Software Intelligence: The Future of Mining Software Engineering Data

Ahmed E. Hassan
School of Computing
Queen's University
Kingston, ON, Canada
ahmed@cs.queensu.ca

Tao Xie
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
xie@csc.ncsu.edu

ABSTRACT

Mining software engineering data has emerged as a successful research direction over the past decade. In this position paper, we advocate Software Intelligence (SI) as the future of mining software engineering data, within modern software engineering research, practice, and education. We coin the name SI as an inspiration from the Business Intelligence (BI) field, which offers concepts and techniques to improve business decision making by using fact-based support systems. Similarly, SI offers software practitioners (not just developers) up-to-date and pertinent information to support their daily decision-making processes. SI should support decision-making processes throughout the lifetime of a software system not just during its development phase.

The vision of SI has yet to become a reality that would enable software engineering research to have a strong impact on modern software practice. Nevertheless, recent advances in the Mining Software Repositories (MSR) field show great promise and provide strong support for realizing SI in the near future. This position paper summarizes the state of practice and research of SI, and lays out future research directions for mining software engineering data to enable SI.

Categories and Subject Descriptors

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

General Terms

Documentation, Economics, Experimentation, Human Factors, Management, Measurement, Reliability, Verification

Keywords

Software intelligence, mining software engineering data, mining software repositories

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

1. INTRODUCTION

Much of software practice centers around daily decisions and questions (e.g., when to release a software system? which parts of a software system to change? which parts of a software system to test? who is using this feature? and who knows about this feature?). Unfortunately, nowadays many decisions related to a software system are based on intuition and gut feeling. Determining when a software system is ready for release, whether a part of a software system should be re-factored or re-written, or which parts of a software system should be thoroughly tested is a matter of art instead of a well-studied science. These haphazard decision processes lead to wasted resources and increased cost of building and maintaining large complex software systems.

Software practitioners are in dire need of what we propose to call Software Intelligence (SI). While Business Intelligence (BI) [27] offers concepts and techniques to improve business decision making by using fact-based support systems, SI offers software practitioners (not just developers) up-to-date and pertinent information to support their daily decision-making processes. SI provides practitioners with access to specialized fact-supported views of their software system so they can answer critical questions about it. Using SI, owners, maintainers, and developers of software systems can perform long-term and short-term informed strategic planning. Moreover, SI gives companies a better understanding of the true potential and actual limitation of their software assets.

Mining software engineering data has emerged as a research direction over the past decade. This research direction has already achieved substantial success in both research and practice. In this position paper, we advocate Software Intelligence (SI) as the future of mining software engineering data, within modern software engineering research, practice, and education.

The vision of SI has yet to become a reality. Nevertheless, recent advances in the Mining Software Repositories (MSR) field show great promise and provide strong support for realizing SI in the near future, as software engineering research aims to ensure its relevance and impact on modern software practice. This position paper summarizes state of practice and research of SI, and lays out future research directions of mining software engineering data to enable SI.

2. STATE OF PRACTICE

Prior experiences and dominant patterns are the driving force for many decision-making processes in modern software organizations. Software practitioners often rely on their experience, intuition, and gut feeling in making important decisions. Managers allocate development and testing resources based on their experience in previous projects and their intuition about the complexity of the new project relative to prior projects. Developers commonly

Software Engineering Data	Mining Algorithms	Software Engineering Tasks
Sequences: execution/static traces, co-changes, etc.	association rule mining, frequent itemset/subseq/ partial-order mining, seq matching/clustering/classification, etc.	programming, maintenance, bug detection, debugging, etc.
Graphs: dynamic/static call graphs, program dependence graphs, etc.	frequent subgraph mining, graph matching/clustering/ classification, etc.	bug detection, debugging, etc.
Text: bug reports, emails, code comments, documentations, etc.	text matching/clustering/ classification, etc.	maintenance, bug detection, debugging, etc.

Figure 1: Software engineering data, mining algorithms, and software engineering tasks [31]

use their experience when adding a new feature or fixing a bug. Testers usually prioritize the testing of features that are known to be error-prone based on field and bug reports.

The state of SI in practice is very rudimentary with many of the decisions being supported by gut feeling and at best through consultation with senior developers. However, access to such developers is limited and the access continues to decrease as systems age and as developers move across companies. In addition, recent efforts to document information are very limited in practice. Primarily non-specialized wikis are used as knowledge repositories and decisions are often made through the support of spreadsheets and slides.

3. STATE OF RESEARCH

Mining software engineering data has emerged as a research direction over the past decade. This research direction achieved substantial success in both research and practice. The Mining Software Repositories (MSR) [16, 14, 17, 19, 29, 31] field is an example of such a research direction. The MSR field analyzes and cross-links the rich data available in software repositories to uncover interesting and actionable information about software systems and projects. Below are examples of software repositories:

Historical repositories such as source control repositories, bug repositories, and archived communications record information about the evolution and progress of a project.

Run-time repositories such as deployment logs contain information about the execution and the usage of a software system at a single or multiple deployment sites.

Code repositories such as Sourceforge.net, Google code, and Codeplex.com contain the source code of various software systems developed by a team of developers.

Software repositories contain a wealth of valuable information about software projects. Using the information stored in these repositories, software practitioners can depend less on their intuition and experience, and depend more on historical and field data. Historical repositories capture important historical dependencies [13] between project artifacts, such as functions, documentation files, and configuration files. Developers can use this information to propagate changes to related artifacts, instead of using only static or dynamic code dependencies, which may fail to capture important dependencies. For example, a change to the code that writes data to a file may require changes to the code that reads data from the file, although there exist no traditional dependencies (e.g., data and control flow) between both pieces of code. Run-time repositories could be used to pinpoint execution anomaly by identifying dominant execution or usage patterns across deployments, and flagging deviations from these patterns (e.g., [18]). Code repositories could be used to identify dominant and correct framework or library API usage patterns by mining the API usage of a framework or library across many projects (e.g., [24]).

While software repositories are often used in practice as record-keeping repositories, they are rarely used to support decision-making processes. For example, historical repositories are used to track the history of a bug or a feature, but are not commonly used to determine the expected resolution time of an open bug based on the resolution time of previously-closed bugs.

The MSR field is one of the most promising fields in supporting and enabling widespread adoption of SI. By transforming these repositories from static record-keeping repositories into active ones, we can guide decision-making processes in modern software projects. For example, data in source control repositories, traditionally used to archive code, could be linked with data in bug repositories to help practitioners propagate complex changes and to warn them about risky code based on prior changes and bugs.

The MSR field is maturing thanks to the rich, extensive, and readily available software repositories. Table 1 lists the descriptions of several examples of software repositories that could be mined. Figure 1 shows example software engineering data being mined (the first column), example software engineering tasks (the last column) assisted by applying various mining algorithms (the middle column) on each type of software engineering data listed in the first column [31].

4. ENABLING SOFTWARE INTELLIGENCE (SI)

We next highlight areas requiring the attention of MSR researchers and software engineering researchers in general so we can ensure that the MSR field can fully contribute towards the full development of SI. For each area, we briefly mention its current state and promising new directions that we believe hold great promise for that particular area.

4.1 SI Throughout the Lifecycle of a Project

Current State. Previous analysis [16] of the publications at the MSR working conference and workshop [21] from 2004 to 2008 shows that a high percentage (∼80%) of the published papers focus on source code and bug-related repositories. Part of the reasons could be that the used bug repositories or source control repositories are commonly available and the source code and bug reports are well structured, facilitating automated data analysis and processing. The analysis of the MSR publications also reveals that documentation repositories (e.g., requirements) are rarely studied, likely due to their limited availability. In summary, the past MSR publications heavily mined source code and bug-related repositories, often with strong emphasis in assisting software engineering tasks in the coding phase of a project's lifecycle, benefiting primarily developers.

Future Directions. To enable SI, future MSR work should look beyond the coding phase as this phase represents a small portion of the lifecycle of a project. Managers, testers, deployers, and support teams are all stakeholders of a software system and they all need SI support from the software engineering community. The overly heavy focus on developers in past MSR work is not healthy and is limiting the impact of SI on the whole software industry. In addition, the MSR results and innovations should be integrated into

Repository	Description
Source control repositories	These repositories record the development history of a project. They track all the changes to the source code along with the meta-data associated with each change, e.g., the name of the developer who performed the change, the time the change was performed and a short message describing the change. Source control repositories are the most commonly available and used repository in software projects. CVS, subversion, Perforce, ClearCase, and Git are examples of source control repositories used in practice.
Bug repositories	These repositories track the resolution history of bug reports or feature requests that are reported by users and developers of large software projects. Bugzilla and Jira are examples of bug repositories.
Archived communications	These repositories track discussions about various aspects of a software project throughout its lifetime. Mailing lists, emails, IRC chats, and instant messages are examples of archived communications about a project.
Deployment logs	These repositories record information about the execution of a single deployment of a software system or different deployments of the same systems. For example, the deployment logs may record the error messages reported by a software system at various deployment sites. The availability of deployment logs continues to increase at a rapid rate due to their use for remote issue resolution (e.g., remote crash uploading tools), and due to recent legal acts. For instance, the Sarbanes-Oxley Act of 2002 [6] stipulates that the execution of telecommunication and financial systems must be logged.
Code repositories	These repositories archive the source code for a large number of projects. Sourceforge.net and Google code are examples of large code repositories.

Table 1: Examples of software repositories

stakeholders' daily working environments, including but not limited to Integrated Development Environments (IDEs).

SI is more than just helping with coding.

4.2 SI Using Non-Historical Repositories

Current State. The MSR field started out with a strong focus on historical repositories such as source control and bug repositories. Therefore, there seems to be a misconception that MSR is all about historical data sources (or repositories). Such a misconception needs to be addressed to help SI achieve its full potential. In our view, MSR and mining software engineering data are synonyms: MSR is about mining any type of software engineering data (e.g., execution logs [18], code snippets scattered throughout the Internet [23, 24, 20], and API documents [32]), even when these data are not stored in an explicit "repository".

Future Directions. To enable SI, future MSR work should look beyond the traditional types of software engineering data stored in repositories. Some emerging promising types of data could include developer interaction data with tools within IDEs, developer-meeting notes (even voice recording and recognition with advances in natural/spoken language processing), recordings of support calls, and online posting about software products. These types of data could be of the nature of real-time stream data, which may not be stored in repositories, due to large volume or privacy concerns. Indeed, privacy issues need to be taken care throughout software engineering research, as more and more relatively private data are becoming available for analysis and mining.

In addition, special attention is needed in research and practice in improving the collection of data. The existing mechanisms of data collection rely heavily on a large number of heuristics – leading to possibly noisy data. Future MSR work should make proactive suggestions and influences on improving the repository or IDE design [10] to ease the collection of data. Some modern IDEs such as IBM Jazz [3] and Microsoft Visual Studio Team Foundation Server [4] are leading examples in the right direction (e.g., allowing explicit traceability across artifacts to be accurately specified rather than being mined from noisy data). However, a great amount of work is needed towards creating higher-quality data for mining. As the SI field matures, we envision the creation of new roles who are focused on maintaining and curating the various types of repositories about software projects. These curators would ensure that high quality data is stored in these repositories and that they are preserved over the years.

Finally, new opportunities could be exploited by mining multiple sources of software engineering data at the same time, even among heterogenous data such as textual data from bug reports and execution data from their associated failing tests (e.g., [28, 22]).

SI should leverage all types of repositories not just historical ones. .

4.3 SI Use of Effective Mining Techniques

Current State. Past MSR work heavily exploited basic off-the-shelf data mining (DM) algorithms (such as association rule mining and frequent itemset mining [15]) or tools (such as Weka [8]). When MSR researchers explored and applied mining techniques on software engineering data, they commonly compromised their mining requirements to overfit what these basic off-the-shelf algorithms or tools could provide.

Future Directions. To enable SI, future MSR work should follow a problem-driven methodology in advancing the field: (1) empirically investigate problems in the software engineering domain, (2) identify mining requirements for addressing those problems, (3) adopt or adapt advanced mining algorithms [9] from the DM community, or develop new mining algorithms [26, 25] for satisfying the mining requirements. Indeed, inventing new mining algorithms for MSR needs could be challenging for software engineering researchers. One possible solution is to collaborate with DM researchers. Another possible solution is to adapt or integrate existing mining algorithms by conducting preprocessing of input data or postprocessing of mined patterns.

SI and DM fields should work closer.

4.4 SI Adoption in Practice

Current Practice. Successful products from Coverity [1] and Pattern Insight [5] already integrate ideas and innovations based on mining software engineering data. These products are used by

practitioners worldwide. Given the dependence of SI on already-available repositories (e.g., historical changes, code or execution logs), the barrier and cost for experimentation with SI innovations are considerably low compared to other software engineering innovations and techniques (e.g., extreme programming or agile development). In short, if your company has a repository, you can mine it with minimal effort.

Future Directions. To enable the widespread adoption of SI, we must first consider the level at which SI support is being provided. For example, SI could help practitioners decide on small-level issues (e.g., review a particular change) or large-scale issues (e.g., re-design a particular portion of a software system). The lower and more focused the challenges or questions that an SI technique provides, the more likely it will be adopted. The less commitment and approvals that are needed throughout the management chain in an organization, the more likely the SI recommendation will be followed (e.g., review this change that you just made versus re-design this component).

Second, we need to make sure that SI techniques are intuitive and the SI results are easy to explain and describe. Explainability and intuitiveness are key, even over higher performance since buy-in throughout a software organization is often a major hurdle: no one would be willing to run their business based on a magic box. While some mining techniques [15] provide mining results with high explainability, an organization will still need to feel confident and at ease with the explanations. In addition, effective tool support such as visualization in understanding the mining results and the data being mined will help in better communicating SI results throughout an organization from developers to managers.

SI should help explain but will never replace practitioners.

5. DISCUSSION AND CONCLUSION

More thoughts are needed to exploit the achievements and lessons learned in the BI field. In many ways, SI is BI for software companies. We should explore whether we can sell software engineering decision making as part of traditional BI platforms, since software plays critical roles in more and more businesses, and software businesses are just a special type of business. We could explore whether we could exploit and reuse traditional BI platforms. For example, IBM's recent Rational Insight product [2] uses the Cognos BI platform to provide SI to project managers. Building SI on top of BI infrastructures holds great promise in enabling easier adoption since BI infrastructures are much more advanced and polished, and are already adopted in many large organizations.

We see SI as providing support for not just software practitioners but also for software engineering researchers. For example, SI can help support research directions and focuses by enabling automated empirical software engineering. We envision a synergetic feedback loop between data generation/selection (based on mining results) and data mining (based on generated data). Recent software testing work [30, 11] is already exploring this feedback-loop concept, which is also known as active learning [12] in the machine learning community.

SI will play more and more important roles in evaluating research outcomes. Research projects and papers can and should be evaluated relative to their SI abilities, and they must demonstrate true value to practitioners. In the MSR field, several start-up companies (founded by academia based on MSR research), such as Coverity [1], Pattern Insight [5], and Tasktop [7], already demonstrated high promises in providing substantial SI capabilities to industrial practices. We expect more such technology transfers and more SI success stories.

While we propose a strong emphasis on practical utilities and applications, SI should not and will not suffocate deep and long-term research. For example, SI will provide researchers and practitioners facts and evidence to help them design evolutionary or transformative approaches such as new languages and tools, and decide on adopting them or not using facts instead of intuition and gut feeling. We envision that SI will be an enabling platform for various types of research throughout software engineering.

Acknowledgments

Ahmed E. Hassan is the NSERC RIM Industrial Chair in Software Engineering. Tao Xie's work is supported in part by NSF grants CNS-0716579, CCF-0725190, CCF-0845272, CCF-0915400, CNS-0958235, an NCSU CACC grant, ARO grant W911NF-08-1-0443, and ARO grant W911NF-08-1-0105 managed by NCSU SOSI.

6. REFERENCES

[1] Coverity, Inc. `http://www.coverity.com/`.

[2] IBM Rational Insight. `http://www-01.ibm.com/software/rational/products/insight/`.

[3] IBM Rational Jazz. `http://www-01.ibm.com/software/rational/jazz/`.

[4] Microsoft Visual Studio Team Foundation Server. `http://msdn.microsoft.com/en-us/vstudio/`.

[5] Pattern Insight Inc. `http://patterninsight.com/`.

[6] Summary of Sarbanes-Oxley Act of 2002. `http://www.soxlaw.com/`.

[7] Tasktop Technologies Inc. `http://tasktop.com/`.

[8] Weka. `http://www.cs.waikato.ac.nz/ml/weka/`.

[9] M. Acharya, T. Xie, J. Pei, and J. Xu. Mining API patterns as partial orders from source code: From usage scenarios to specifications. In *Proceedings of 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007)*, pages 25–34, September 2007.

[10] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu. The promises and perils of mining git. In *Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories (MSR 2009)*, pages 1–10, May 2009.

[11] J. F. Bowring, J. M. Rehg, and M. J. Harrold. Active learning for automatic classification of software behavior. In *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004)*, pages 195–205, July 2004.

[12] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[13] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the 14th International Conference on Software Maintenance (ICSM 1998)*, pages 190–198, November 1998.

[14] M. W. Godfrey, A. E. Hassan, J. Herbsleb, G. C. Murphy, M. Robillard, P. Devanbu, A. Mockus, D. E. Perry, and D. Notkin. Future of mining software archives: A roundtable. *IEEE Software*, 26(1):67–70, 2009.

[15] J. Han and M. Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann Publishers Inc., 2000.

[16] A. E. Hassan. The road ahead for mining software repositories. In *Proceedings of the Future of Software Maintenance at the 24th IEEE International Conference on Software Maintenance (ICSM 2008)*, pages 48–57, September-October 2008.

[17] A. E. Hassan and T. Xie. Mining software engineering data. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*, pages 503–504, May 2010.

[18] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann. Automatic identification of load testing problems. In *Proceedings of the 24th International Conference on Software Maintenance (ICSM 2008)*, pages 307–316, 2008.

[19] H. H. Kagdi, M. L. Collard, and J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance*, 19(2):77–131, 2007.

[20] M. R. Marri, S. Thummalapenta, and T. Xie. Improving software quality via code searching and mining. In *Proceedings of the 1st International Workshop on Search-Driven Development – Users, Infrastructure, Tools and Evaluation (SUITE 2009)*, pages 33–36, May 2009.

[21] The Working Conference on Mining Software Repositories. Available online at `http://www.msrconf.org`.

[22] D. Thakkar, Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora. Retrieving relevant reports from a customer engagement repository. In *Proceedings of the 24th International Conference on Software Maintenance (ICSM 2008)*, pages 117–126, September 2008.

[23] S. Thummalapenta and T. Xie. PARSEWeb: A programmer assistant for reusing open source code on the web. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, pages 204–213, November 2007.

[24] S. Thummalapenta and T. Xie. SpotWeb: Detecting framework hotspots and coldspots via mining open source code on the web. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008)*, pages 327–336, September 2008.

[25] S. Thummalapenta and T. Xie. Alattin: Mining alternative patterns for detecting neglected conditions. In *Proceedings of 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*, pages 283–294, November 2009.

[26] S. Thummalapenta and T. Xie. Mining exception-handling rules as sequence association rules. In *Proceedings of 31th International Conference on Software Engineering (ICSE 2009)*, pages 496–506, May 2009.

[27] E. Vitt, M. Luckevich, and S. Misner. *Business Intelligence: Making Better Decisions Faster*. Microsoft Press, 2002.

[28] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th International Conference on Software Engineering (ICSE 2008)*, pages 461–470, May 2008.

[29] T. Xie. Bibliography on mining software engineering data. Available online at `https://sites.google.com/site/asergrp/dmse`.

[30] T. Xie and D. Notkin. Mutually enhancing test generation and specification inference. In *Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES 2003)*, pages 60–69, October 2003.

[31] T. Xie, S. Thummalapenta, D. Lo, and C. Liu. Data mining for software engineering. *IEEE Computer*, 42(8):35–42, August 2009.

[32] H. Zhong, L. Zhang, T. Xie, and H. Mei. Inferring resource specifications from natural language API documentation. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*, pages 307–318, November 2009.