Software Process Recovery: Recovering Process From Artifacts

Abram Hindle
David Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
ahindle@swag.uwaterloo.ca

***Abstract*—Often stakeholders, such as developers, managers, or buyers, want to find out what software development processes are being followed within a software project. Their reasons include: CMM and ISO 9000 compliance, process validation, management, acquisitions, and business intelligence. Recovering the software development processes from an existing project is expensive if one must rely upon manual inspection of artifacts and interviews of developers and their managers. Researchers have suggested live observation and instrumentation of a project to allow for more measurement, but this is costly, invasive, and also requires a live running project. Instead, we propose an after the fact analysis: software process recovery. This approach analyzes version control systems, bug trackers and mailing list archives using a variety of supervised and unsupervised techniques from machine learning, topic analysis, natural language processing and statistics. We can combine all of these methods to recover process events that we map back to software development processes like the Unified Process. We can produce diagrams called Recovered Unified Process Views (RUPV) that are similar to the Unified Process diagram, a time-line of effort per parallel discipline occurring across time. We then validate these methods using case studies of multiple open source software systems.**

I. Introduction

Software Process Recovery is an attempt to recover the underlying software development processes from the evidence hidden within the artifacts that programmers leave behind. Many stakeholders care about the recovery of software development processes from a project. Managers care about what their developers are doing and if the developers are following a prescribed process. Employers care if they can document these development processes in order to achieve certifications like ISO 9000 [1] or a Capability Maturity Model (CMM) rating [2].Those involved in the acquisition of a company may wish to see how the product was made. Developers often seek an awareness of the project they are working on and its software development processes.

Our goal is to recover software development processes from the software development artifacts that programmers leave behind. We want to avoid modifying or instrumenting the current existing practice, we want to rely upon the records that are left behind rather than rely on the perceptions of developers gleaned from interviews.

This kind of historical analysis of software development serves multiple purposes: to recover and understand development processes that occurred within a software project; to reconcile the prescribed software development processes with the observed software development processes; to elicit process related information and developer behaviour from objective artifacts without interviewing developers or relying on their perception or judgement.

Thus we ask: is software process recovery feasible? Can we recover the underlying software development processes of a project by analyzing the artifacts that are left behind in an automatic and semi-automatic manner?

We believe that there are many avenues for recovering software development processes and that repositories such as version control systems, mailing-list archives and bug trackers are good places to start. These repositories are full of interesting documents that are rich with time sensitive data, source code and natural language based artifacts. Thus we hypothesize that we can recover some software development process information from these repositories by utilizing research from the field of *Mining Software Repositories* (MSR).

II. Previous Work

Our work relies heavily on the field of *mining software repositories* (MSR) [3]. MSR is dedicated to exploiting and understanding these artifacts of development and inferring the relationships between them. *Software process recovery* is a sub-field of MSR but closely related to *process mining* [4], the extraction of business processes, and *process discovery* [5],the application of process mining to software development. Software process recovery effectively combines MSR work with process mining and process discovery.

Much MSR research is based upon studying large samples of free/libre open-source software (FLOSS) projects. Capiluppi et al. studied and measured the general characteristics of FLOSS projects [6]. Herraiz et al. [7] and Mockus et al. [8] took a more statistical approach, they extracted metrics and summary statistics on huge collections of FLOSS projects. Israel Herraiz et al. [9] applied time-series analysis using ARIMA to model the number of changes over time.

Much development is about requirements and features. Sometimes features or concepts can be automatically identified from software artifacts. Research on formal concept analysis and concept location [10], [11] has often utilized LSI or semantic clustering [12],while topic analysis has used both LDA and LSI [13], [14], [11]. For instance, Lukins et al. [14],

used LDA for bug localization. Grant et al. [15] have used independent component analysis to separate topic signals from source code.

With respect to software process recovery, Cook [5] used Markov chains to describe the state transitions in a process using *process discovery*. Jensen and Scacchi [16] attempted a manual approach to process discovery, they mined web resources to "discover workflows". German manually mined process documentation and mailing-lists in order to describe the development processes of GNOME project developers [17]. Ripoche tried a more automatic approach and studied bug tracker processes using state diagrams [18]. Zaidman et al. [19] studied testing related development processes using co-changes between source code and tests.

III. Methodology

Our goal is to demonstrate that software process recovery is feasible, so we need to be able to demonstrate that:

- process is observable [20], [21], based upon the repositories we are studying;
- software development artifacts can be partitioned and associated with various work flows such as implementation or testing, and this can be done manually or automatically [20], [21], [22], [23], [24];
- the purpose of an artifact can sometimes be recovered from itself based upon its own attributes [22], [23];
- the topics of change and development are partially observable [25], [26];
- generalizable topics exist within software development projects [25], [26];
- a general overview of a software development process is recoverable [24] using these techniques.

Thus for a project we have to identify iterations, releases and tags [20], [21]; describe behaviour within a project at a certain time [20], [21], [22], [23], [25], [26]; identify repeating and consistent behaviour around events [20], [21]; show that some FLOSS projects demonstrate concurrent and parallel effort across disciplines [20], [21], [24].

If we can find topics or the purposes behind events we can relate them to parts of the software development life cycle: requirements [25], [26], [24] , design [25], [26] , implementation [22], [23] , testing [20], [21] , deployment [24] , project management [24] , maintenance [20], [21], [24] , and quality assurance [25], [26], [24].

The artifacts that relate to these various aspects of software development and software development processes are recoverable from repositories such as version control systems, mailing-list archives and bug trackers. Thus we plan to mine revisions and commits to source code, documentation, test code, build system, and assets from version control systems such as CVS, Bitkeeper, SVN, Fossil and Git. From bug trackers and mailing-list archives we mostly mine the discussions, the threads of messages that developers and users leave behind. Discussions are often unstructured and require different kinds of natural language processing in order to tease out events that are relevant to the development of the project and the de facto software development processes.

IV. Results

Much of the work in this PhD thesis has been previously peer reviewed and published in multiple venues. Most of this research was proposed and then evaluated via case studies. Each of these works, which are components of the chapters of the thesis, illustrated different methods to get different information about software development processes out of the artifacts of development.

A. Is process observable? Is there consistent behaviour?

In our work on release patterns [20], [21] we characterized the software development processes of multiple database systems by observing how changes to source code, test code, build system code and documentation reacted around release time. Releases are very important events in a software project's life because they are an explicit process event: they indicate that the current branch of the software is ready to be deployed and executed. Releases are easy to mine manually whether by checking documentation or inferring automatically via version control tags. These releases are very important because they are points in time where the developers agree that the project has reached a certain milestone or stability point such that it can be packaged for consumption. We found we could describe the process around release time by breaking up revisions into STBD changes: Source, Test, Build, Documentation. We found that the behaviour of changes in these four revision streams around release time was consistent internally to a project, but inconsistent across projects. This internal consistency was an indication that a software development process was being followed by the project's developers. This work also gave us access to four valuable event streams that are process rich, but also provided evidence of concurrent development effort.

B. Are there parallel or concurrent development efforts within some FLOSS projects?

We have investigated, characterized, and classified large changes in version control systems [22], [23]. We observed that different kinds of maintenance activities occurred in parallel with implementation changes. We found that for many FLOSS projects implementation changes and maintenance changes were occurring concurrently. We even refined these results further and produced automatic systems that could classify changes by their maintenance classification type.

C. Are topics of development evident in these repositories?

We investigated windowed developer topics [25], [26], where we applied natural language processing and Latent Dirichlet Allocation (LDA) to find local and global developer topics, we found that the topics of change and development are partially observable. We found that there was a great amount of topic churn over time. Most topics, 80 to 90 percent, never ever reoccurred and were quite local, many of the 10 to 20 percent of repeating topics did reoccur across most

of the project's lifetime. In collaboration with Neil Ernst, we investigated these topics further and in our "What's in a name?" study [26]; we found that many of these repeating topics were related to non-functional requirements. The *non-functional requirements* (NFRs) we investigated included: efficiency, reliability, maintainability, portability, usability and correctness. What is interesting about this result is that we took previous knowledge that software had these issues and we found that tools like LSI and LDA could spot topics related to these issues within a repository. NFRs occur across many software projects. Most projects have to deal with multiple NFRs. We also found that developer topics were more useful when they were labelled and that topics lacked descriptive power until they were labelled, named or interpreted by man or machine.

D. Can we recover some of the software development processes of a project?

Our work on Recovered Unified Process Views [24] (RUPV) demonstrated an integration of MSR research and our work in order to produce a summary of observable software development process activities recovered from artifacts of software development. We modelled our summary on the Unified Process, based on the Unified Process diagram. The Unified Process diagram illustrates how software development consists of parallel and concurrent workflows (such as implementation, requirements and design) over time, and at different times in a project's life-cycle there is a different amount of effort or emphasis on certain workflows. For example it is assumed at the start of a project's life, that much effort is spent requirements and design while in the middle of a life-cycle a project is primarily being implemented, maintained and deployed. The RUPVs showed that we could observe many aspects of software development processes and show them as parallel time-lines much like the Unified Process diagram. These concurrent time-lines, with their differing proportions of process events, would be representative of the underlying software development process being followed by a project's developers.

Thus we were able to integrate MSR work and our own work to successfully recover some aspects of software development processes observed from artifacts that developers leave behind.

E. Threats and Challenges

The three main issues that this research faces are the observability of process related events, the reliance on programmers to annotate or describe their actions, and the use of case studies to validate these techniques and methods.

Observability of process is an important issue facing this research, as certain aspects of a project are often implicit or unobservable. For instance face to face meetings are often not recorded and cannot be extracted. As well some projects have implicit requirements because they are meant to mimic the functionality of existing products. We found that business analysis and project management events were particularly hard to recover from the artifacts that we analyzed [24]. The quality of the data in the project affects the observability of the software development processes that were used.

We rely on the artifacts that developers create. Developers do not create these artifacts in order to enable research, these artifacts are meant to help the development of a software project. Thus the annotations used by programmers might not be enough to determine their behaviour. Developer behaviour can change over time, programmers can annotate and describe their changes inconsistently. Thus much analysis depends on the context, the programmers, and the culture of the project.

Our validations mostly consisted of case studies. Often we would create training sets of data based on real data in repositories, the classification of this data was done manually by hand and using our own judgement. This implies that some of these results might not be as generalizable as we hope thus we have to approach each project distinctly and evaluate the context of the project and the availability of its artifacts.

V. Conclusions

Software process recovery is the recovery of software development processes from the artifacts that developers leave behind when they develop software. Our thesis is that we can recover software development process information from artifacts found in version control repositories, mailing-list archives and bug trackers.

We have motivated the need for software process recovery as a cost effective, non-invasive and non-developer intensive method of recovering software development process information from existing projects, even dead projects, for a variety of reasons and for a variety of stakeholders. Practical reasons for software process recovery include certification for ISO 9000 or CMM, project management, developer awareness and acquisitions. Stakeholders who could benefit from software process recovery include: managers, new developers, investors, buyers, and researchers.

Our main contribution is the leveraging of MSR research to recover software development processes from software repositories. From a research perspective, we have contributed multiple ways of inferring the purpose of certain development artifacts. In the previous sections we illustrated many methods of automated and semi-automated software process recovery. These techniques can be applied after the fact, they rely on artifacts of software development rather than programmer interviews. We have integrated many of these techniques into Recovered Unified Process Views (RUPV) and published that work [24].

We have many repositories to analyze: version control systems, mailing-list archives and bug tracking systems. The artifacts within these repositories are often fine grained and lacking in summaries. One of the primary methods of this work has been to summarize these artifacts whether by statistics, case study, or classification. We classified revisions by their file types associated with software processes (source code, testing, documentation, build) [20], [21]. We classified version control commits by their maintenance categories [22],

[23]. We classified bugs and mailing-list discussions by non-functional requirements [25], [26], [24].

We were successful in recovering some software development processes. By splitting revisions into source, test, build, and documentation changes we could clearly see that within a project there were a repeating release patterns [20], [21]. We could observe the breakdown of changes by maintenance class [22], [23] and automate this classification [23]. We observed that implementation changes still occur late in a project's life cycle. Also we observed that most developer topics [25], [26] occur once but there are many topics which reoccur across the lifetime of a project. We eventually found that many of the recurrent topics were related to non-functional requirements. We integrated much of our previous work and were able to describe how different disciplines, such as requirements or implementation, were being worked on in different proportions across time [24].

Do our techniques work? Each technique was evaluated and validated against multiple case studies. In most cases these techniques have been published and peer-reviewed. Thus each part is validated, leaving only the sum of the parts to be truly validated. We attempted to validate the integration of these techniques using the Recovered Unified Process Views (RUPV) [24]. RUPV enabled us to make interesting observations about deployment and requirements related artifacts in SQLite and FreeBSD case studies.

We proposed software process recovery. We proposed many methods and techniques to recover specific aspects of software development processes, we then integrated many of these techniques to produce a general overview: Recovered Unified Process Views. We are confident that the combination of these proposed techniques and their integration confirms that we can recover some software development processes from these artifacts. This thesis is very preliminary work in the field of software process recovery; there is much to do, and still much MSR-related research to apply to software process recovery.

Acknowledgements: I thank my supervisors, Michael W. Godfrey and Richard C. Holt, for all of their support.

REFERENCES

[1] D. Stelzer, W. Mellis, and G. Herzwurm, "A critical look at iso 9000 for software quality management," *Software Quality Control*, vol. 6, no. 2, pp. 65–79, 1997.

[2] M. C. Paulk, B. Curtis, E. Averill, J. Bamberger, T. Kasse, M. Konrad, J. Perdue, C. Weber, and J. Withey, *The capability maturity model: guidelines for improving the software process*, M. C. Paulk, C. V. Weber, B. Curtis, and M. B. Chrissis, Eds. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[3] 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," *J. Softw. Maint. Evol.*, vol. 19, no. 2, pp. 77–131, 2007.

[4] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters, "Workflow mining: A survey of issues and approaches," *Data & Knowledge Engineering*, vol. 47, no. 2, pp. 237 – 267, 2003.

[5] J. E. Cook, "Process discovery and validation through event-data analysis," Ph.D. dissertation, Computer Science Dept., University of Colorado, 1996.

[6] A. Capiluppi, P. Lago, and M. Morisio, "Characteristics of open source projects," vol. 00. Los Alamitos, CA, USA: IEEE Computer Society, 2003, p. 317.

[7] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles, "Towards a theoretical model for software growth," in *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*. Washington, DC, USA: IEEE Computer Society, 2007, p. 21.

[8] A. Mockus, "Amassing and indexing a large sample of version control systems: Towards the census of public source code history," in *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, May 2009, pp. 11–20.

[9] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles, "Forecasting the number of changes in eclipse using time series analysis," in *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*. Washington, DC, USA: IEEE Computer Society, 2007, p. 32.

[10] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic, "An information retrieval approach to concept location in source code," *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, pp. 214–223, Nov. 2004.

[11] D. Poshyvanyk and A. Marcus, "Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code," in *International Conference on Program Comprehension*, June 2007, pp. 37–48.

[12] A. Kuhn, S. Ducasse, and T. Girba, "Enriching reverse engineering with semantic clustering," *Reverse Engineering, 12th Working Conference on*, pp. 10 pp.–, Nov. 2005.

[13] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining concepts from code with probabilistic topic models," in *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. New York, NY, USA: ACM, 2007, pp. 461–464.

[14] L. H. E. Stacy K. Lukins, Nicholas A. Kraft, "Source code retrieval for bug localization using latent dirichlet allocation," in *15th Working Conference on Reverse Engineering*, 2008.

[15] S. Grant, J. R. Cordy, and D. Skillicorn, "Automated concept location using independent component analysis," in *15th Working Conference on Reverse Engineering*, 2008.

[16] C. Jensen and W. Scacchi, "Simulating an automated approach to discovery and modeling of open source software development processes," in *Proceedings of the Third Workshop on Open Source Software Engineering ICSE03-OSSE03*, May 2003.

[17] D. M. German, "Decentralized open source global software development, the GNOME experience," *Journal of Software Process: Improvement and Practice*, vol. 8, no. 4, pp. 201–215.

[18] G. Ripoche and L. Gasser, "Scalable automatic extraction of process models for understanding f/oss bug repair," in *Proceedings of the International Conference on Software and Systems Engineering and their Applications (ICSSEA'03)*, December 2003.

[19] A. Zaidman, B. V. Rompaey, S. Demeyer, and A. v. Deursen, "Mining software repositories to study co-evolution of production & test code," in *ICST '08: Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 220–229.

[20] A. Hindle, M. Godfrey, and R. Holt, "Release Pattern Discovery via Partitioning: Methodology and Case Study," in *Proceedings of the Mining Software Repositories 2007*. IEEE Computer Society, 2007.

[21] ——, "Release pattern discovery: A case study of database systems," in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, Oct. 2007, pp. 285–294.

[22] A. Hindle, D. M. German, and R. Holt, "What do large commits tell us?: a taxonomical study of large commits," in *MSR '08: Proceedings of the 2008 international working conference on Mining software repositories*. New York, NY, USA: ACM, 2008, pp. 99–108.

[23] A. Hindle, D. M. German, M. W. Godfrey, and R. C. Holt, "Automatic classification of large changes into maintenance categories," in *International Conference on Program Comprehension*, Vancouver, 2009.

[24] A. Hindle, M. W. Godfrey, and R. C. Holt, "Software process recovery using recovered unified process views," in *Proceedings of the International Conference on Software Maintenance 2010 (ICSM 2010)*, 2010.

[25] ——, "What's hot and what's not: Windowed developer topic analysis," in *International Conference on Software Maintenance*, Edmonton, Alberta, Canada, September 2009, pp. 339–348.

[26] A. Hindle, N. Ernst, M. W. Godfrey, R. C. Holt, and J. Mylopoulos, "What's in a name? on the automated topic naming of software maintenance activities," 2010, in submission: http://softwareprocess.es/whats-in-a-name.