accumulation of massive data in biological and medical research lead to the birth of a new field: bioinformatics. We are facing a similar paradigm shift in software engineering, where more and more tasks are reinterpreted as optimization or search problems, which require data mining-based solutions. Bioinformatics blends together research in data mining, machine learning and statistical analysis with biology. We can draw the analogy and think of *software informatics* as a new research area that combines research in data mining, machine learning and statistical analysis with software engineering. While we can learn from the history and development of bioinformatics (and other such fields), the challenges in the analysis of software engineering data are unique and will most likely result in different solutions and research agenda.

A recent cover article on data mining and machine learning in *IEEE Computer* features software engineering as one of the four major application areas . The conclusion of the article on data mining in software engineering is that "much work needs to be done to further adapt general-purpose mining algorithms or develop specific algorithms to satisfy the unique requirements of software engineering data and tasks" .

Software engineering applications of data mining face unique challenges compared to other fields, where data mining is used every day (e.g., medicine, bioinformatics, business, finances, security, etc.), as in software engineering the data is extremely heterogeneous. For example, a data mining algorithm applied on genetic information from one species can be easily used, with the same configuration, on genetic information of other species. The data is different, but the underlying structure is very similar. On the other hand, in software engineering no two real-world software systems are that much alike, even within the same application domain. What we learn about one system and one process may not apply to another. Software engineering tasks demand the mining of multiple correlated data types, including both sequence and text data, or text data and graph data, used together to achieve the best results. Data mining is often performed off-line, but some time is required just-in-time.

Last but not least, software engineers are no experts in data mining and data mining researchers are no experts in software engineering data. This poses problems when using off the shelf solutions for software engineering problems.

A common and critical problem with data mining solutions to software engineering tasks (in their current form) is that most algorithms use a variety of parameters, which are domain and data specific. Reusing a set of parameter values from one application to another usually leads to poor results. For example, Bayes classifiers use the "M" and Laplace factors to handle low frequency counts. Standard default values are M=2 and L=1, but there is no telling what is the best configuration for a particular domain. These standard parameter values are usually derived empirically from applications in domains outside software engineering. For example, default parameter values used by many information retrieval techniques are established based on data from text retrieval tasks on natural language corpora. Recent empirical research in applications of text retrieval in software engineering showed that best results are in fact obtained for configurations different from those used in natural language retrieval applications. More than that, the quality of the results is highly dependent on these configurations . The observation also holds for many defect prediction approaches . For most data mining algorithms, the space of possible parameter values is extremely large. Finding the right combination is often based on intuition rather than on science. Without a well defined methodology that allows all these parameters to be effectively customized for applications in software engineering, the research in the field is limited to proof-of-concept efforts and generalization of the results is rarely achieved or ensured. This customization problem is one of the main reasons that stop many data mining solutions to software engineering problems from migrating from the research labs to industry. To make things worse, algorithms that work on data from a software system, for a particular task (let's say defect prediction) may not work well on data from another software system, for the same task.

New research will have to address this generic algorithm customization problem, while also creating domain (i.e., software engineering) specific algorithms.

Data selection is an equally important problem that has to be in the focus of future research. What data should be mined and analyzed in support of a specific software engineering task? Should we mine the same type of data in each case (i.e., software systems)? These are questions that only future research can answer.

The heterogeneous nature of the data generated and collected in a software system raises the need for the use of data and information integration techniques. Much of the software data is unstructured (i.e., natural language), while some is stored in structured formats. Once again, we can borrow techniques from another field (i.e., enterprise information integration and the theory of data integration) and adapt them to the software engineering context.

3. NEW APPROACH FOR EMPIRICAL RESEARCH

The research we are discussing here is empirical in nature. In order to ensure its success and the translation of its results into practice and education, empirical and external validation is needed. We need to adopt new approaches to empirical research that will allow researchers to share data and lessons learned from the data. Software engineering is about created constructs (i.e., software) and not about immutable structures in the hard world. Software changes much, much faster than, let's say, the genome of a species. Hence, we need tools for checking, and constantly rechecking, the lessons-learned that are being applied to a particular data.

Once again, we can see how other similar field developed. One of the core areas of the bioinformatics discipline is the design, development, and long term maintenance of biological databases, which are shared by the community. There are many types of such data bases, such as: primary sequence databases, genome databases, protein sequence and protein structure databases, protein-protein databases, signaling pathway databases, metabolic pathway databases, microarray databases, etc.

We have to adopt a similar core area and create such databases with software analysis data. One can argue that the open source movement generates and shares data that researchers can use. That is certainly true. However, if we follow the analogy with bioinformatics, the data in a software repository is more akin to the genome of a few species. It is the raw data, not the results of the analysis. Only if we store and share, as a community, the results of various analysis techniques on various data, we will be able to learn from it, repeat and validate the experiments, and improve our solutions. A small step in this direction is the PROMISE repository (http://promisedata.org). As of today, it contains about 100 data sets, contributed by a few dozen researchers over the past five years. While we like to think of it as a success story, the reality is that it is a mere indication of how far we still have to go.

Having the data is not enough. We also need to change how research is practiced, published, and evaluated. We have to try new algorithms on old data, repeat experiments on data from multiple projects, challenge or re-assess prior results, etc.

4. CONCLUSION

The body of work ahead of us is considerable. There are many research questions that we need to address, such as:

- Which software engineering tasks can benefit from data analysis and data mining support?
- What data should we analyze and mine?
- How to integrate heterogeneous data?
- What algorithms should we use for analysis and information extraction?
- How to customize these algorithms?
- Can we use these algorithms across systems?
- How to store and share analysis data?
- How can we integrate data analysis and data mining into existing software engineering processes and curriculum?

Think of how much data was generated in a large active software project in the time it took to write, review, and present this position paper. None of us would be able to deal with it as is – we need the assistance of automated data extraction and analysis tools. Data mining, information retrieval, machine learning, and statistical analysis applied to software engineering data should become a mainstream research area in our field. The results of such research should find their way in new software engineering processes and in the education of future software engineers. In order to achieve that we need to adopt new ways of empirical research based on shared open data and replication.

5. ACKNOWLEDGMENTS

This work is supported in part by the US National Science Foundation through grant number CCF-1017263.

6. REFERENCES

[1] De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G. 2007. Recovering Traceability Links in Software Artefact Management Systems. *ACM Transactions on Software Engineering and Methodology* 16, 4 (September 2007) 13. DOI= http://doi.acm.org/10.1145/1276933.1276934

[2] Gay, G., Haiduc, S., Marcus, A., and Menzies, T. 2009. On the Use of Relevance Feedback in IR-Based Concept Location, In *Proceedings of IEEE International Conference on Software Maintenance* (Edmonton, AB, Canada, September 20-September 26, 2009), 351-360.

[3] Halevy, A. Y., Ashish, N., Bitton, D., Carey, M., Draper, D., Pollock, J., Rosenthal, A., and Sikka, V. 2005. Enterprise information integration: successes, challenges and controversies. In *Proceedings of the 2005 ACM SIGMOD international Conference on Management of Data* (Baltimore, Maryland, June 14 - 16, 2005). SIGMOD '05. ACM, New York, NY, 778-787. DOI= http://doi.acm.org/10.1145/1066157.1066246.

[4] Hayes, J. H., Dekhtyar, A., and Sundaram, S. K. 2006. Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. *IEEE Trans. Softw. Eng.* 32, 1 (Jan. 2006), 4-19. DOI= http://dx.doi.org/10.1109/TSE.2006.3

[5] Lenzerini, M. 2002. Data integration: a theoretical perspective. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Madison, Wisconsin, June 03 - 05, 2002). PODS '02. ACM, New York, NY, 233-246. DOI= http://doi.acm.org/10.1145/543613.543644

[6] Marcus, A., Maletic, J. I., and Sergeyev, A. 2005. Recovery of Traceability Links Between Software Documentation and Source Code. *International Journal of Software Engineering and Knowledge Engineering* 15, 5 (2005), 811-836.

[7] Menzies, T., Greenwald, J., and Frank, A. 2007. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Trans. Softw. Eng.* 33, 1 (Jan. 2007), 2-13. DOI= http://dx.doi.org/10.1109/TSE.2007.10

[8] Poshyvanyk, D., Gueheneuc, Y., Marcus, A., Antoniol, G., and Rajlich, V. 2007. Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval. *IEEE Trans. Softw. Eng*. 33, 6 (Jun. 2007), 420-432. DOI= http://dx.doi.org/10.1109/TSE.2007.1016

[9] Ramakrishnan, N. 2009. The Pervasiveness of Data Mining asnd Machine Learning. *Computer* 42, 8 (Aug. 2009), 28-29. DOI= http://dx.doi.org/10.1109/MC.2009.268

[10] Xie, T., Thummalapenta, S., Lo, D., and Liu, C. 2009. Data Mining for Software Engineering. *Computer* 42, 8 (Aug. 2009), 55-62. DOI= http://dx.doi.org/10.1109/MC.2009.256

[11] Yang, Y. and Webb, G. I. 2003. Weighted proportional k-interval discretization for naive-Bayes classifiers. In *Proceedings of the 7th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining* (Seoul, Korea, April 30 - May 02, 2003). K. Wang, J. Jeon, K. Shim, and J. Srivastava, Eds. Lecture Notes In Artificial Intelligence. Springer-Verlag, Berlin, Heidelberg, 501-512.