Ownership at Large

Open Problems and Challenges in Ownership Management

John Ahlgren, Maria Eugenia Berezin, Kinga Bojarczuk, Elena Dulskyte, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Shan He, Ralf Lämmel, Erik Meijer, Silvia Sapora, and Justin Spahr-Summers[*]
Facebook Inc.

ABSTRACT

Software-intensive organizations rely on large numbers of software assets of different types, e.g., source-code files, tables in the data warehouse, and software configurations. Who is the most suitable owner of a given asset changes over time, e.g., due to reorganization and individual function changes. New forms of automation can help suggest more suitable owners for any given asset at a given point in time. By such efforts on ownership health, accountability of ownership is increased. The problem of finding the most suitable owners for an asset is essentially a program comprehension problem: how do we automatically determine who would be best placed to understand, maintain, evolve (and thereby assume *ownership* of) a given asset. This paper introduces the Facebook *Ownesty* system, which uses a combination of ultra large scale data mining and machine learning and has been deployed at Facebook as part of the company's ownership management approach. *Ownesty* processes many millions of software assets (e.g., source-code files) and it takes into account workflow and organizational aspects. The paper sets out open problems and challenges on ownership for the research community with advances expected from the fields of software engineering, programming languages, and machine learning.

KEYWORDS

ownership, machine learning, global software engineering

1 INTRODUCTION

Managing software asset ownership in any organization is important. Many pressing industrial concerns such as security, reliability, and integrity depend crucially on well-defined ownership so that there are clear lines of responsibility for maintenance tasks, code review, incident response, and others. Ownership management requires and connects research on a wide variety of topics including program comprehension, and more generally, software engineering, programming languages, and machine learning.

In this paper, when we refer to (software) 'asset' we include entities as diverse as source-code files, tables in the data warehouse, and software configurations. When we refer to the 'owner' of an asset, we mean this term in a broad sense: a set of people who take responsibility for the asset. The set can be singleton, but may also be a group or sub-organization. The owner can also vary depending on purpose – such as code review versus incident response. If the set was ever empty, the asset is *unowned*. Standard processes, e.g., based on escalation, are typically in place to rule out unowned assets, as they would clearly be a cause for concern. A more nuanced question is the one of 'ownership health', i.e., whether each asset is attributed to the 'most suitable' owner. Who is the most suitable owner of a given asset changes over time, e.g., due to reorganization and individual function changes. Ownership health give rises to interesting research problems and challenges.

Attributing assets to owners and measuring ownership health encompasses a combination of static and dynamic properties of the software assets themselves, the workflows for developing and managing the assets, and the structures of the organization that possesses the assets. As such, the problem of ownership draws on topics from a diverse set of research fields and previously studied problem domains, such as *global software engineering* [11, 13, 14] at the highest level of abstraction through to *program dependence analysis* [7, 42, 45] at the lowest level of abstraction.

The paper outlines the authors' work at Facebook on the problem of ownership management with a focus on ultra large scale data mining and machine learning, subject to collaboration with other teams focusing on additional aspects such as tooling and workflow integration. This work has resulted in the *Ownesty* system, which is introduced in Section 2.

There remain many open challenges and problems that need to be addressed in the more specific context of, for example, reverse engineering, architecture recovery, mining software repositories, process mining, and interpretable models. None of these challenges and problems are specific to the Facebook setting and, in fact, much of the progress in this area can be expected to be achieved in the context of research on open-source ecosystems. Therefore, the

[*]Author order is alphabetical. Contact author: Ralf Lämmel (`rlaemmel@acm.org`). This paper appears in Proceedings of 28th International Conference on Program Comprehension, ICPC 2020. The subject of the paper is covered by the contact author's keynote at the same conference.

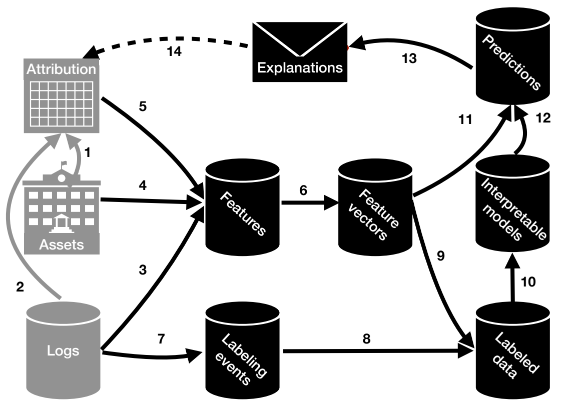

Figure 1: Overall *Ownesty* data flow for ownership recommendation. (See the text for the numbered arrows. The data flow starts from the asset-to-owner attribution mapping on the left.)

paper describes, in Section 3, a set of challenges and open problems for an ongoing research agenda on modern ownership management.

2 THE *OWNESTY* SYSTEM

Let us briefly describe the *Ownesty* system for ownership management, as developed and used at Facebook.

2.1 Vocabulary of Ownership Management

The term *asset* refers to any sort of entity that is a part of a system or is possessed by a company of interest. (We skip over hardware here for simplicity.) Examples of assets are these: a file in the repository for a system, a database that is part of the system, a VM to run the system, or a configuration of the VM. Ownership is also lifted to compound or distributed entities such as components, products, apps, or the scattered implementation of a logging feature.

The term *owner candidate* refers to any sort of individual or group entity which is associated with the system (or company) of interest and could possibly be accountable for any number of assets in this scope. In the work on *Ownesty* at Facebook, we deal with a few types of owner candidates: *individual owner*, *team* (supported by a director), *reporting team* (supported by a manager), and *oncall rotation* (some sort of response team type). There are a few more obscure types that we skip here. In the engineering practice, the types *individual owner* and *oncall rotation* are particularly important.

We assume a special part of a system: its *asset-to-owner attribution mapping* or just *attribution*, which maps assets to owner candidates, which are thus referred to as *owners*. Individuals or processes with appropriate permissions may modify the mapping. In particular, when an asset is mapped to a new owner, then this may be referred to as *ownership transfer*. The main purpose of a system like *Ownesty* is to recommend suitable owners and thereby also to validate ownership health, i.e., the suitability of the currently attributed owners. To this end, machine learning and heuristics are leveraged. Humans may be in the loop for the purpose of confirmation, also depending on the degree of confidence for the available recommendations.

2.2 The ML Architecture

In Figure 1, the arrows denote data flow (computation). The gray shapes and arrows (see on the left) exist regardless of *Ownesty*. Several of the arrows are supported by metadata, which we do not further detail here for brevity.

The gray arrows on the left express that the asset-to-owner attribution mapping is partially encoded in the assets themselves such as by annotation within files or a metastore for tables, in which case extraction can be applied to assets (1) or possibly to logs (2) that record the owners 'in action'.

Ownesty extracts features from the available logs (3) that record some relevant form of interactions between assets and owner candidates. (For instance, a log for a database admin tool would record who was taking what administrative action when.) This is a data and feature engineering challenge because of the plethora of logs and the fact that they were not designed with ownership management in mind. Feature extraction also involves assets and attribution (4–5), e.g., features obtained by source-code analysis. (For instance, we may extract a feature regarding an oncall annotation from a build file.) The individually extracted features are composed into feature vectors (6) – these are specific to the asset type.

Ownesty leverages supervised learning and thus relies on labeled data for positive and negative attribution. To this end, so-called 'labeling events' are extracted from the logs (7), e.g., events that recorded reliable human decisions to accept or reject owner recommendations for attributing assets to owner candidates. The labeled data for training and test is then obtained by joining labeled events with the feature vectors for those events (8–9).

We build interpretable models and provide prediction sets (10–12) for the various asset types. Interpretable or explainable models (e.g., basic decision trees or linear models lifted to scoring systems) are essential because the predictions and the underlying models need to be understood by humans.

Subject to further metadata (e.g., documentation for the features), predictions are mapped to actionable 'explanations' and surfaced through project/ownership management tooling (13) so that humans in the loop can accept or reject, thereby modifying the asset-to-owner attribution mapping (14) (and providing more labeled data eventually).

2.3 Ownership at Large

In this section we describe the scale of ownership management at Facebook, giving some key metrics we use to characterize the asset-owner space covered.

Number of asset types. We distinguish a few hundreds of types. Of course, not every type calls for advanced heuristics or ML for ownership management. The number of asset types is an artifact of the kind of distinctions made. For instance, we do not simply use the type 'database table', but we distinguish different storage engines, as they need to be addressed in different ways, e.g., in terms of ownership signal, in fact, features.

Number of assets of type t. This depends on t, of course. For instance, there are many millions of files under version control; there are millions of tables based on different storage engines; there are several 100k assets for scheduled pipelines in the data warehouse. Even a type with just 10k+ assets may benefit from advanced heuristics or ML, if precision is high and the investment is outweighed by the resulting savings from the automation.

Number of owner types. As discussed, there are two particularly important ones: individual owners and oncall owners.

Number of owner candidates of type t. The number of individual owner candidates translates to the number of employees or engineers or yet other appropriately defined subsets of employees at Facebook; we typically look at several 1k or 10k individual owner candidates. The number of oncall rotations is less than 10k.

Number of (shortlisted) owner candidates for a given asset a. It is often possible, subject to heuristics based on key features, to narrow down to a short list of (3-100) owner candidates that are at all plausible for the asset a and that need to be ranked thus.

Daily churn for asset type t. (That is, the number of assets of type t that are added, deleted, or changed per day.) Such churn is relevant, as it may affect ownership, but see the next metric for deeper insight. For instance, the daily churn for source-code files in one of the bigger mono repos of Facebook is around 100k.

Daily owner churn for asset type t. (That is, the number of assets of type t where the owner is changed per day.) Such churn is relevant as heuristics / ML are supposed to automate (recommend, validate) these changes. For instance, for an important asset type for scheduled pipelines in the data warehouse with about 100k assets, the aggregated daily owner churn over the last 4 months is about 10k while there were several days with several hundreds of affected assets – this is a consequence of prioritized efforts on ownership management at Facebook, based on *Ownesty* or otherwise.

3 OPEN PROBLEMS AND CHALLENGES

We lay out a number of research areas around ownership by describing the open problems and challenges in these areas and referring to related work to capture the state of the art.

3.1 Heterogeneity of Owned Assets

Ownership recommendation – especially when focusing on code assets – is related to *code authorship attribution* [22], as relevant, for example, for detecting malicious or plagiarized code, subject to stylometry methods and the overarching assumption of distinctive writing style to be viewed as a fingerprint.

Ownership recommendation bears also similarities with *reviewer recommendation* [27] which aims at recommending reviewers for new patches (diffs, commits) based on a model built from past patches and possibly other data. For instance, one may extract features such as filenames, module, author, lines deleted, added, number of braces and train a Bayesian Network for recommendations [21]. Reviewer recommendation focuses on code assets – here: files and patches under the regime of version control and code review. We mention in passing that there are many challenges of automating reviewer recommendations at scale [4], e.g., the need for load balancing so that reviewers are not overloaded.

Ownership recommendation needs to address the heterogeneity of asset types such as database tables and software configurations in addition to just code. Even just the 'code type' breaks down into many different subtypes based on language and purpose. Each asset type necessitates specific features. Accordingly, a generic core is needed to be reused across different types and 'patterns' are needed to help onboarding new types. All these features are to be organized and standardized in a manner to convey and leverage similarities across asset types. Further, the multitude of models and the underlying computations need to be managed in an efficient and robust manner.

Within each asset there resides a wealth of information that can be used to determine a suitable owner. Such information has been the topic of study in the program comprehension community for many years. For example, program slicing [24], concept assignment [18], and search-based optimization [17], as well as many other analysis techniques, have all been used to investigate structural components of a software asset to support programmers' comprehension of the asset. The same kind of information can be used to provide features to machine learning, the aim of which is to identify owner candidates who are best-placed to understand the asset in question.

3.2 Dependency Awareness

We cannot look at assets in isolation, but we need to leverage and respect various kinds of dependencies. Let us draw again inspiration from reviewer recommendation with an instance of heterogeneous dependencies, in this case, between regular and library code [36] where such dependencies are aggregated over all developer contributions, thereby essentially aggregating developer experiences which can be considered in addition to 'blame'-based information for finding reviewers. (The cited work relies on a form of token extraction applied to regular and library code; it uses then cosine similarity for comparing aggregated experience of developers with the 'required' experience for patches in need of a review.)

More generally, we need to take into account build dependencies (e.g., a file being generated from another file), usage dependencies (e.g., a database table being consumed by a pipeline), feature mapping (e.g., a logging configuration being associated with a product feature), product mapping (e.g., a collection of assets being shared across products, subject to per-product owners), and requirements to assets mapping. Several of the mentioned dependencies are also version-/variant-specific.

Research is hence needed to integrate ownership management, with various software engineering aspects such as feature location [12], software variability [6], package management and reuse [5], build management [23], traceability recovery [10, 37], change impact analysis [26, 35, 38], and software ecosystems [20, 28, 30–32].

Existing dependency analyses need to be further generalized to apply better to heterogeneous assets and the problem of attributing assets to owners. For instance, provenance or lineage may be aimed at dependencies for data assets while information flow may be aimed at program assets, but combinations or generalizations of such methods are needed to cover the asset types that occur together in practice [1, 9, 43].

3.3 Workflow and Organizational Aspects

Attribution of assets to owners also needs to take into account interactions of owner candidates with the assets. In Section 2, we already discussed the essential use of system logs for ownership recommendation such as interpreting the logged used of a database admin tool as an ownership signal. Let us receive inspiration again from the area of reviewer recommendation, where additional developer-workflow data such as reviewer activity for commits or commenting in an issue tracker are leveraged to identify and rank reviewer candidates [46].

Clearly, ownership recommendation requires a generalization of the analysis of interactions to account for the different types of assets and owner candidates and diverse forms of interaction. Ultimately, the identification of the most suitable owners for assets relies on a deeper understanding of the involved workflows of engineers. For instance, we may take into account project management-based workflow constraints [8]. In this manner, we enter the realm of *process mining* or workflow modeling and encounter the challenging notion of case ID recovery [16, 25, 34].

Human-to-asset and human-to-human interaction and collaboration does not only exercise workflow aspects; it also relates to organizational aspects of ownership management. In this manner, we enter the realm of *global software engineering* [11, 13, 14, 44]. The systematic extraction, integration, and interpretation of all the diverse ownership-related signal (per-asset data, asset dependencies, workflows, organizational structures) calls for *knowledge management* [15], subject to a dedicated knowledge graph [19].

To be more concrete, organizational structure may support better understanding of ownership in that, for example, the health of a particular attribution of an asset to an owner may depend on past, recent, and upcoming changes to teams ('reorganizations') or individual roles. This may suggest future research to revise existing (software engineering) concepts. For example, consider change-impact analysis [26, 35, 38], which needs to be advanced to take into account organizational aspects – the impact of a change depends not *only* on the forward slice of the change locations, but also the owners of those affected assets in the forward slice.

Human aspects of ownership and their interplay with technical aspects provide a rich area of future research. We can expect Computer Supported Collaborative Working (CSCW) [3] and Crowdsourced Software Engineering (CSE) [33] to have a role to play here. Tools for CSCW can be developed or adapted to support ownership, while CSE can contribute a ground-truth approach for ownership decisions used in machine learning.

3.4 Understandable Recommendations

It is important for recommended owners to be 'understandable', thereby entering the realm of interpretable or explainable models in machine learning [40], giving rise to the following options.

Ideally, the ownership model is directly interpretable, as in the case of 'plain' decision trees with some limit on the depth (such as 5). We can also use scoring systems based on supersparse linear integer models [41], even though they require extra effort to deal with correlated features. (We currently favor decision tree-based algorithms in *Ownesty*, but also consider embeddings.) One may also commence with an indirectly interpretable model using, for example, permutation-based feature importance [2].

If we were to give up on interpretable models, we can still maintain that individual predictions (owners) can be directly explained. This is possible, for example, when decision trees (e.g., random forest or gradient boosting) are used. In addition, prediction-specific feature importance [29] can be taken into account. (Adding some sophistication, one can also explain predictions by an interpretation around a given prediction [39].)

When black-box models are used (e.g., embeddings with deep learning), individual predictions can be still explained by using counterfactuals by means of perturbing input features. For instance, an explanation can take the form "Had you touched the file in the last 2 days, you would have been recommended as owner".

The following domain-specific constraints challenge the provision of interpretable and explainable models for ownership recommendation; dedicated research is needed thus:

- The attribution relationship between assets and owner candidates may be intrinsically inconclusive. That is, some assets may be hard to associate with very suitable owners because, for example, the most suitable candidates may just have left the team or the company. Also, some assets may associate with several similarly suitable candidates, which is also problematic in terms of acting on such recommendations; see the next item.
- The process of communicating, discussing, and deciding on ownership is a social one. For instance, ownership recommendations may be subject to rejection, delegation, and escalation – these decisions are not solely based on facts and the resulting limits of feature engineering and explainable predictions need to be explored. (Compare this with image recognition: Human subjects will typically agree on how to distinguish cats and dogs.)
- The introduction of rigorous ownership management is a process as opposed to the installment of a recommendation system. The side effect of a project like *Ownesty* is that one provides highly structured and aggregated information that would not be available otherwise. Those who need to accept or reject recommendations may start to take a dependency on data they would not have had available before. This may lead to 'concept drift' that needs to be addressed by the ML approach.

4 CONCLUSION

This paper characterizes the general notion of ownership management and the specific aspects of using ownership recommendation for attributing assets to owners and measuring the health of any such attribution for large and complex projects and systems. The recommendation of suitable owners and the assessment of ownership health relies on data extracted from assets (per-asset data as well as asset dependencies), workflows and organizational structures. We hope to stimulate interest and activity in this exciting area. We have introduced the Facebook *Ownesty* system to illustrate the practical industrial relevance of the accompanying ownership research agenda. We also set out open problems and challenges and their relationships to existing research activities and communities. We are keen to collaborate with the research communities working on software engineering, programming languages, and machine learning on these open problems and challenges.

REFERENCES

[1] Umut A. Acar, Amal Ahmed, James Cheney, and Roly Perera. 2012. A Core Calculus for Provenance. In *Principles of Security and Trust - First International Conference, POST 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012 (LNCS)*, Vol. 7215. Springer, 410–429.

[2] André Altmann, Laura Tolosi, Oliver Sander, and Thomas Lengauer. 2010. Permutation importance: a corrected feature importance measure. *Bioinformatics* 26, 10 (2010), 1340–1347.

[3] Jerry Andriessen, Michael Baker, and Dan D Suthers. 2013. *Arguing to learn: Confronting cognitions in computer-supported collaborative learning environments*. Vol. 1. Springer Science & Business Media.

[4] Sumit Asthana, Rahul Kumar, Ranjita Bhagwan, Christian Bird, Chetan Bansal, Chandra Shekhar Maddila, Sonu Mehta, and B. Ashok. 2019. WhoDo: automating reviewer suggestions at scale. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019*. ACM, 937–945.

[5] Amine Benelallam, Nicolas Harrand, César Soto-Valero, Benoit Baudry, and Olivier Barais. 2019. The maven dependency graph: a temporal graph-based representation of maven central. In *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019*. IEEE / ACM, 344–348.

[6] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. 2013. A survey of variability modeling in industrial practice. In *The Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13*. ACM, 7:1–7:8.

[7] David Binkley and Mark Harman. 2004. A Survey of Empirical Results on Program Slicing. *Advances in Computers* 62 (2004), 105–178.

[8] Marcelo Cataldo, Audris Mockus, Jeffrey A. Roberts, and James D. Herbsleb. 2009. Software Dependencies, Work Dependencies, and Their Impact on Failures. *IEEE Trans. Software Eng.* 35, 6 (2009), 864–878.

[9] James Cheney, Amal Ahmed, and Umut A. Acar. 2011. Provenance as dependency analysis. *Mathematical Structures in Computer Science* 21, 6 (2011), 1301–1337.

[10] Jane Cleland-Huang, Olly Gotel, and Andrea Zisman (Eds.). 2012. *Software and Systems Traceability*. Springer.

[11] Georgios A. Dafoulas, Cristiano Maia, Almaas Ali, Juan Carlos Augusto, and Victor Lopez Cabrera. 2017. Understanding Collaboration in Global Software Engineering (GSE) Teams with the Use of Sensors: Introducing a Multi-sensor Setting for Observing Social and Human Aspects in Project Management. In *2017 International Conference on Intelligent Environments, IE 2017*. IEEE, 114–121.

[12] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process* 25, 1 (2013), 53–95.

[13] Christof Ebert, Marco Kuhrmann, and Rafael Prikladnicki. 2016. Global Software Engineering: An Industry Perspective. *IEEE Software* 33, 1 (2016), 105–108.

[14] Christof Ebert, Marco Kuhrmann, and Rafael Prikladnicki. 2016. Global Software Engineering: Evolution and Trends. In *11th IEEE International Conference on Global Software Engineering, ICGSE 2016*. IEEE, 144–153.

[15] John Girard and JoAnn Girard. 2015. Defining knowledge management: Toward an applied compendium. *Online Journal of Applied Knowledge Management* 3, 1 (2015).

[16] Sukriti Goel, Jyoti M. Bhat, and Barbara Weber. 2013. End-to-End Process Extraction in Process Unaware Systems. In *Business Process Management Workshops - BPM 2012 International Workshops. Revised Papers (Lecture Notes in Business Information Processing)*, Vol. 132. Springer, 162–173.

[17] Mark Harman. 2007. Search Based Software Engineering for Program Comprehension. In *15th International Conference on Program Comprehension (ICPC 2007)*. IEEE, 3–13.

[18] Mark Harman, Nicolas Gold, Robert Mark Hierons, and David Binkley. 2002. Code Extraction Algorithms which Unify Slicing and Concept Assignment. In *IEEE Working Conference on Reverse Engineering (WCRE 2002)*. IEEE, 11 – 21.

[19] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2020. Knowledge Graphs. arXiv:2003.02320 [cs.AI]

[20] Slinger Jansen and Michael A. Cusumano. 2012. Defining Software Ecosystems: A Survey of Software Platforms and Business Network Governance. In *Proceedings of the Forth International Workshop on Software Ecosystems (CEUR Workshop Proceedings)*, Vol. 879. CEUR-WS.org, 40–58. http://ceur-ws.org/Vol-879

[21] G. Jeong, S. Kim, T. Zimmermann, and K. Yi. 2009. Improving code review by predicting reviewers and acceptance of patches. (2009), 18 pages. Research on Software Analysis for Error-free Computing Center Tech-Memo ROSAEC MEMO 2009-006.

[22] Vaibhavi Kalgutkar, Ratinder Kaur, Hugo Gonzalez, Natalia Stakhanova, and Alina Matyukhina. 2019. Code Authorship Attribution: Methods and Challenges. *ACM Comput. Surv.* 52, 1 (2019), 3:1–3:36.

[23] Gabriël Konat, Sebastian Erdweg, and Eelco Visser. 2018. Scalable incremental building with dynamic task dependencies. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018*. ACM, 76–86.

[24] Bogdan Korel and Jurgen Rilling. 1997. Dynamic Program Slicing in Understanding of Program Execution. In 5^{th} *IEEE International Workshop on Program Comprenhesion (IWPC'97)*. IEEE, 80–89.

[25] Ralf Lämmel, Alvin Kerber, and Liane Praza. 2020. Understanding What Software Engineers Are Working on – The Work-Item Prediction Challenge. In *Proceedings of the 28th IEEE/ACM International Conference on Program Comprehension (ICPC Industry Track)*. IEEE / ACM.

[26] Bixin Li, Xiaobing Sun, Hareton Leung, and Sai Zhang. 2013. A survey of code-based change impact analysis techniques. *Softw. Test., Verif. Reliab.* 23, 8 (2013), 613–646.

[27] Jakub Lipcak and Bruno Rossi. 2018. A Large-Scale Study on Source Code Reviewer Recommendation. In *44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2018*. IEEE, 378–387.

[28] Yaxin Liu, Peng He, Gaoyan Wu, and Yilu Li. 2017. Towards Understanding Developers' Collaborative Behavior in Open Source Software Ecosystems. *JSW* 12, 6 (2017), 393–405.

[29] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*. NIPS, 4765–4774.

[30] Mircea Lungu, Romain Robbes, and Michele Lanza. 2010. Recovering inter-project dependencies in software ecosystems. In *ASE 2010, 25th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 309–312.

[31] Konstantinos Manikas. 2016. Revisiting software ecosystems Research: A longitudinal literature study. *Journal of Systems and Software* 117 (2016), 84–103.

[32] Konstantinos Manikas and Klaus Marius Hansen. 2013. Software ecosystems - A systematic literature review. *Journal of Systems and Software* 86, 5 (2013), 1294–1306.

[33] Ke Mao, Licia Capra, Mark Harman, and Yue Jia. 2017. A survey of the use of crowdsourcing in software engineering. *Journal of Systems and Software* 126 (2017), 57–84.

[34] Hamid R. Motahari Nezhad, Boualem Benatallah, Régis Saint-Paul, Fabio Casati, and Periklis Andritsos. 2008. Process spaceship: discovering and exploring process views from event logs in data spaces. *PVLDB* 1, 2 (2008), 1412–1415.

[35] Denys Poshyvanyk, Andrian Marcus, Rudolf Ferenc, and Tibor Gyimóthy. 2009. Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering* 14, 1 (2009), 5–32.

[36] Mohammad Masudur Rahman, Chanchal K. Roy, and Jason A. Collins. 2016. CoRReCT: code reviewer recommendation in GitHub based on cross-project and technology experience. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Companion Volume*. ACM, 222–231.

[37] Patrick Rempel, Patrick Mäder, Tobias Kuschke, and Ilka Philippow. 2013. Requirements Traceability across Organizational Boundaries - A Survey and Taxonomy. In *Requirements Engineering: Foundation for Software Quality - 19th International Working Conference, REFSQ 2013 (LNCS)*, Vol. 7830. Springer, 125–140.

[38] Xiaoxia Ren, Barbara G. Ryder, Maximilian Störzer, and Frank Tip. 2005. Chianti: a change impact analysis tool for Java programs. In *27th International Conference on Software Engineering (ICSE 2005)*. ACM, 664–665.

[39] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1135–1144.

[40] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* 1 (05 2019), 206–215.

[41] Cynthia Rudin and Berk Ustun. 2018. Optimized Scoring Systems: Toward Trust in Machine Learning for Healthcare and Criminal Justice. *Interfaces* 48, 5 (2018), 449–466.

[42] Josep Silva. 2012. A Vocabulary of Program Slicing-Based Techniques. *Comput. Surveys* 44, 3 (June 2012), 12:1 – 12:48.

[43] Xuan Sun, Xin Gao, Huiying Du, and Wei Ye. 2016. A Query Language of Data Provenance Based on Dependency View for Process Analysis. In *The 28th International Conference on Software Engineering and Knowledge Engineering, SEKE 2016*. KSI Research Inc. and Knowledge Systems Institute Graduate School, 110–113.

[44] Yi Wang and David F. Redmiles. 2016. Cheap talk, cooperation, and trust in global software engineering - An evolutionary game theory model with empirical support. *Empirical Software Engineering* 21, 6 (2016), 2233–2267.

[45] Mark Weiser. 1979. *Program slices: Formal, psychological, and practical investigations of an automatic program abstraction method*. Ph.D. Dissertation. University of Michigan, Ann Arbor, MI.

[46] Yue Yu, Huaimin Wang, Gang Yin, and Tao Wang. 2016. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? *Information & Software Technology* 74 (2016), 204–218.