Data Mining for Software Process Discovery in Open Source Software Development Communities

Chris Jensen, Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA, USA 92697-3425
{cjensen, wscacchi}@ics.uci.edu

Abstract

Software process discovery has historically been an intensive task, either done through exhaustive empirical studies or in an automated fashion using techniques such as logging and analysis of command shell operations. While empirical studies have been fruitful, data collection has proven to be tedious and time consuming. Existing automated approaches have expedited collection of fine-grained data, but do so at the cost of impinging on the developer's work environment, few of who may be observed. In this paper, we explore techniques for discovering development processes from publicly available open source software development repositories that exploit advances in artificial intelligence. Our goal is to facilitate process discovery in ways that are less cumbersome than empirical techniques and offer a more holistic, task-oriented view of the process than current automated systems provide.

1. Introduction and Beginnings

Software process models represent a networked sequence of activities, object transformations, and events that embody strategies for accomplishing software evolution [10]. Software process discovery seeks to take artifacts of development (e.g. source code, communication transcripts, and so forth), as its input and elicit the networked sequence of events characterizing the tasks that led to their development. This process model may then be used as input to other process engineering techniques such as redesign and re-engineering.

Open source software development (OSSD) communities are a rich opportunity for software process discovery and analysis with the benefit that so much of their process-relevant data is publicly available. Though many researchers have sought non-automated means of software process modeling, often there is so much information that it becomes intractable to subsume unaided, thus motivating the push for tools to assist in process discovery. In our past efforts [6], we have shown the feasibility of automating the discovery of software process models by using manual simulation of how such automated techniques might operate as a basis to substantiate that discovery and modeling of software development processes in large OSSD communities such as Mozilla, Apache, NetBeans, and Eclipse (consisting of tens of thousands of developers continuously contributing software artifacts to the community repository) is both plausible and amenable to automation. In this paper, we explore techniques for searching OSSD Web repositories for process data, relating these data in the form of process events, and assigning them to meaningful orders as a process model in an attempt to reduce the manual effort necessary to discover and model software processes.

We take, as our process meta-model, that of Noll and Scacchi [8]. Software processes are composed of events: relations of agents, tools, resources, and activities organized by control flow structures dictating that sets of events execute in serial, parallel, iteratively, or that one of the set is selectively performed.

It has been shown [6] that OSSD community Web repositories encode process data in terms of the structure of the community repository, its content, and its usage and update patterns. OSSD artifacts vary along these three dimensions over time, and this variance is the source of process events. To effectively discover a software process, we must be able capture these data and their changes. This may be done through combined application of text and link analysis techniques, as described below. We propose the use of text analysis techniques for extracting instances of process meta-model entities from the content of the community repositories, followed by link analysis to assert relationships between the mined entities in the form of process events. Next, we apply usage and update patterns to guide integration of the results of text and link analysis together in the form of a process model (see Figure 1). Finally, we conclude with addressing the knowable validity of discovered software process models and future directions for continuing work.

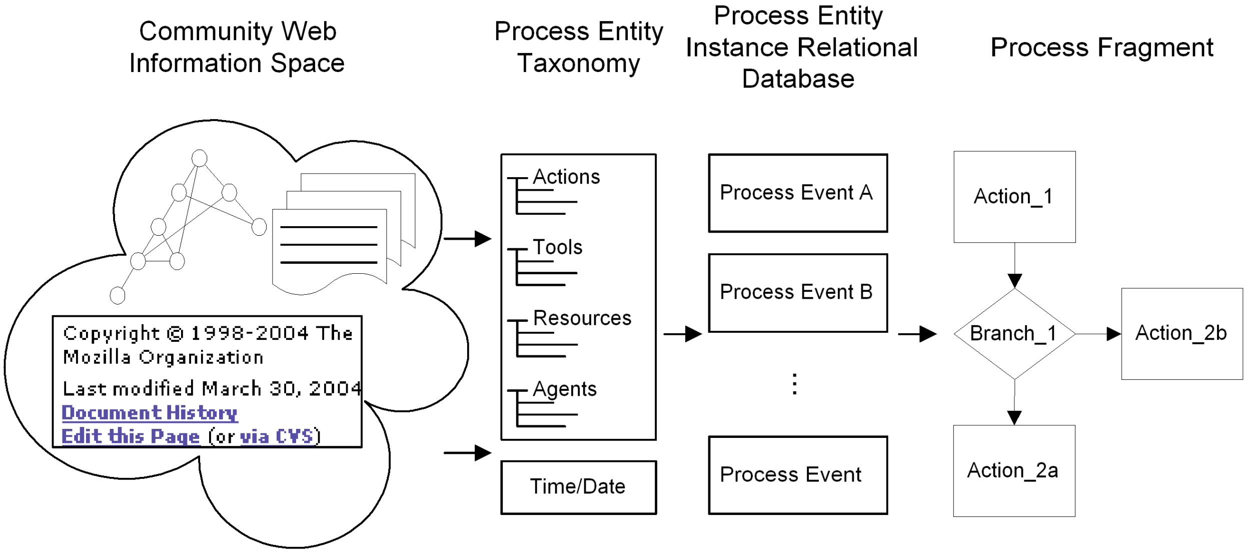
Community Web Information Space
Process Entity Taxonomy
Process Entity Instance Relational Database
Process Fragment
Copyright © 1998-2004 The Mozilla Organization
Last modified March 30, 2004
Document History
Edit this Page (or via CVS)
Actions
Tools
Resources
Agents
Time/Date
Process Event A
Process Event B
Process Event
Action_1
Branch_1
Action_2b
Action_2a

Figure 1: Web artifacts are filtered through a process entity taxonomy to extract atomic process action events, sequenced using temporal indications within the artifacts and reconstructed into a process using PRM

2. Text Analysis

The bulk of the process data is found within the content of Web artifacts. Much of the mapping consists of text extraction, matching between text strings in artifacts such as web pages and email messages and a taxonomy of process related keywords [5]. In the case of web content, we are especially looking for items like date stamps on email messages to place the associated events in time, document authors, and message recipients. This matching is done using a name recognizer.

An inherent challenge to name recognition is that many classes of lexical items we desire to recognize are open sets since we cannot enumerate all possible proper names they contain. Further, name classification suffers from synonymy and polysemy- the same concept represented using different terms, and different concepts represented using the same term, respectively. This frequently occurs between OSSD communities, using terms such as release *manager* rather than release *coordinator* to describe the same role. Fortunately, these are well known problems in text analysis and most text analysis systems provide some support for managing them. The SENSUS ontology system [3] is one such system that attempts to automate much of the domain modeling work allegedly covering most areas of human expertise. This automation is critical considering lexicographical differences across and evolution within communities.

Different types of content yield different opportunities for gathering data. Common to most open source communities are mailing lists and discussion forums, source repositories, community newsletters, issue repositories, and binary release sections, among others. The mere presence of these suggests certain activities in the development process. They also signal what types of data may be contained within. If we just look at source code repositories, we can derive a process specification of a limited set of activities- those that involve changes to the code. Similarly, issue and defect databases tell us that some testing is done on which the issue reports are based. In some communities, issue reports are also used to file feature requests. Such information may also be found within discussion forums or email lists.

Although it may seem tempting to attempt to tailor analysis of artifacts to their type (e.g. email message, defect report, etc) to capitalize on the structure of the artifact type thereby facilitating analysis. While this approach would potentially lead to increased performance in analysis of artifacts conforming to the structure expected by the artifact model, this structure varies widely between communities. To achieve high performance using artifact structure models requires development of models, not only for each artifact type in a community repository, but also for each artifact type used by all repositories under study.

It is interesting to note that we may uncover "how-to" guides or other partial process prescriptions in examining the community repository. Like all content, these may not accurately reflect the process as it is currently enacted, if they ever did. This suggests the need for probabilistic methods for modeling software development processes to filter noise within a process instance and accounting for variance across instances.

By itself, the result of text extraction gives us the raw ingredients of a process model. We look to link analysis to put these ingredients together into atomic process events.

3. Link Analysis

Text extraction allows us to ask questions such as who is collaboration with whom. From this information, we can construct a social network [Madey, et al] for the community. Social networks may identify developers that frequently collaborate, but they do not tell us what the developers are doing, and, more importantly, how they are doing it. One way to associate what and how information is through the use of probabilistic relational modeling (PRM).

Probabilistic relational modeling [4] is somewhat inspired by entity relationship modeling used to describe databases. In the classical example, we might have tables of movie actors, movies, and roles actors have played in movies and want to learn relationships between them. Conceptually, this is no different from linking process agents playing a role to complete an action (using various tools that consume and produce resources). Probabilistic relational modeling allows inference about individual process entities while taking into account the relational structure between them, unlike traditional approaches that assume independence between entities. Why is this the right approach? Software processes driven by the choice of tools used in development. Tools either dictate what and when activities are performed, or tools are selected to support desired activities, and to an extent, suggest methods of completing activities (i.e. enforce process compliance). Developer roles emerge to perform these activities and carry out supplemental work not performed by development tools. Further, process entity instances arising from text analysis have other relationships. They are related contextually to other entities in the artifacts in which they are found. They are also related to artifacts hyperlinked to those in which they are present. Such contextual relationships arising from the logical structure of the repository are also good candidates for probabilistic relational modeling. Indeed, doing so allows us to form process events whose entities span multiple artifacts.

To learn relationships between process entities, we must know the context of the entity with respect to others. This context can be represented in two ways. Extracting the URL of the artifact in which each entity is located allows us to cross-reference that entity with others in the same artifact, as well as other artifacts in which that entity is located. Additionally, if we look at the creation date of the artifact in which it was located, we may be able to intuit that those instances that are temporally distant may signal an activity of lengthy duration multiple instances of the same activity. This determination, however, is the work of usage and update pattern analysis.

4. Usage and Update Patterns

Usage patterns, like content size, are indicators of which areas of the Web space are most active, which reinforces the validity of the data found therein and also claims of what activities in the process may be occurring at a given time. Web access logs, if available, provide a rich source of data. Web page hit counters and last update statistics are also useful for this purpose.

Cadez [1] and Hong, et al [2] demonstrate two techniques for capturing Web navigation patterns, however neither can be done in a strictly noninvasive manner. The first uses server logs and cannot provide tours of the repository and the latter requires members to access the community Web through a proxy server used to track tours. Nevertheless, if we can map tours of the community Web to process events, we can get a sense of which activities are dependent on which other activities, which can be done in parallel, which sequences are done iteratively.

Fortunately, most large OSSD communities use content managing tools to perform versioning of not only product source code, but of other artifacts in the repository, as well. By analyzing changelogs we can learn the frequency of Web updates, in addition to the agent performing the update, and to some extent, the tools used to create the artifact, given its type. Work by Ripoche and Gasser [9] does this to an extent, studying defect resolution status in open source defect repositories. The approach may be generalized, extended with using the text and link analysis techniques given above, and applied to other types of artifacts, though with somewhat less precision due to the inferential nature of process entity relationship construction.

Unfortunately, revision histories are not always available. Since OSSD repositories are publicly accessible, it is possible to spider the Web repository periodically to track changes externally via *diff* tools, though information regarding the precise time of update and author would be lost. As an ethical matter, periodic spidering increases the load on the server that, for large repositories, is potentially burdensome.

By examining usage and update patterns, it is possible for us to detect process control flow structures. If we merely order then by time, the set of process events discovered is sequential. Iterations can be teased out of the sequence by considering patterns of repeated tours and updates of and to the Web. Activities being performed in parallel may also be discerned by examining non-intersecting concurrent usage and update patterns. Further, by analyzing the variance between iterations of the same task, we can identify sets of alternate activities, if the variance is small.

5. Process Model Verification

A critical question of software process discovery, regardless of automation, is how we may discern if the process discovered is a correct reflection of the process enacted by the community. The likelihood of arriving at an accurate model increases with the amount of data examined, within the limitations of the techniques applied. This is because the confidence of an asserted relationship between process entities increases with more positive instances of those relationships. Likewise, weak relationships are rejected due to insufficient evidence. At the same time, relationships between entities cannot be discovered if the entities are not in the list of process-relevant terms we look for during text extraction. Thus, the process model obtained is only as good as the taxonomy.

6. Conclusion

In this paper, we have presented a novel approach to discovering software processes from OSSD Web repositories, combining techniques for text analysis, link analysis, and of repository usage and update patterns. Though we have focused our discussion on open source repositories, given the availability of the artifacts, we believe that these techniques can be applied to closed source software repositories, and given the appropriate domain information, other types of processes, as well. Our hope is that in doing so, we may increase understanding of the process techniques that have led to their success.

7. Acknowledgments

The research described in this report is supported by grants from the National Science Foundation #ITR-0083075 and #ITR-0205679 and #ITR-0205724. No endorsement implied. Contributors to work described in this paper include Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; and Margaret Elliott at the UCI Institute for Software Research.

8. References

[1] Cadez, I.V., Heckerman, D., Meek, C., Smyth, P., and White, S. Visualization of Navigation Patterns on a Web Site Using Model Based Clustering. In Proc. 2000 Knowledge Discovery and Data Mining Conference, 280-284. (2000).

[2] Hong, J. Heer, S. Waterson, and J. Landay, WebQuilt: A proxy-based approach to remote web usability testing, ACM Transactions on Information Systems, 19(3), 263-285. (2001).

[3] Hovy, E.H., A. Philpot, J.-L. Ambite, Y. Arens, J.L. Klavans, W. Bourne, and D. Saroz. 2001. Data Acquisition and Integration in the DGRC's Energy Data Collection Project. In *Proceedings of the dg.o 2001 Conference*. Los Angeles, CA.

[4] Getoor, L., Friedman, N., Koller, D., Taskar B. Learning Probabilistic Models of Link Structure, Journal of Machine Learning Research, 2002.

[5] Jensen, C. Applying a Reference Framework to Open Source Software Process Discovery. In Proceedings of the First Workshop on Open Source in an Industrial Context OOPSLA-OSIC03, Anaheim, CA October 2003.

[6] Jensen, C. and Scacchi W. Simulating an Automated Approach to Discovery and Modeling of Open Source Software Development Processes. In Proceedings of ProSim'03 Workshop on Software Process Simulation and Modeling, Portland, OR May 2003.

[7] Madey, G., Freeh, V., and Tynan, R. "Modeling the F/OSS Community: A Quantitative Investigation," in *Free/Open Source Software Development*, ed., Stephan Koch, Idea Publishing, forthcoming.

[8] Noll, J. and Scacchi, W. Specifying Process Oriented Hypertext for Organizational Computing. *Journal of Network and Computer Applications* 24, (2001). 39-61.

[9] Ripoche, G. and Gasser, L. "Scalable Automatic Extraction of Process Models for Understanding F/OSS Bug Repair", submitted to the 2003 International Conference on Software & Systems Engineering and their Applications (ICSSEA'03), CNAM, Paris, France, December 2003.

[10] Scacchi, W. Process Models in Software Engineering, in J. J. Marciniak (ed.), Encyclopedia of Software Engineering, 2nd. Edition, 2002.