Commits Density

Yegor Bugayenko

Lecture #20 out of 24
80 minutes

Marc J. Rochkind

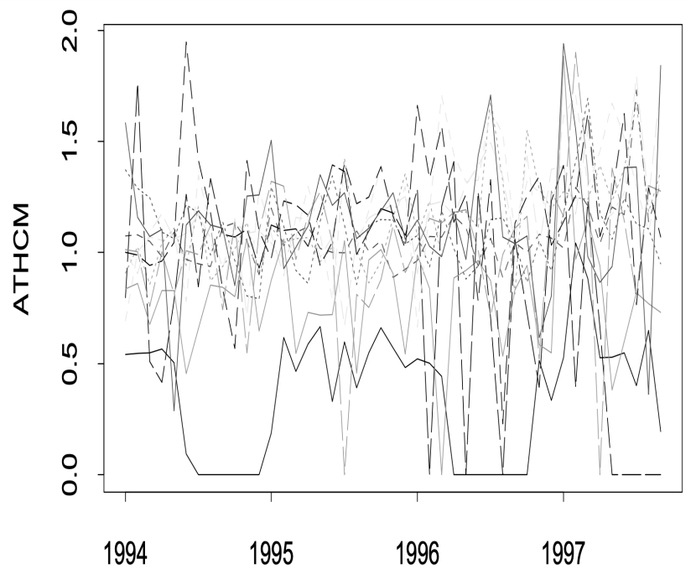
"Where the change was made, that is, to what source lines, and what the change actually was is recorded by the nature of the deltas themselves. The reason for the delta is not recorded automatically; it must be supplied by the programmer adding the delta, but it is required. The quality of the reason (like the quality of the change itself) depends on the conscientiousness of the programmer."

— Marc J. Rochkind. The Source Code Control System. *IEEE Transactions on Software Engineering*, (4):364–370, 1975

Todd Graves

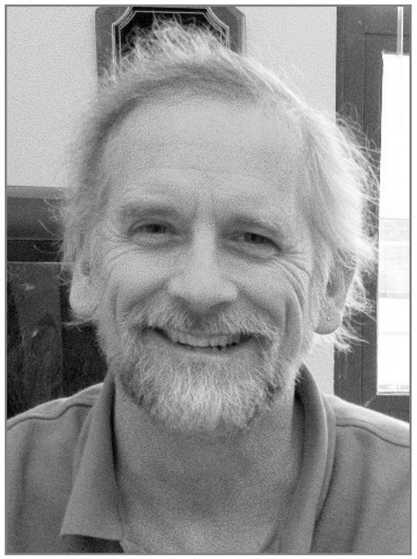
"By studying effort at the level of individual changes, we are able to judge the influence of factors whose contributions are not estimable at a large project level due to aggregation."

— Todd L. Graves and Audris Mockus. Inferring Change Effort from Configuration Management Databases. In *Proceedings of the Fifth International Software Metrics Symposium*, pages 267–273. IEEE, 1998



"Effort, measured in Average Technical Head Count Months (ATHCM), is recorded for each person every month broken down by charging numbers... The eleven developers completed 2794 changes (Maintenance Requests) in the 45 month period under study."

Source: Todd L. Graves and Audris Mockus. Inferring Change Effort from Configuration Management Databases. In *Proceedings of the Fifth International Software Metrics Symposium*, pages 267–273. IEEE, 1998



David Atkins

"Observation is that the change history of a software entity (i.e., the version control data about the modifications to the entity) can be used to estimate the amount of effort a developer expended on a particular modification or set of modifications."

— David Atkins, Thomas Ball, Todd Graves, and Audris Mockus. Using Version Control Data to Evaluate the Impact of Software Tools. In *Proceedings of the 21st International Conference on Software Engineering*, pages 324–333, 1999

Audris Mockus

"Our hypothesis is that a textual description field of a change is essential to understanding why that change was performed. Also, we expect that difficulty, size, and interval would vary strongly across different types of changes."

— Audris Mockus and Lawrence G. Votta. Identifying Reasons for Software Changes Using Historic Databases. In *Proceedings of the International Conference on Software Maintenance*, pages 120–130. IEEE, 2000

Table 1. Result of the MR Classification Algorithm.

	Corrective	Adaptive	Perfective	Inspection	Unclassified	Total
MR	33.8%	45.0%	3.7%	5.3%	12.0%	33171
delta	22.6%	55.2%	4.3%	8.5%	9.4%	129653
lines added	18.0%	63.2%	3.5%	5.4%	9.8%	2707830
lines deleted	18.0%	55.7%	5.8%	10.8%	9.6%	940321
lines unchanged	27.2%	48.3%	4.5%	10.3%	9.6%	328368903

Source: Audris Mockus and Lawrence G. Votta. Identifying Reasons for Software Changes Using Historic Databases. In *Proceedings of the International Conference on Software Maintenance*, pages 120–130. IEEE, 2000

German Daniel

"Developers take care to explain, in each Modification Request (MR), the reason for the change (CVS allows developers to add a log message to every file revision during a CVS commit). The average log for an MR is 300 characters, with a minimum length of 1 (only 8 MRs) and 18 000 for the longest log (which involved the merging of a branch to the main CVS tree)."

— Daniel M. German. Using Software Trails to Reconstruct the Evolution of Software. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6):367–384, 2004

Ahmed E. Hassan

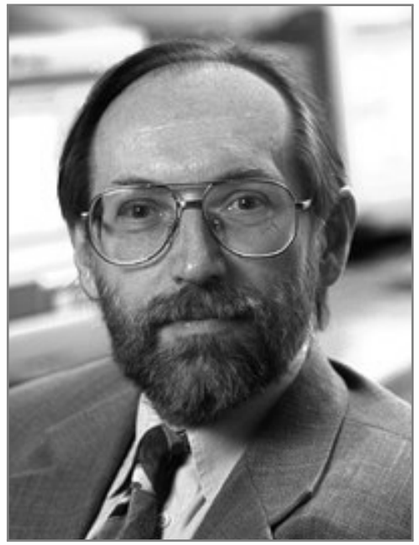
"The Mining Software Repositories (MSR) field is maturing thanks to the rich, extensive, and readily available software repositories... After four successful years as ICSE's largest workshop, MSR became a Working Conference in 2008."

— Ahmed E. Hassan. The Road Ahead for Mining Software Repositories. In *Frontiers of Software Maintenance*, pages 48–57. IEEE, 2008

An Overview of MSR Achievements

- **Understanding Software Systems** "Using the historical sticky notes on the NetBSD system, a large open source operating system, many unexpected dependencies could be easily explained and rationalized."
- **Propagating Changes** "Instead of using traditional dependency graphs to propagate changes, we could make use of the historical co-changes. The intuition is that entities co-changing frequently in the past are very likely to co-change in the future."
- **Predicting and Identifying Bugs** "Tools can flag bugs by recognizing deviations in mined patterns for renaming variables when cloning (i.e., copy-and-paste) code."
- **Understanding Team Dynamics** "Mailing lists discussions could uncover the overall morale of a development team with developers using more optimistic words when they feel positive about the progress of the project."
- **Improving the User Experience** "Instead of studying the quality of the source code, they mine data captured by project monitoring and tracking infrastructures as well as customer support records to determine the expected quality of a software application."
- **Reusing Code** "The techniques locate uses of code such as library APIs, and attempt to match these uses to the needs of a developer working on a new piece of code."
- **Automating Empirical Studies** "The automation permits the repetition of studies on a large number of subject and the ability to verify the generality of many findings in these studies."

Source: Ahmed E. Hassan. The Road Ahead for Mining Software Repositories. In *Frontiers of Software Maintenance*, pages 48–57. IEEE, 2008



Witold Pedrycz

"Results indicate that for the Eclipse data, process metrics are more efficient defect predictors than code metrics... Files with high revision numbers are by nature defect prone... Files that are part of large CVS commits are likely to be defect free."

— Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction. In *Proceedings of the 30th international conference on Software engineering*, pages 181–190, 2008

Table 4. List of Change metrics used in the study.

Metric name	Definition
REVISIONS	Number of revisions of a file
REFACTORINGS	Number of times a file has been refactored[1]
BUGFIXES	Number of times a file was involved in bug-fixing[2]
AUTHORS	Number of distinct authors that checked a file into the repository
LOC_ADDED	Sum over all revisions of the lines of code added to a file
MAX_ LOC_ADDED	Maximum number of lines of code added for all revisions
AVE_ LOC_ADDED	Average lines of code added per revision
LOC_DELETED	Sum over all revisions of the lines of code deleted from a file
MAX_ LOC_DELETED	Maximum number of lines of code deleted for all revisions
AVE_ LOC_DELETED	Average lines of code deleted per revision
CODECHURN	Sum of (added lines of code – deleted lines of code) over all revisions
MAX_ CODECHURN	Maximum CODECHURN for all revisions
AVE_ CODECHURN	Average CODECHURN per revision
MAX_CHANGESET	Maximum number of files committed together to the repository
AVE_CHANGESET	Average number of files committed together to the repository
AGE	Age of a file in weeks (counting backwards from a specific release)
WEIGHTED_AGE	See equation (1)

“Our set of change metrics is obviously only one possible proposal for change metrics we can extract from a CVS repository.”

Source: Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction. In *Proceedings of the 30th international conference on Software engineering*, pages 181–190, 2008

Abdulkareem Alali

"One observation is that the terms that suggest bug related changes are associated with fairly small-sized commits."

— Abdulkareem Alali, Huzefa Kagdi, and Jonathan I. Maletic. What's a Typical Commit? A Characterization of Open Source Software Repositories. In *Proceedings of the 16th International Conference on Program Comprehension*, pages 182–191. IEEE, 2008

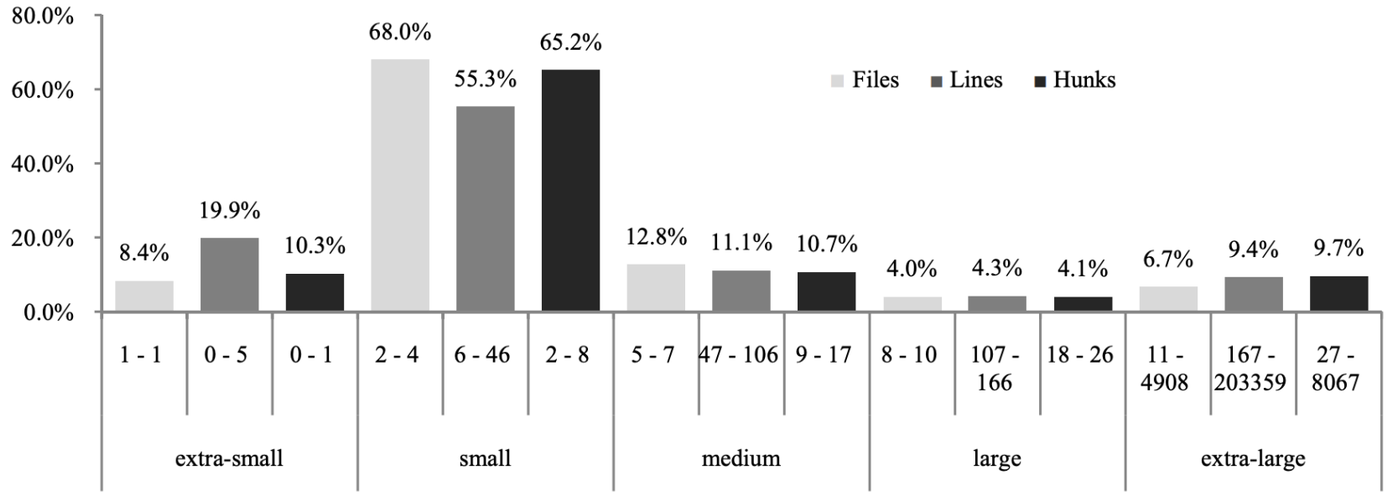


Figure 4. Histogram for the number of commits distributed over number of files, lines, and hunks that were changed for *gcc* (54,536 commits over 8 years).

Source: Abdulkareem Alali, Huzefa Kagdi, and Jonathan I. Maletic. What's a Typical Commit? A Characterization of Open Source Software Repositories. In *Proceedings of the 16th International Conference on Program Comprehension*, pages 182–191. IEEE, 2008

Table 4. Top 25 average term frequency over all nine systems.

Term	Average Rank
fix	36.05%
add	18.22%
test	13.96%
file	11.85%
new	11.54%
support	11.35%
chang	11.14%
bug	10.88%
patch	10.03%
code	9.41%
remov	9.38%
set	8.45%
work	8.24%
updat	7.92%
get	6.75%
error	5.93%
build	5.73%
function	5.57%
typo	4.70%
call	4.66%
messag	4.65%
includ	4.52%
path	4.52%
need	4.49%

“For each project, we collected the log messages and eliminated stop words using the Lovins stemmer algorithm... The result is a ranked list of frequent terms for each project. Then we cross join those nine lists and take the top most 50 frequent terms.”

Source: Abdulkareem Alali, Huzefa Kagdi, and Jonathan I. Maletic. What’s a Typical Commit? A Characterization of Open Source Software Repositories. In *Proceedings of the 16th International Conference on Program Comprehension*, pages 182–191. IEEE, 2008

Abram Hindle

"Large commits were more likely to perfective than corrective, while small changes were more often corrective rather than perfective. In a way it makes sense, correcting errors is surgical, perfecting a system is much more global in scope."

— Abram Hindle, Daniel M. German, and Ric Holt. What Do Large Commits Tell Us? A taxonomical study of large commits. In *Proceedings of the International Working Conference on Mining Software Repositories*, pages 99–108, 2008

Type of Change	Percent
Merge	19.4 %
Feature Addition	19.2 %
Documentation	17.9 %
Add Module	15.8 %
Cleanup	11.9 %
Module Initalization	11.5 %
Token Replacement	9.1 %
Bug	9.0 %
Build	8.9 %
Refactor	8.0 %
Test	6.5 %
External Submission	6.4 %
Legal	6.0 %
Module Move	5.3 %
Remove Module	4.7 %
Platform Specific	4.4 %
Versioning	4.3 %
Source Control System	3.7 %
Indentation and Whitespace	2.3 %
Rename	2.2 %
Internationalization	1.7 %
Branch	1.6 %
Data	0.7 %
Cross cutting concern	0.6 %
Maintenance	0.6 %
Split Module	0.3 %
Unknown	0.3 %
Debug	0.1 %

Table 7: Distribution of commits belonging to each Type of Change over all projects (1 commit might have multiple types).

"We believe that reading a commit log and its diff gives an idea of how easy or difficult it is to maintain a system. For example, several features in PostgreSQL required large commits to be implemented. This is very subjective, but reliable methods could be researched and developed to quantify such effect."

Source: Abram Hindle, Daniel M. German, and Ric Holt. What Do Large Commits Tell Us? A taxonomical study of large commits. In *Proceedings of the International Working Conference on Mining Software Repositories*, pages 99–108, 2008

Oliver Arafat

"We suggest distinguishing commit types by their size, using the following simple heuristic: single commits of 1 to 100 SLoC, aggregate commits of 101 to 10000 SLoC, and repository refactorings of more than 10000 SLoC."

— Oliver Arafat and Dirk Riehle. The Commit Size Distribution of Open Source Software. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*, pages 1–8. IEEE, 2009

Marco D'Ambros

"Developers do not always document all the changes in the commit comment. A common cause is that writing exhaustive comments is time consuming, and—being the last step of a coding session—the necessary time and energy is not always available. Moreover, for commits with many changes, the developers might not remember all of the modifications."

— D'Ambros, Marco and Lanza, Michele and Robbes, Romain. Commit 2.0. In *Proceedings of the 1st Workshop on Web 2.0 for Software Engineering*, pages 14–19, 2010

Raymond Buse

"We present an automatic technique for synthesizing succinct human-readable documentation for arbitrary program differences. We compare our documentation to 250 human-written log messages from 5 popular open source projects. Employing a human study, we find that our generated documentation is suitable for supplementing or replacing 89% of existing log messages that directly describe a code change."

— Raymond P. L. Buse and Westley R. Weimer. Automatically Documenting Program Changes. In *Proceedings of the 25th International Conference on Automated Software Engineering*, pages 33–42, 2010

Jon Eyolfson

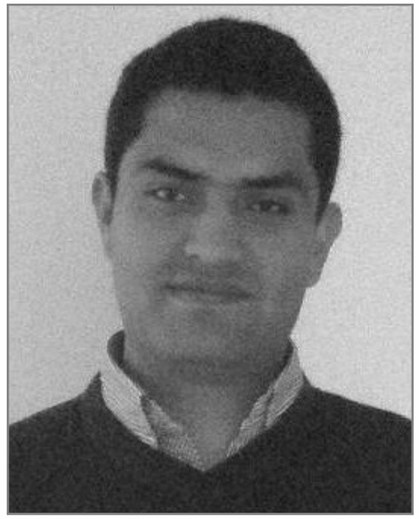
"Commits submitted between midnight and 4 AM (referred to as late-night commits) are significantly buggier and commits between 7 AM and noon are less buggy, implying that developers may want to double-check their own latenight commits."

— Jon Eyolfson, Lin Tan, and Patrick Lam. Do Time of Day and Developer Experience Affect Commit Bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153–162, 2011

Robert Dyer

"First, around 14% of all log messages were completely empty. Second, over two thirds of the messages contained 1–15 words, which is less than the average length of a sentence in English. A normal length sentence in English is 15–20 words (according to various results in Google) and thus we see that very few logs (10%) contained descriptive messages."

— Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale Software Repositories. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 422–431. IEEE, 2013



Luis Fernando Cortés-Coy

"The results of the user study demonstrate that 84% of the generated commit messages do not miss essential information required to understand the changes, 25% of them are concise, and in 39% of the cases the generated messages are easy to read and understand."

— Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. On Automatically Generating Commit Messages via Summarization of Source Code Changes. In *Proceedings of the 14th International Working Conference on Source Code Analysis and Manipulation*, pages 275–284. IEEE, 2014

Shane McIntosh

"JIT models, which aim to predict the commits that will introduce future defects, are typically trained using code-based metrics. We add metrics that estimate the level of detail in commit messages to JIT models with code-based metrics. We find that 43% and 80% of the JIT models of the studied systems are significantly improved by adding metrics that measure commit message volume and content, respectively."

— Jacob G. Barnett, Charles K. Gathuru, Luke S. Soldano, and Shane McIntosh. The Relationship Between Commit Message Detail and Defect Proneness in Java Projects on Github. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 496–499, 2016

Jeongju Sohn

"Age simply measures how long a given program element has existed in the code base. We calculate the age of a given statement as the number of consecutive versions from the faulty version backwards to the latest version containing a modication to the statement."

— Jeongju Sohn and Shin Yoo. FLUCCS: Using Code and Change Metrics to Improve Fault Localization. In *Proceedings of the 26th International Symposium on Software Testing and Analysis*, pages 273–283, 2017

Iftekhar Ahmed

"Commit message quality has an impact on software defect proneness, and the overall quality of the commit messages decreases over time, while developers believe they are writing better commit messages."

— Jiawei Li and Iftekhar Ahmed. Commit Message Matters: Investigating Impact and Evolution of Commit Message Quality. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, pages 806–817. IEEE, 2023

Yuxia Zhang

"Although LLMs can take larger diffs as input, their performance of generating messages leaves much to be improved. UniXcoder tends to generate short messages, while ChatGPT can generate more detailed messages, which are very different from those written by developers."

— Yuxia Zhang, Zhiqing Qiu, Klaas-Jan Stol, Wenhui Zhu, Jiaxin Zhu, Yingchen Tian, and Hui Liu. Automatic Commit Message Generation: A Critical Review and Directions for Future Work. *IEEE Transactions on Software Engineering*, 2024

Pitfalls of Automated Commits Generation

- “Developers indicate that writing the subject of a commit message is hard, and approximately 37% of developers also find writing subjects time-consuming.”
- “The state-of-the-art approaches for automated commit message generation have limited their datasets to commits whose diffs have no more than 100 or 200 tokens. However, only 5% of commits have a diff length of no more than 100 tokens. The performance of four state-of-the-art approaches on commits with larger diffs degrades significantly.”
- “After removing bot-generated and uninformative commit messages from the training and testing datasets, the performance of NNGen, CoRec, and CCRep greatly declines in comparison to the original evaluations.”

Source: Yuxia Zhang, Zhiqing Qiu, Klaas-Jan Stol, Wenhui Zhu, Jiaxin Zhu, Yingchen Tian, and Hui Liu. Automatic Commit Message Generation: A Critical Review and Directions for Future Work. *IEEE Transactions on Software Engineering*, 2024

Commits Best Practices (Coders' Folklore)

- "Commit it as soon as it compiles" — here
- "Prefer small commits to large commits." — here
- "You shouldn't commit based on a time basis, but on a feature basis" — here
- "I tend to commit anytime I take a break" — here
- "If you have to put the word "and" or "also" in your summary, you need to split it up." — here
- "If you can't adequately comment a commit in one line, then it's already too large." — here
- "Files that are generated by build tools, compilers, or other automated processes should not typically be committed." — here
- "The first line must be maximum 50 characters long, each line in the description should though wrap at the 72nd mark." — here

References

Abdulkareem Alali, Huzefa Kagdi, and Jonathan I. Maletic. What's a Typical Commit? A Characterization of Open Source Software Repositories. In *Proceedings of the 16th International Conference on Program Comprehension*, pages 182–191. IEEE, 2008.

Oliver Arafat and Dirk Riehle. The Commit Size Distribution of Open Source Software. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*, pages 1–8. IEEE, 2009.

David Atkins, Thomas Ball, Todd Graves, and Audris Mockus. Using Version Control Data to Evaluate the Impact of Software Tools. In *Proceedings of the 21st International Conference on Software Engineering*, pages 324–333, 1999.

Jacob G. Barnett, Charles K. Gathuru, Luke S. Soldano, and Shane McIntosh. The Relationship Between Commit Message Detail and Defect Proneness in Java Projects on Github. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 496–499, 2016.

Raymond P. L. Buse and Westley R. Weimer. Automatically Documenting Program Changes. In *Proceedings of the 25th International Conference on Automated Software Engineering*, pages 33–42, 2010.

Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. On Automatically Generating Commit Messages via Summarization of Source Code Changes. In *Proceedings of the 14th International Working Conference on Source Code Analysis and Manipulation*, pages 275–284. IEEE, 2014.

D'Ambros, Marco and Lanza, Michele and Robbes, Romain. Commit 2.0. In *Proceedings of the 1st Workshop on Web 2.0 for Software Engineering*, pages 14–19, 2010.

Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. Boa: A Language and Infrastructure for Analyzing Ultra-Large-Scale

Software Repositories. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 422–431. IEEE, 2013.

Jon Eyolfson, Lin Tan, and Patrick Lam. Do Time of Day and Developer Experience Affect Commit Bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153–162, 2011.

Daniel M. German. Using Software Trails to Reconstruct the Evolution of Software. *Journal of Software Maintenance and Evolution: Research and Practice*, 16(6):367–384, 2004.

Todd L. Graves and Audris Mockus. Inferring Change Effort from Configuration Management Databases. In *Proceedings of the Fifth International Software Metrics Symposium*, pages 267–273. IEEE, 1998.

Ahmed E. Hassan. The Road Ahead for Mining Software Repositories. In *Frontiers of Software Maintenance*, pages 48–57. IEEE, 2008.

Abram Hindle, Daniel M. German, and Ric Holt. What Do Large Commits Tell Us? A taxonomical study of large commits. In *Proceedings of the International Working Conference on Mining Software Repositories*, pages 99–108, 2008.

Jiawei Li and Iftekhar Ahmed. Commit Message Matters: Investigating Impact and Evolution of Commit Message Quality. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, pages 806–817. IEEE, 2023.

Audris Mockus and Lawrence G. Votta. Identifying Reasons for Software Changes Using Historic Databases. In *Proceedings of the International Conference on Software Maintenance*, pages 120–130. IEEE, 2000.

Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction. In *Proceedings of the 30th international conference on Software engineering*, pages 181–190, 2008.

Marc J. Rochkind. The Source Code Control System. *IEEE Transactions on Software Engineering*, (4):

364–370, 1975.

Jeongju Sohn and Shin Yoo. FLUCCS: Using Code and Change Metrics to Improve Fault Localization. In *Proceedings of the 26th International Symposium on Software Testing and Analysis*, pages 273–283, 2017.

Yuxia Zhang, Zhiqing Qiu, Klaas-Jan Stol, Wenhui Zhu, Jiaxin Zhu, Yingchen Tian, and Hui Liu. Automatic Commit Message Generation: A Critical Review and Directions for Future Work. *IEEE Transactions on Software Engineering*, 2024.