Towards mining norms in open source software repositories

Bastin Tony Roy Savarimuthu[1] and Hoa Khanh Dam[2]

[1] Department of Information Science, University of Otago, Dunedin, New Zealand
[2] School of Computer Science and Software Engineering, University of Wollongong, New South Wales 2522, Australia
tonyr@infoscience.otago.ac.nz, hoa@uow.edu.au

Abstract. The concept of norms has attracted a lot of interest in various disciplines including computer science since it facilitates collaboration and cooperation of individuals in societies. Extracting norms from computer-mediated human interactions is gaining popularity since huge volume of data is available from which norms can be extracted or "mined". The emerging open source communities offer exciting new application opportunities for norms mining since such communities involve collaboration and cooperation among developers from different geographical regions, background and cultures. Mining norms from open source projects however has not received much attention from the normative multi-agent system community. Therefore, our position paper addresses this issue by discussing the opportunities and the challenges presented by this domain for the study of norms. It provides a brief description of existing technologies in mining software repositories (MSR) that can be leveraged. In addition, it highlights the motivations for the study of normative behaviour in open source software development from the data available in various software repositories. On this basis, it lays out the main research questions and open challenges in mining norms from these repositories.

1 Mining software repositories

Mining Software Repositories (MSR) [4] is an emerging software engineering research area that focuses on mining the rich and large quantities of data available in software repositories to uncover useful and important patterns and information about software systems and projects. Such information assists developers, managers, testers, etc. working on those systems by offering insights and predictions about the future of software systems. Software repositories can be in various forms such as historical repositories (e.g. SVN or CVS repositories, archived communication records, bug repositories), code repositories (e.g. Sourceforge.net or Google code), and run-time repositories (e.g. deployment and/or execution logs). Recently, mining software repositories has generated a lot of interest due to the widespread adoption and growth of the open source software (OSS) projects. The field of MSR is producing techniques and tools for offering insights into the

nature of open source software development (OSSD). In fact, large-scale open source systems such as Linux or Android OS used by millions of users and developed by hundreds of developers over extended period of time have produced rich, extensive, and easy-to-access software repositories.

Efforts in MSR[3] research have been mainly on providing techniques to extract and cross-link important information from different software repositories. Such information can be used in various activities during the development of software systems. For instance, a range of work (e.g. [11]) have proposed to make use of historical co-changes (e.g. entities or artefacts that were changed together) in a software project to propagate changes across the software system during maintenance and evolution. A large number of existing MSR work (e.g. [6]) also mine bug reports and historical changes to predict the occurrence of bugs, e.g. parts of the code that likely to contain errors. Such predictions are useful for managers in allocating and prioritizing testing resources. Information mined from reported bugs and execution logs can also be used to improve the user experience, e.g. warning the user when they are about to perform a buggy action, and suggesting when an existing piece of code can be re-used. Empirical software engineering also substantially benefits from MSR since many empirical studies can be done (and repeated) on a large number of subjects, i.e. OSS repositories enable the verification of generality of prior findings (e.g. the study in [5] confirming that cloning seems to be a reasonable or even beneficial design option in some situations). Recent MSR work (e.g. [1]) has also attempted to mine discussions from archived mailing lists, forums, and instant messaging to understand the dynamics of large software development teams, including how and when team members get invited, detecting team morale at a particular point in time, and understanding the process of bug triage.

Traditional commercial software projects have a carefully structured and hierarchical team organization with norms and policies that are explicitly stated and enforced. In contrast, OSS projects have no pre-designed organizational structure but they are large and successful. Some recent work (e.g. [2]) have tried to understand the social structure in OSS projects in which how dynamic self-organizing subgroups spontaneously form and evolve. *However, to the best of our knowledge, there has been no existing work in mining norms in OSS projects.*

OSS projects present an interesting context for norm mining for several reasons. Firstly, there are a substantial number of OSS projects in various sizes (ranging from a few developers to hundreds of contributors), different coding cultures (e.g. Java vs. C), or different application domains (browsers vs. operating systems). These projects would allow us to understand how norms emerge and how they are enforced in different settings. Secondly, OSS projects tend to involve communication and coordination of contributors from different background, cultures and geographical regions, which makes OSS an exciting domain for exploring how norms affect the success or failure of a particular software project or community. Thirdly, such rich, extensive and readily available data

[3] A extensive review of the work in MSR can be found from the "Bibliography on Mining Software Engineering Data" available at http://ase.csc.ncsu.edu/dmse

from OSS projects allow us to mine norms from different sources. For instance, we can directly observe developer discussions, identify their contents (e.g. patches, bugs, reviews) on mailing lists or forums. We can build social networks, and cross-check associated discussion and programming activity. In addition, we can leverage existing MSR technologies such as data pre-processing, cross-linking data from different sources for mining norms.

2 Research questions and opportunities

This section discusses the research opportunities for *norms* researchers in the area of mining software repositories. It also points to how constructs and mechanisms developed by the Normative Multi-Agent Systems (NorMAS) community can be employed to infer different types of norms from software repositories.

2.1 Norm types/classifications

This sub-section provides some answers to the basic question of *what types of norms exist in open source software development communities*. Several research work in NorMAS have treated both conventions and norms under the same umbrella of norms despite the differences between the two. We briefly discuss the distinction between the two examples from OSSD.

Conventions - Conventions of a community are the behavioural regularities that can be observed. Coding standards of a project community is an example of a convention. The specifications of these conventions may be explicitly available from the project websites[4] or can be inferred implicitly (e.g. a wide spread convention that may not be explicitly specified in project websites).

Norms - Norms are conventions that are enforced. A community is said to have a particular norm, if a behaviour is expected of the individual members of the community and there are approvals and disapprovals for norm abidance and violation respectively. There have been several categorizations of norms proposed by researchers (c.f. [7]). We believe that deontic norms - the norms describing prohibitions, obligations and permissions studied by the NorMAS community [10] is an appropriate categorization for investigating different types of norms that may be present in OSSD communities. We believe most norms in software repositories will either be prohibitions or obligations.

Prohibition norms - These norms prohibit members of a project group from performing certain actions. However, when those actions are performed, the members may be subjected to sanctions. For example, the members of an open source project may be prohibited to check-in code that does not compile, and

[4] Refer to http://source.android.com/source/code-style.html for the coding guidelines for Android development.

they may be prohibited to check-in a revised file without providing a comment describing the change that has been made.

Obligation norms - Obligations describe activities that are expected to be performed by the members of a project community. When the members of a community fail to perform those, they may be subjected to sanctions. For example, the members may be expected to follow the coding convention that has been agreed upon. Failure to adhere to this convention may result in the code not being accepted by the repository (e.g. based on automatic checking) or a ticket may be issued by a quality assurance personnel. Another obligation may be that the members should complete a task within a time frame. The failure to do so may result in a warning message to be issued (either automatically or manually) in the first instance.

2.2 Research opportunities for norm identification

In NorMAS, researchers have proposed a life-cycle for norms [7] which broadly consists of five phases namely norm creation, identification, spreading, enforcement and emergence. We believe all these five phases can be studied based on the data available from software repositories.

The high-level questions that can be investigated in the context of mining software repositories for norms are the following.

- What are the modes of norm creation in a project community?
- How are pre-specified norms enforced? What kinds of sanctions exist for norm violations? What is the uptake of a norm in a community (i.e. level of conformance)?
- How can emergent norms be detected? How are these norms spread? What contributes to the acceptance or rejection of these norms in a community?

Below, we offer some initial thoughts on addressing these questions.

Modes of norm creation - There could be two modes for norm creation in a software development community. They are 1) explicitly specified norms which every project member is expected to know (pre-specified norms) and 2) norms that arise due to interactions between agents (emergent norms).

Enforcement of pre-specified norms - In a project, both conventions and norms may exist. Conventions agreed upon by project members can be easily monitored. Examples include coding conventions and conventions of not uploading files that do not compile to a version control system. It should be noted that coding conventions can be checked for compliance by evaluating the code using an automated software program such as CheckStyle[5].

Norms on the other hand are enforced. Enforcement involves the delivery of appropriate sanctions. In the domain of software repositories these sanctions are

[5] http://checkstyle.sourceforge.net/

present in artifacts. For example, a bug report on a module that does not deliver the functional requirements can be viewed as a sanction. Additionally, tickets issued for not resolving a bug completely can also be considered as a sanction. Therefore, sanctions that follow violations act as triggers to infer norms. Frequency of norm violations over time may provide evidence for the uptake of a norm in a society. We note that identifying and categorizing different types of sanctions from different types of artifacts is a challenge since the extraction of sanctions involves natural language processing[6].

Identifying emergent norms - Norms that are not pre-specified but that emerge at run-time will be challenging to identify. We believe that emergent norms can be identified by identifying violations first and then inferring what the norms might be. The machinery proposed for norm identification by Savarimuthu et al. [8, 9] can be used as a starting point to infer prohibition and obligation norms. In their work, prohibition norms are identified by extracting sequence of action (or actions) that could have caused the sanction by using a data mining approach [8]. Sanctions form the starting point for norm identification. In the case of obligation norms, missing event sequence (or sequences) that was responsible for the occurrence of a sanction, is identified [9]. While these work on norm identification can be used as a starting point for the extraction of emergent norms in simple cases, the domain of MSR offers more challenges. For example, correlating or linking different types of documents containing relevant information is required before a sequence of actions can be constructed. For example, an email message may contain the sanction message exchanged between developers A and B. Let us assume that A sanctions B for not adding a valid comment to the latest version of the file he or she uploaded. The problem in extracting the norm in this case is that first the verbose message sent from A to B should be understood as a normative statement which involves natural language processing. Second, a cross-check should be conducted to evaluate whether the normative statement is indeed true (i.e. checking whether the comment entered by B is invalid by investigating the log)[7]. Third, the support for endorsements or

[6] Verbose text may be used in the construction of sanction messages. For example, the messages may involve terms that are well beyond the deontic terms such as 'should not', 'must not', 'ought not' in the case of prohibitions. One way to address this problem is to use existing tools such as WordNet [3] to extract synonyms of terms used in the text to infer deontic terms and also use information retrieval tools that offer data manipulation functions such as cleaning and disambiguating the verbose text in order to extract sanctions. Suitability of tools such as OpenCalais (http://www.opencalais.com) and AlchemyAPI (http://www.alchemyapi.com) for this purpose can be investigated. We believe recognizing sanctions is indeed a huge challenge. At the same time, it presents opportunities such as the construction of normative ontologies that can be used across projects for recognizing sanctions.

[7] In this example only two artifacts, the email message and the log are involved. But in practice, several different types of documents may need to be traversed to find the relevant information.

oppositions to such normative positions need to be evaluated in order to extract this as a norm.

Norms that are identified through this process can then be made available to the project community (e.g. on the project websites) once it has been verified by the project administration team.

2.3 A first step in addressing these issues

A first step towards addressing these issues is to create a framework that can extract both conventions and norms from software repositories. The framework should be equipped with appropriate libraries for a) information retrieval techniques (including natural language processing) in order to identify sanctions b) mining software repositories (e.g. cross-linking different sources) and c) norm extraction (e.g. inferring norms from sequences of events). Additionally, it should be able to track and trace the life-cycle of a norm. For example, it should provide appropriate features to capture the waxing and waning of a norm across different periods of time.

2.4 Other relevant research questions

The following are interesting research questions that can be considered.

- How different are norms in large projects (e.g measured based on total number of members or size of the project in KLOC) than the smaller projects? Are norm violation and enforcement patterns different in these projects?
- What are the relationships between roles of individuals in software development and norms (e.g. contributor vs. reviewer vs. module administrator)?
- Are there cultural differences within members of a project with regards to norms (inter- and intra- project comparisons) since individuals from different cultures may have different norms?
- Is there a difference between norm adoption and compliance between open-source and closed-source projects?

The above mentioned questions may interest both humanities researchers and computer scientists. Synergy between the two is required for addressing these questions. As computer scientists we can employ our expertise in several areas (i.e. information retrieval, MSR and normative multi-agent systems) to help answering these questions.

References

1. C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, pages 137–143, New York, NY, USA, 2006. ACM.

2. C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 24–35, New York, NY, USA, 2008. ACM.
3. C. Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.
4. A. Hassan. The road ahead for mining software repositories. In *The 24th IEEE International Conference on Software Maintenance, Frontiers of Software Maintenance.*, pages 48 –57, October 2008.
5. C. Kapser and M. W. Godfrey. Cloning considered harmful considered harmful. In *Proceedings of the 13th Working Conference on Reverse Engineering*, pages 19–28, Washington, DC, USA, 2006. IEEE Computer Society.
6. N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, ICSE '06, pages 452–461, New York, NY, USA, 2006. ACM.
7. B. T. R. Savarimuthu and S. Cranefield. Norm creation, spreading and emergence: A survey of simulation models of norms in multi-agent systems. *Multiagent and Grid Systems*, 7(1):21–54, 2011.
8. B. T. R. Savarimuthu, S. Cranefield, M. A. Purvis, and M. K. Purvis. Norm identification in multi-agent societies. Discussion Paper 2010/03, Department of Information Science, University of Otago, 2010.
9. B. T. R. Savarimuthu, S. Cranefield, M. A. Purvis, and M. K. Purvis. Obligation norm identification in agent societies. *Journal of Artificial Societies and Social Simulation*, 13(4):3, 2010.
10. R. J. Wieringa and J.-J. C. Meyer. Applications of deontic logic in computer science: a concise overview. In *Deontic logic in computer science: Normative system specification*, pages 17–40. John Wiley & Sons, Inc., New York, USA, 1994.
11. T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 563–572, Washington, DC, USA, 2004. IEEE Computer Society.