Kwan, Irwin and Damian, Daniela, 2011. A Survey of Techniques in Software Repository Mining.
Technical Report DCS-340-IR, University of Victoria

A SURVEY OF TECHNIQUES IN SOFTWARE REPOSITORY MINING

IRWIN KWAN, DANIELA DAMIAN
SOFTWARE ENGINEERING GLOBAL INTERACTION LABORATORY
UNIVERSITY OF VICTORIA
{IRWINK,DANIELAD}@CS.UVIC.CA

Abstract. Digital records of software-engineering work are left by software developers during the development process. Source code is usually kept in a software repository, and software developers use issue-tracking repositories and online project-tracking software, as well as informal documentation to support their activities. The research discipline of *mining software repositories* (MSR) uses these extant, digital repositories to gain understanding of the system. MSR has not been applied to model-driven development or *model-driven engineering* (MDE). In particular, model management deserve particular attention. Model management covers challenges associated with "maintaining traceability links among model elements to support model evolution and roundtrip engineering", "tracking versions", and "using models during runtime". These problems can be addressed by investigating the models themselves and their relationship to other artifacts using MSR. The objective of this report is to survey state-of-the-art research in MSR and to discuss how these MSR techniques are applicable to the problems faced in MDE. Extracting information about what factors affect model quality, how people interact with models in the repository, and traceability to other artifacts advance our understanding of software engineering when MDE is used.

1. Introduction

Digital records of software-engineering work are left by software developers during the development process. Source code is usually kept in a software repository, from which the history of the source code can be extracted. Software developers usually augment their development by using issue-tracking repositories [14, 36] and online project-tracking software [6], as well as formal documentation like specifications and manuals and informal communications like emails [10, 42] and Wiki pages [18]. The research discipline of mining software repositories (MSR) uses these extant, digital repositories to gain understanding of the system with respect to defect-prone modules, social interactions, or traceability.

Models, at the time of writing, have not received much attention in MSR. A handful of research exists in model mining for the purpose of gaining an understanding of software evolution and software engineering [67]. Model-driven development or *model-driven engineering* (MDE) is experiencing a kind of "coming of age" with interest in both industry and research rising significantly [61, 24]. However, there are a number of issues that continue to affect MDE. In a roadmap described by France and Rumpe [24], the primary issue with MDE is the "implementation gap" that requires a model be translated from the domain level to the software-engineering level. As this gap rapidly closes, there are a number of additional issues that are arising as a consequence of MDE. In particular, model management deserve particular attention. Model management covers challenges associated with "maintaining traceability links among model elements to support model evolution and roundtrip engineering", "tracking versions", and "using models during runtime". These problems can be addressed by investigating the models themselves and their relationship to other artifacts using MSR.

The objective of this report is to report on state-of-the-art research in MSR and to discuss how these MSR techniques are applicable to the problems faced in MDE. We believe that many existing techniques in social software engineering and software repository mining may be applicable to models. Extracting information about what factors affect model quality, how people interact with models in the repository, and traceability to other artifacts advance our understanding of software engineering when MDE is used.

2. A Brief Overview of MSR

The purpose of mining software repositories is to use the wealth of information available as a result of the software-development process to learn how the process can be improved. Using information available such as issue-tracking repositories, source code, and documents, relationships between qualities in these artifacts can be identified, and knowledge of software processes and characteristics can be acquired. This knowledge can lead to the development of improvements. These improvements may come in the form of better processes, techniques, or perhaps simply an awareness of an organization's engineers of a particular phenomenon.

2.1. **MSR Data Sources.**

MSR relies extensively on source code as a source of data. This source code is commonly found in repositories such as Subversion [23], CVS [22], Git [26] or Mercurial [48], and therefore contains multiple versions of the same system over time. In addition to the source code, each source code management system contains a link from the source code to a user ID of the person who made a change. Each change also comes with an attached comment as well that usually provides context about the change.

In addition to source code, MSR has extended its reach to include data surrounding the software process such as the bug repository, project management, and mailing lists. Using these sources helps make the analysis of a system and its history more comprehensive than using source code alone, and especially provides context to the system under investigation.

While many elements in MSR are traceable to each other, some aspects of MSR are related to rebuilding trace links from one kind of artifact to another. Reverse engineering, for instance, involves reconstructing design specifications from source code. There is also much work on traceability between source code and natural language documents such as documents and email [63, 44, 7].

The relationships between these digital repositories provides a large amount of information, although these repositories alone are not sufficient to acquire a full understanding of an organization [4]. Nonetheless, the value of repository mining is in acquiring an understanding of software using automated techniques, and in developing technology that can assist developers in their tasks.

2.2. **Areas of MSR.** There are few existing taxonomies of MSR in literature. A comprehensive review of literature prior to August 2006 is by Kagdi et al. [38], which classifies MSR in the context of software evolution. It identifies four layers: software evolution, purpose, representation, and information sources. We are concerned mostly with the purpose—divided into "market basket questions" where MSR is used to answer a question about software or its process, or to investigate the prevalence of some kind of metric.

In this report, we provide an overview of recent MSR work, mostly from 2006 and later. The literature is classified roughly by the area of application.

- Identifying and Predicting Software Quality
- Identifier Analysis and Traceability
- Clone Detection
- Process Improvement and Software Evolution
- Social Aspects in Software Development
- Recommender Systems and Interactive Systems

At this time, we believe that these categories represent the state-of-the-art work in the MSR community right now, and that these techniques are also applicable to examining MSR in MDE.

3. Identifying and Predicting Software Quality

MSR is used to identify potential quality issues in a software system. In general the research into applying MSR to improve quality investigates two questions [29]. These questions are, "How many defects will be in this module?" and "In which modules do defects occur?" MSR also develops prediction models to determine how many defects are in the software, and to determine which modules have defects.

There is a distinction between *defect* and *failure*. A defect is an error in the source code, which may or may not result in a problem experienced by the end user. A failure, on the other hand, is a problem that the user experiences, usually as a result of a defect. Not every defect leads to a failure. Thus, it is common for developers to examine *post-release failures* because these are the issues that affect customers and therefore have most impact on the perceived quality of the product. Many studies are also interested in the number and the location of *post-release defects* and *pre-release defects* as well.

There has also been interest in measuring the performance of the development team itself. Pre-release defects indicate how well the team performs with respect to programming. Another criteria for quality is *build success* [30, 64, 43], which counts if the software builds successfully. The suggestion is that a software build fails because of a mistake within the development team, because of poor integration processes, or because of a lack of coordination. Finally, a limited number of studies have examined *defect resolution time* [31, 17, 53] to identify how quickly defects are fixed after being identified in the organization.

Most of the recent literature in software defect analysis examines if there is a correlation between a particular phenomenon, such as a software complexity metric, and defect count, and examines if that phenomenon can be used to predict defect count or defect-prone modules. Usually, statistical models are used to identify failure-prone files [66, 52] though some techniques are applicable to individual lines of code [68] or to modules [54]. Zimmermann et al. [71] used the categories "complexity metrics" and "historical data"; complexity metrics included dependency calculations, and historical data in this context included code churn. In recent years there has also been an interest in using metrics that reflect social aspects of software organizations for failure prediction [52, 47, 11].

I use the following categories to broadly classify the current literature in MSR failure counting and failure prediction.

- Software metrics, which includes complexity metrics and dependencies
- Software evolution, which examines repository histories for changes
- Social factors, which examines social interactions, often in relation to technical aspects

This is not a comprehensive list of techniques, but for the purpose of this report most existing studies can be placed under one of these categories.

As an introduction to the field of defect prediction, Zimmermann et al. [71] provides an overview on preparing software repositories for the analysis of defects. Nagappan et al. [51] also provide a great description of the techniques commonly employed in defect counting and prediction.

3.1. **Software Metrics and Quality.** A number of classic metrics have been used extensively for failure prediction, primarily Halstead's complexity measures [28] and McCabe cyclomatic complexity [45]. McCabe tends to be a relatively good predictor [51], though recently software metrics have been used in conjunction with other factors such as types of dependencies or social interactions. As the field of MSR has matured, Nagappan et al. [51] identified that the choice of metrics depends highly on the project under investigation, and that a successful prediction model used in one project was not appropriate for use in another project.

Initially, syntactic dependencies based on call graphs were commonly employed but logical coupling is considered standard. Gall et al. [25] described *logical coupling* as one such dependency that is constructed by grouping the files together that comprise a bug or feature change to the software. Cataldo et al. [16] proposed *workflow dependency* as a socio-technical dependency that occurs when an issue in the issue-tracking system is reassigned from one developer to another. Each of these dependency measurements have been used successfully in failure analysis but logical coupling has been shown to have a strong influence on failures [16] and have been utilized for recommendations [68].

Zimmermann and Nagappan [**?**] mapped software dependencies as a dependency graph and applied social network analysis measurements to Windows Server 2003. They found that some centrality measures correlated to the number of failures, but that code complexity measures had higher correlations to the number of defects. However, network measures on dependency graphs in combination with complexity measures were able to successfully predict failures. Nguyen et al. [54] performed a replication of this analysis on Eclipse, but found that social network analysis metrics did not correlate as highly to failures as in Windows Server 2003. Their finding that both social network analysis metrics and complexity could predict failures in Eclipse corroborated the earlier finding. Nguyen et al. [54] postulated that ego centrality metrics reflect software engineering practices. Cataldo et al. [16] compared different representations of a dependency to identify failure-prone files. They examined syntactic dependencies based on call graphs; logical dependencies based on relationships between files that tend to be changed together; workflow dependencies, which capture when an issue's assignment is changed from one developer to another; and coordination requirements, which captures relationships among developers that reflect dependencies among files. The authors identified that workflow dependencies, representing a change in assigning of an issue, and logical dependencies, representing files that are changed to fix a particular issue, are factors that are more likely to affect the likelihood of failure-prone files than syntactic dependencies.

3.2. **Software Evolution and Quality.** Software evolution techniques use historical data, usually code changes, to predict defects that may occur later in the project.

Nagappan and Ball [50] used code churn metrics for predicting defect density in Windows Server 2003. A code churn metric measure the amount of change that source code experiences in a time period. They found that these code churn metrics correlated highly to defects per thousand lines of code, and were excellent predictors of defect density.

Hassan and Holt [29] proposed the use of a "cache" from the operating system discipline in defect prediction. This cache identifies the top failure-prone files and updates the cache when a fault is fixed. This was later expanded into the FixCache algorithm [41], which has attracted interest among researchers (ex. [60]). FixCache works by keeping a cache of entities that are likely to have faults. When a fault is repaired, the algorithm consults the cache. This algorithm takes advantage of temporal locality, as well as spatial locality to predict faults in entities.

Hassan and Zhang [30] applied decision trees to analyse build certification results, and determined that the status of the previous build was one of the most important determining factors, followed by the developer and subsystem attributes.

3.3. **Social Factors and Quality.** In addition to using software metrics, social factors related to the project have also been used to predict failures. Cataldo et al. identified a correlation between an alignment between social interaction and software dependencies, called socio-technical congruence, and defect repair times [17]. Nagappan et al.[52] identified that organizational aspects could predict which binaries were prone to post-release failures. In particular, their model involved metrics that counted code changes belonging to a by counting the number of change software engineers working for this person made. Their results had higher precision and recall than other models using code churn, code complexity, dependencies, and pre-release defects. A follow-up study on Windows Vista by Pinzger et al. [56] examined the relationship between developers that contributed to the same code module, and determined that social network centrality measures were able to predict failure-prone binaries, as well as the number of post-release failures. A similar study of developer networks by Meneely et al. [47] also used centrality and other abstractions of code churn metrics to successfully prioritizing failure-prone files. Bird et al. [12] examined socio-technical networks, which are networks that represent both contributions and dependencies among developers and binaries. Social network analysis measurements applied to the socio-technical network were able to predict failures in both Windows Vista and Eclipse. The success of social network analysis in MSR led to Wolf et al. describing a general technique to apply social networks in software engineering [65].

Distance has also been used to determine influences on defects. Cataldo et al. [15] investigated feature changes from a repository and identified that global software engineering in their context was the largest effect contributing to defects. However, Bird et al. [11] reported a different result, where the amount of distance between developers in Windows did not significantly influence the number of post-release failures in binaries.

With respect to build success, Wolf et al. [64] determined for the IBM Jazz team that centrality measures did not identify a difference between successful builds and failed builds, but were able to use a Bayesian classification technique to predict build success. Cataldo et al. [16] found that high values for workflow dependencies and coordination requirements, which reflect socio-technical aspects in a software team, were more likely to indicate defects in files.

3.4. **Moving Forward With Defect Prediction.** Recent work examines the quality of cross-product defect prediction, in which one software project is used to predict defects in another similar project. This was applied by Zimmermann et al. [69] on a study of two web browsers; they found that Firefox could be used to predict defects for either itself, or for Internet Explorer, but that Internet Explorer could not be used to predict defects for Firefox. Advancing research in this direction may be applicable for product lines, commonly seen in the form of different car models in the automotive sector.

It should be emphasized again that no single metric can be considered a "gold standard" for defect prediction. Nagappan et al. [51] found that no single measure works across every project. Shihab et al. [62] evaluated a statistical technique that can, with improved reliability, identify which factors in a logistic regression prediction model are significant. Such techniques are necessary to identify which metrics are most applicable to one's own software.

The number of warranty incidents can be considered as a post-release failure, as in Gokpinar et al. [27]. If the MDE process uses a central asset manager, and an issue-tracking system that connects to this asset manager, then the changes made to a model to address a particular software defect can be identified. In embedded systems, a post-release defect is a pretty big deal, so while I would not expect many of these issues coming up, if one does occur a lot of attention will probably be paid to it.

4. Identifier Analysis and Traceability

Traceability is an important problem in software engineering. Enabling good traceability improves maintenance and program comprehension. One way to approach this problem is through identifier analysis. Programmers try to use meaningful names for entities in their programs. These names, or *identifiers*, often correspond to real-world domain concepts or actions [2]. Being able to extract identifiers from source code can lead to link recovery and traceability between source code and other texts.

Source code often involves identifiers made of compound words, like “getProp”, “debug_system”, or “print_file2device”. Identifier splitting is the process of converting these identifiers from source code into real-world concepts. Two techniques, CamelCase and Samurai, are commonly used for identifier splitting: CamelCase [2], which splits identifiers using underscores, numbers, and alphabetical case changes; and Samurai [21], which further uses substrings to refine identifiers. Both techniques have advantages but it has been identified that they are extremely similar in functional performance [20]. Arnaoudova et al. [5] used identifiers as a basis for their defect prediction model, and identified that a module containing terms “scattered” throughout the source code is highly correlated with the number of defects in that module.

Many elements of models require user-assigned names. States and signals use various names assigned by an engineer. Identifier analysis improves traceability between models and other artifacts such as requirements documents or even natural language documents.

5. Clone Detection

Clone detection is the investigation of duplicated sections of source code, usually through the copy and paste function in an editor. Clone detection started to become of interest since it was discovered that 10-15% of code in a software system was copied and pasted [8]. If there is a defect in reused code fragments, then fixing that defect can be problematic. Frequent cloning can also suggest the need for the creation of subroutines that use the cloned fragments. Cloning has received a lot of attention in source code, with different types of cloning being used such as token-based cloning [39] and abstract syntax tree-based cloning [9]. Al-Ekram et al. [1] discovered that source code cloning for the purpose of reuse was not prominent in a number of open-source text editors.

The application of clone detection to MDE is not yet common, though early investigations indicate that a large proportion of models are cloned [19, 37]. Early work by Diessenboeck et al. [19] applied a graph-based algorithm to detect clones, but over half of the clones identified were false positives. More research into clones in models is needed, and especially into the semantics of model cloning. Because models are manipulated in a different way compared to source code, many of the assumptions of cloning in source code may not apply to MDE. This is one area in which empirical research would be valuable.

However, there is an argument that not all code cloning should be considered harmful [40], which may be the case in MDE, which is manipulated primarily through using graphical tools rather than directly using text. The semantics of a model must be considered carefully before clone detection can be applied to MDE.

6. Process Improvement and Software Evolution

MSR can identify how source code changes over time during a process. It can also identify the general course of a project. Examples of improvements would be identifying who should triage a bug [3], identifying the trends of code commits [32, 34], and determining how to file a good bug report [70]. Ratzinger, et al. identified classification techniques that were able to predict refactoring activity [58]. Each of these findings lead to the improvement of areas in the software process.

MSR can also be used to gain an understanding of system evolution as a whole. Robles, et al. used source code extraction to examine changes over time in the Debian Linux distribution [59]. Hindle, et al. [33, 34] extracted revisions associated with various steps of a software process using text analysis techniques of source code commits and mailing lists. Communication content over time has been examined as well; Ernst and Mylopoulos [**?**] examined mailing lists to determine if requirement-related discussions became more prominent as software matures, but did not find clear evidence supporting this hypothesis.

MSR techniques have also been used to provide a statistical model of software readability [57]. This model of readability determines that program readability can largely be determined by the size of the code fragment, the variety of tokens in the fragment. While a statistical model cannot be used as the sole determinant of whether code is readable, it moves us toward automatically assisting developers with writing code.

This understanding of the software process at large is applicable to MDE. One of the reported benefits of using MDE is that models are more abstract and are easier to understand [61]. A thorough understanding of current software practices using models can help incorporate the beneficial properties of models into improving both tools and processes especially with respect to modifying and maintaining them.

7. Social Aspects in Software Development

In recent years, many of the issues in software engineering have been attributed to social and human factors. Distributed software engineering also results in increased use of visible communication, especially over mailing lists and through issue-tracking systems. These sources made it possible to investigate collaboration in software engineering using various data mining techniques, and in particular to investigate how people behave in such a setting based on their communication practices. This was made lucid by research in distributed software development. Herbsleb and Mockus [31] examined a number of issues from industry and found that issues that required multiple sites increased the closing time by an average of 2.5 times longer than single-site issues. However, Nguyen et al. [53] more recently identified that multi-site collaboration in a different team did not affect defect repair times as strongly as in Herbsleb and Mockus's work.

There has also been interest in learning about communication structures overall, especially in the open-source context. Open-source software development is driven by motivation factors that differ highly from traditional software development and therefore may behave differently from traditional industrial software development. Bird et al. [10] used social network analysis to visualize interactions between developers in a mailing list. Howison et al. [35] used mailing list data to identify organizational structures of numerous open-source systems, identifying if they were centralized or decentralized. Bird et al. [13] identified that many open-source projects were grouped into "communities", which meant that multiple small groups of developers tended to communicate; his technique allows the identification of community structure in software development projects.

A study of "Linus's Law", in which Eric Raymond said "Many eyes makes all bugs shallow" was made using MSR techniques [46]. The authors concluded that files with many contributors were more likely to have security issues.

The value of examining how team members behave and interact contributes to our understanding of software development. Though one might presume that it would be difficult to reason about the social side of software development from repositories alone, MSR research has shown that a large amount of information can be extracted. In a distributed organization that uses MDE, the collaboration patterns may be different than developers who write source code. Examining how team members working on models collaborate may reveal findings that lead the development of tools and processes that are specially-tailored for MDE.

8. Recommender Systems and Interactive Systems

Though software repository mining has leaned toward knowledge discovery and relationship identification, many of the fundamental techniques have been adapted for use within software development environments. These techniques are usually used as interactive tools that respond to the user's actions and use the information available in a repository to assist the user with decisions. While this area is too large to survey thoroughly in this report, we highlight some recommender systems and tools below.

8.1. **Change Recommenders.** Zimmermann et al. [68] used a technique that examined changes from a project's history in order to make fine-grained recommendations of what source code entities should be changed at the function or subsection level, in addition to files. This was built into a recommender system in Eclipse that suggests entities to change as a developer makes modifications to code.

8.2. **Document Traceability.** De Lucia et al. [44] implement a traceability technique that uses identifier analysis and the unsupervised Latent Semantic Indexing (LSI) technique in an Eclipse plugin called COCONUT. This plugin makes the relationship between code and natural language documentation explicit. In a controlled experiment, subjects who used this plugin wrote code that used vocabulary similar to vocabulary in existing documentation compared to those who did not use the plugin. It is believed that by employing similar vocabulary in the source code and the documentation, both the source code and the documentation is easier to understand.

8.3. **Expertise Recommenders.** Minto and Murphy [49] created a recommender system that would identify developers to communicate with based on a calculation suggested by Cataldo et al. [17]. This system identifies developers that frequently modify a file and therefore can be considered experts regarding the contents of that file.

Ohira et al. [55] use social network analysis and code contributions to recommend project contributors across projects. A software developer who is working on one project may be able to seek advice from an expert in another area.

9. Applying Software Repository Mining to Model-driven Development

Research on data mining model-driven development is relatively new. Zhang and Sheth [67] discuss one examination of model-driven development using a statistical control process but few applications of the above techniques discussed in this report appear in MDE, to our knowledge.

There are many differences between the traditional software repository mining research and model-driven development. Techniques are applied with the assumption that text artifacts are changed, often a few lines at a time across many files. The experiments are often done on open-source projects, which have characteristics that are not common to industrial, model-driven development such as open repositories and systems that are not mission-critical and embedded.

However, this does not mean that these techniques cannot be applied. There are a number of common techniques and research goals that can be achieved, but the model-driven development environment is a different context. Both models and source code are stored in a repository, which can be mined. Models are commonly stored in XML or another markup format, meaning that there are semantic differences between models and source code. However, because models are in plain text format, like code, the analysis techniques used for source code (such as token-based clone detection) should be applicable to models as well. However, changes to the metrics will have to be made to ensure that the theory behind the mining technique is reasonable. For instance, in a fault detection algorithm based on code churn, lines is a reasonable estimator of churn in source code. However, a small change in a model, such as creating a transition, may manipulate many lines in a file simultaneously, and thus some pre-processing may be necessary to determine the nature of the change.

Though many of the existing techniques in MSR are applied to open-source projects, there is no reason that these projects cannot be applied to industrial software. However, context of the setting is extremely important [43]. As a consequence, in addition to applying the techniques, some understanding of the automotive domain and of the way that software developers create and interact with models, through observations, interviews, or surveys, will augment software repository mining greatly.

References

[1] R. Al-Ekram, C. Kapser, R. Holt, and M. Godfrey. Cloning by accident: an empirical study of source code cloning across software systems. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, page 10 pp., nov. 2005. `doi:10.1109/ISESE.2005.1541846`.

[2] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *Software Engineering, IEEE Transactions on*, 28(10):970 – 983, oct 2002. `doi:10.1109/TSE.2002.1041053`.

[3] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 361–370, New York, NY, USA, 2006. ACM Press. Available from: `http://portal.acm.org/citation.cfm?id=1134285.1134336`, `doi:10.1145/1134285.1134336`.

[4] Jorge Aranda and Gina Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. *Software Engineering, International Conference on*, 0:298–308, 2009. `doi:http://doi.ieeecomputersociety.org/10.1109/ICSE.2009.5070530`.

[5] V. Arnaoudova, L. Eshkevari, R. Oliveto, Y.-G. Gué andhé andneuc, and G. Antoniol. Physical and conceptual identifier dispersion: Measures and relation to fault proneness. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1 –5, sept. 2010. `doi:10.1109/ICSM.2010.5609748`.

[6] Atlassian. Bug, Issue, and Project Tracking for Software Development - JIRA [online]. 2011. Available from: `http://www.atlassian.com/software/jira/` [cited 2011-03-16].

[7] Alberto Bacchelli, Michele Lanza, and Marco D'Ambros. Miler: a toolset for exploring email data. In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 1025–1027, New York, NY, USA, 2011. ACM. Available from: `http://doi.acm.org/10.1145/1985793.1985984`, `doi:http://doi.acm.org/10.1145/1985793.1985984`.

[8] B.S. Baker. On finding duplication and near-duplication in large software systems. In *Reverse Engineering, 1995., Proceedings of 2nd Working Conference on*, pages 86 –95, jul 1995. `doi:10.1109/WCRE.1995.514697`.

[9] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *Software Maintenance, 1998. Proceedings. International Conference on*, pages 368 –377, nov 1998. `doi:10.1109/ICSM.1998.738528`.

[10] Christian Bird, Alex Gourley, and Anand Swaminathan. Mining email social networks in postgres. In *Mining Software Repositories Workshop 2006, ICSE*, 2006.

[11] Christian Bird, Nachiappan Nagappan, Premkumar Devanbu, Harald Gall, and Brendan Murphy. Does Distributed Development Affect Software Quality? An Empirical Case Study of Windows Vista. *Communications of the ACM*, 52(8):85–93, August 2009.

[12] Christian Bird, Nachiappan Nagappan, Harald Gall, Brendan Murphy, and Premkumar Devanbu. Putting it all together: Using socio-technical networks to predict failures. *Software Reliability Engineering, International Symposium on*, 0:109–119, 2009. `doi:http://doi.ieeecomputersociety.org/10.1109/ISSRE.2009.17`.

[13] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. Latent social structure in open source projects. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 24–35, New York, NY, USA, 2008. ACM. `doi:http://doi.acm.org/10.1145/1453101.1453107`.

[14] Bugzilla.org. Home :: Bugzilla [online]. 2011. Available from: `http://www.bugzilla.org/` [cited 2011-03-16].

[15] Marcelo Cataldo and James D. Herbsleb. Factors leading to integration failures in global feature-oriented development: an empirical analysis. In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 161–170, New York, NY, USA, 2011. ACM. Available from: `http://doi.acm.org/10.1145/1985793.1985816`, `doi:http://doi.acm.org/10.1145/1985793.1985816`.

[16] Marcelo Cataldo, Audris Mockus, Jeffrey A. Roberts, and James D. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering*, 42, 2009. `doi:http://doi.ieeecomputersociety.org/10.1109/TSE.2009.42`.

[17] Marcelo Cataldo, Patrick A. Wagstrom, James D. Herbsleb, and Katheeln M. Carley. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In *Conf on Computer-supported Cooperative Work, Banff, Canada*, October 2006.

[18] Barthélémy Dagenais and Martin P. Robillard. Creating and evolving developer documentation: understanding the decisions of open source contributors. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, FSE '10, pages 127–136, New York, NY, USA, 2010. ACM. Available from: `http://doi.acm.org/10.1145/1882291.1882312`, `doi:http://doi.acm.org/10.1145/1882291.1882312`.

[19] Florian Deissenboeck, Benjamin Hummel, Elmar Jürgens, Bernhard Schätz, Stefan Wagner, Jean-François Girard, and Stefan Teuchert. Clone detection in automotive model-based development. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 603–612, New York, NY, USA, 2008. ACM. Available from: `http://doi.acm.org/10.1145/1368088.1368172`, `doi:http://doi.acm.org/10.1145/1368088.1368172`.

[20] Bogdan Dit, Latifa Guerrouj, Denys Poshyvanyk, and Giuliano Antoniol. Clustering support for static concept location in source code. In *International Conference on Program Comprehension, Kingston, Canada, June 22–24*, 2011.

[21] Eric Enslen, Emily Hill, Lori Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, MSR '09, pages 71–80, Washington, DC, USA, 2009. IEEE Computer Society. Available from: `http://dx.doi.org/10.1109/MSR.2009.5069482`, `doi:http://dx.doi.org/10.1109/MSR.2009.5069482`.

[22] Free Software Foundation. CVS - Open Source Version Control [online]. 2006. Available from: `http://www.nongnu.org/cvs/` [cited 2011-03-16].

[23] The Apache Software Foundation. Apache subversion [online]. 2011. Available from: `http://subversion.apache.org/` [cited 2011-03-16].

[24] Robert France and Bernhard Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society. Available from: `http://dx.doi.org/10.1109/FOSE.2007.14`, `doi:http://dx.doi.org/10.1109/FOSE.2007.14`.

[25] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance, Bethesda, USA.*, pages 190–198, 1998.

[26] git. Git - Fast Version Control System [online]. 2011. Available from: `http://git-scm.com/` [cited 2011-03-16].

[27] Bilal Gokpinar, Wallace J. Hopp, and Seyed M. R. Iravani. The Impact of Misalignment of Organizational Structure and Product Architecture on Quality in Complex Product Development. *Management Science*, 2010. Available from: `http://mansci.journal.informs.org/cgi/content/abstract/56/3/468`, `doi:10.1287/mnsc.1090.1117`.

[28] Maurice H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., New York, NY, USA, 1977.

[29] A.E. Hassan and R.C. Holt. The top ten list: dynamic fault prediction. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 263 – 272, sept. 2005. `doi:10.1109/ICSM.2005.91`.

[30] A.E. Hassan and K. Zhang. Using decision trees to predict the certification result of a build. In *Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on*, pages 189 –198, sept. 2006. `doi:10.1109/ASE.2006.72`.

[31] J. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, 29(6):481–494, 2003.

[32] A. Hindle, M.W. Godfrey, and R.C. Holt. What's hot and what's not: Windowed developer topic analysis. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 339 –348, sept. 2009. `doi:10.1109/ICSM.2009.5306310`.

[33] A. Hindle, M.W. Godfrey, and R.C. Holt. Software process recovery using recovered unified process views. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1 –10, sept. 2010. `doi:10.1109/ICSM.2010.5609670`.

[34] Abram Hindle, Neil A. Ernst, Michael W. Godfrey, and John Mylopoulos. Automated topic naming to support cross-project analysis of software maintenance activities. In *Proceeding of the 8th working conference on Mining software repositories*, MSR '11, pages 163–172, New York, NY, USA, 2011.

ACM. Available from: http://doi.acm.org/10.1145/1985441.1985466, doi:http://doi.acm.org/10.1145/1985441.1985466.

[35] James Howison, Keisuke Inoue, and Kevin Crowston. Social dynamics of free and open source team communications. In *Second Intl Conf on Open Source Systems*, Como, Italy, June 2006.

[36] IBM. The jazz project.

[37] Elmar Juergens. Research in cloning beyond code: a first roadmap. In *Proceeding of the 5th international workshop on Software clones*, IWSC '11, pages 67–68, New York, NY, USA, 2011. ACM. Available from: http://doi.acm.org/10.1145/1985404.1985419, doi:http://doi.acm.org/10.1145/1985404.1985419.

[38] Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.*, 19:77–131, March 2007. Available from: http://portal.acm.org/citation.cfm?id=1345056.1345057, doi:10.1002/smr.344.

[39] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Trans. Softw. Eng.*, 28:654–670, July 2002. Available from: http://portal.acm.org/citation.cfm?id=636188.636191, doi:10.1109/TSE.2002.1019480.

[40] Cory Kapser and Michael Godfrey. "cloning considered harmful" considered harmful: patterns of cloning in software. *Empirical Software Engineering*, 13:645–692, 2008. 10.1007/s10664-008-9076-6. Available from: http://dx.doi.org/10.1007/s10664-008-9076-6.

[41] Sunghun Kim, Thomas Zimmermann, E. James Whitehead Jr., and Andreas Zeller. Predicting faults from cached history. In *Proceedings of the 29th international conference on Software Engineering*, ICSE '07, pages 489–498, Washington, DC, USA, 2007. IEEE Computer Society. Available from: http://dx.doi.org/10.1109/ICSE.2007.66, doi:http://dx.doi.org/10.1109/ICSE.2007.66.

[42] Irwin Kwan and Daniela Damian. The hidden experts in software-engineering communication (nier track). In *ICSE'11: Proceedings of the Intl Conf on Software Engineering, Waikiki, USA*, 2011.

[43] Irwin Kwan, Adrian Schröter, and Daniela Damian. Does Socio-Technical Congruence Have An Effect on Software Build Success? A Study of Coordination in a Software Project. *Transactions on Software Engineering*, To appear. Available from: http://www.computer.org/portal/web/csdl/doi/10.1109/TSE.2011.29.

[44] Andrea De Lucia, Rocco Oliveto, Francesco Zurolo, and Massimiliano Di Penta. Improving comprehensibility of source code via traceability information: a controlled experiment. *International Conference on Program Comprehension*, 0:317–326, 2006. doi:http://doi.ieeecomputersociety.org/10.1109/ICPC.2006.28.

[45] Thomas J. McCabe. A complexity measure. In *Proceedings of the 2nd international conference on Software engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press. Available from: http://portal.acm.org/citation.cfm?id=800253.807712.

[46] Andrew Meneely and Laurie Williams. Strengthening the empirical analysis of the relationship between linus' law and software security. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 9:1–9:10, New York, NY, USA, 2010. ACM. Available from: http://doi.acm.org/10.1145/1852786.1852798, doi:http://doi.acm.org/10.1145/1852786.1852798.

[47] Andrew Meneely, Laurie Williams, Will Snipes, and Jason Osborne. Predicting failures with developer networks and social network analysis. In *Proceedings of the 16th International Symposium on Foundations of Software Engineering*, pages 13–23, New York, NY, USA, 2008. ACM. doi:http://doi.acm.org/10.1145/1453101.1453106.

[48] Mercurial. Mercurial SCM [online]. 2011. Available from: http://mercurial.selenic.com/ [cited 2011-03-16].

[49] Shawn Minto and Gail C. Murphy. Recommending emergent teams. In *Mining Software Repositories, 2007. ICSE Workshops MSR '07. Fourth International Workshop on*, pages 5–5, 2007. Available from: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4228642`.

[50] Nachiappan Nagappan and Thomas Ball. Use of relative code churn measures to predict system defect density. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 284–292, New York, NY, USA, 2005. ACM. `doi:http://doi.acm.org/10.1145/1062455.1062514`.

[51] Nachiappan Nagappan, Thomas Ball, and Andreas 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. Available from: `http://doi.acm.org/10.1145/1134285.1134349`, `doi:http://doi.acm.org/10.1145/1134285.1134349`.

[52] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In *Intl Conf on Software Engineering, Leipzig, Germany*, pages 521–530, May 2008. `doi:http://doi.acm.org/10.1145/1368088.1368160`.

[53] Thanh Nguyen, Timo Wolf, and Daniela Damian. Global software development and delay: Does distance still matter? In *Intl Conf on Global Software Engineering, Bangalore, India*, pages 45–54, August 2008. `doi:http://doi.ieeecomputersociety.org/10.1109/ICGSE.2008.39`.

[54] T.H.D. Nguyen, B. Adams, and A.E. Hassan. Studying the impact of dependency network measures on software quality. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1 –10, sept. 2010. `doi:10.1109/ICSM.2010.5609560`.

[55] Masao Ohira, Naoki Ohsugi, Tetsuya Ohoka, and Ken-ichi Matsumoto. Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. *SIGSOFT Softw. Eng. Notes*, 30:1–5, May 2005. Available from: `http://doi.acm.org/10.1145/1082983.1083163`, `doi:http://doi.acm.org/10.1145/1082983.1083163`.

[56] Martin Pinzger, Nachiappan Nagappan, and Brendan Murphy. Can developer-module networks predict failures? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 2–12, New York, NY, USA, 2008. ACM. Available from: `http://doi.acm.org/10.1145/1453101.1453105`, `doi:http://doi.acm.org/10.1145/1453101.1453105`.

[57] Daryl Posnett, Abram Hindle, and Premkumar Devanbu. A simpler model of software readability. In *Proceeding of the 8th working conference on Mining software repositories*, MSR '11, pages 73–82, New York, NY, USA, 2011. ACM. Available from: `http://doi.acm.org/10.1145/1985441.1985454`, `doi:http://doi.acm.org/10.1145/1985441.1985454`.

[58] J. Ratzinger, T. Sigmund, P. Vorburger, and H. Gall. Mining software evolution to predict refactoring. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 354 –363, sept. 2007. `doi:10.1109/ESEM.2007.9`.

[59] Gregorio Robles, Jesus M. Gonzalez-Barahona, Martin Michlmayr, and Juan Jose Amor. Mining large software compilations over time: another perspective of software evolution. In *Proceedings of the 2006 international workshop on Mining software repositories*, MSR '06, pages 3–9, New York, NY, USA, 2006. ACM. Available from: `http://doi.acm.org/10.1145/1137983.1137986`, `doi:http://doi.acm.org/10.1145/1137983.1137986`.

[60] Caitlin Sadowski, Chris Lewis, Zhongpeng Lin, Xiaoyan Zhu, and E. James Whitehead, Jr. An empirical analysis of the fixcache algorithm. In *Proceeding of the 8th working conference on Mining software repositories*, MSR '11, pages 219–222, New York, NY, USA, 2011. ACM. Available from: `http://doi.acm.org/10.1145/1985441.1985475`, `doi:http://doi.acm.org/10.1145/1985441.1985475`.

[61] B. Selic. The pragmatics of model-driven development. *Software, IEEE*, 20(5):19 – 25, sept.-oct. 2003. `doi:10.1109/MS.2003.1231146`.

[62] Emad Shihab, Zhen Ming Jiang, Walid M. Ibrahim, Bram Adams, and Ahmed E. Hassan. Understanding the impact of code and process metrics on post-release defects: a case study on the eclipse project. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering*

and Measurement, ESEM '10, pages 4:1–4:10, New York, NY, USA, 2010. ACM. Available from: `http://doi.acm.org/10.1145/1852786.1852792`, `doi:http://doi.acm.org/10.1145/1852786.1852792`.

[63] Davor Čubranić, Gail C. Murphy, Janice Singer, and Kellogg S. Booth. Hipikat: A project memory for software development. *IEEE Transactions on Software Engineering*, 31(6):446–465, 2005. `doi:http://doi.ieeecomputersociety.org/10.1109/TSE.2005.71`.

[64] Timo Wolf, Adrian Schröter, Daniela Damian, and Thanh Nguyen. Predicting build failures using social network analysis on developer communication. In *Intl Conf on Software Engineering, Vancouver, Canada*, pages 1–11, May 2009. `doi:http://dx.doi.org/10.1109/ICSE.2009.5070503`.

[65] Timo Wolf, Adrian Schröter, Daniela Damian, Lucas D. Panjer, and Thanh H.D. Nguyen. Mining task-based social networks to explore collaboration in software teams. *IEEE Software*, 26(1):58–66, 2009. `doi:http://doi.ieeecomputersociety.org/10.1109/MS.2009.16`.

[66] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll. Predicting source code changes by mining change history. *Software Engineering, IEEE Transactions on*, 30(9):574–586, 2004. Available from: `ieeexplore.ieee.orgabs_all.jsp<http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1324645>`.

[67] Y. Zhang and D. Sheth. Mining software repositories for model-driven development. *Software, IEEE*, 23(1):82 –90, jan.-feb. 2006. `doi:10.1109/MS.2006.23`.

[68] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *Software Engineering, IEEE Transactions on*, 31(6):429 – 445, june 2005. `doi:10.1109/TSE.2005.72`.

[69] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC/FSE '09, pages 91–100, New York, NY, USA, 2009. ACM. Available from: `http://doi.acm.org/10.1145/1595696.1595713`, `doi:http://doi.acm.org/10.1145/1595696.1595713`.

[70] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schroter, and Cathrin Weiss. What makes a good bug report? *IEEE Transactions on Software Engineering*, 36:618–643, 2010. `doi:http://doi.ieeecomputersociety.org/10.1109/TSE.2010.63`.

[71] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting defects for eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, May 2007.