Weighted Multi-Factor Multi-Layer Identification of Potential Causes for Events of Interest in Software Repositories

Philip Makedonski, Jens Grabowski
Institute of Computer Science, University of Göttingen
Goldschmidtstr. 7, 37077 Göttingen, Germany
{makedonski,grabowski}@cs.uni-goettingen.de

Abstract

Change labelling is a fundamental challenge in software evolution. Certain kinds of changes can be labeled based on directly measurable characteristics. Labels for other kinds of changes, such as changes causing subsequent fixes, need to be estimated retrospectively. In this article we present a weight-based approach for identifying potential causes for events of interest based on a cause-fix graph supporting multiple factors, such as causing a fix or a refactoring, and multiple layers reflecting different levels of granularity, such as project, file, class, method. We outline different strategies that can be employed to refine the weights distribution across the different layers in order to obtain more specific labelling at finer levels of granularity.

1 Introduction

The field of software mining explores different approaches for extracting information from software repositories both in the form of basic facts and in the form of derived knowledge. While software repositories provide a wealth of information related to the development and evolution of software projects, most of it is of empirical nature, that is, describing consequences rather than causes. For example, developers typically describe their development and maintenance activities as *fixing* issues and problems, *improving* certain properties, *adding* features and functionality, and *refactoring* code. In contrast, during software assessment, we are often more interested in the potential causes for such activities which are typically not explicitly labelled as such due to the fact that such knowledge is usually not available at the time when the corresponding activity was performed.

In this article, we are concerned with activities which are associated with contributing to various technical risks for undesirable phenomena, such as failures, or difficult to maintain code that needs refactoring. We explore means for the retrospective identification and quantification of such activities based on empirical data and different factors contributing to labelling activities as risky. The quantitative information in the form of weights provides a more refined view on the extent to which an activity can be considered a technical risk. The presented approach can be generalised to labelling activities as potential causes for events of interest with respect to any particular assessment task, regardless of whether it is concerned with a technical risk or not.

Existing approaches are typically based on some form of origin analysis [GT02], involving line-tracking and annotation graphs [KZPW06], line histories [CC06], line mapping [MHC14], as well as several refinements to these [WS08, CCDP09] in order to map and track entities across revisions. Different applications for such approaches have been discussed in the literature, ranging from finding fix-inducing changes [SZZ05] and the role of authorship on implicated code [RD11] to defect-insertion circumstance analysis [PP14]. While these are closely related to the topic of this article, to our knowledge none of the exist-

In: A.H. Bagge, T. Mens (eds.): Postproceedings of SATToSE 2015 Seminar on Advanced Techniques and Tools for Software Evolution, University of Mons, Belgium, 6-8 July 2015, published at `http://ceur-ws.org`

ing approaches has incorporated weighting of the extent to which a change contributes to a subsequent fix. The weighting information can be used to refine and improve existing applications, such as better targeted recommendations for artifacts that need additional review or testing.

This article is structured as follows: In Section 2 we outline the basic notions related to our approach. In Section 3 we discuss the weighting approach and its generalisation for arbitrary factors. In Section 4 we refine the approach to cover multiple levels of abstraction across distinct layers. Then, in Section 5, we discuss different strategies for distributing the weights across the layers. Section 6 summarises related work. Finally, we conclude with a short summary and outlook in Section 7.

2 Causes and Fixes

In this article we are concerned with determining the likely causes for events of interest. Before we proceed, we need to establish what we consider under "events of interest" and other related notions:

Artifact: A generalised notion of a software-related entity a at any level of granularity, such as project, file, class, method, on which developers perform development and maintenance activities. An artifact may contain other artifacts at finer levels of granularity.

State: A generalised notion of a revision a_t of artifact a at a point in time t. The set of all states of an artifact a is denoted as A.

Event of interest: A state a_t of an artifact a at a point in time t which can be described by some quantitative or qualitative characteristic *factor*, such as the content of a descriptive message associated with the state.

Fix: A modification to an existing part of an artifact a in a given state a_t, that was last modified or created at an earlier point in time $t - n$ resulting in a state a_{t-n}. The modification may, but does not strictly need to, relate to fixing a problem.

Cause: A modification of a part of an artifact a at a given state a_t that was modified at a later point in time $t + n$ resulting in a state a_{t+n}.

Cause-Fix Relationship: A relationship between two states (a_{t-n}, a_t) of an artifact a, where a part of a that was modified in a_{t-n} was subsequently modified in a later state a_t, hence a_{t-n} is considered a cause for a_t. It is denoted as $a_{t-n} \xrightarrow{causes} a_t$.

Cause-Fix Graph: A hierarchical directed graph $G = (N, E)$, where the set of nodes N includes representations for each state of each artifact. A state may contain other states at finer levels of granularity, i.e. $c_t \xrightarrow{contains} a_t$, based on the containment relationships between the corresponding artifacts for the states (assuming that artifact c contains artifact a, i.e. $c \xrightarrow{contains} a$). For example, the state for a class may contain also states for methods modified at the same time as the class. The set of directed edges E includes representations for each cause-fix relationship between two states of an artifact.

Based on the cause-fix relationships, for a given state a_t identified as a fix, we define the set of states fixed by a_t (i.e. the set of causes for a_t) as:

$$a_t^{\text{FIXES}} = \{a_{t-n} \in A : a_{t-n} \xrightarrow{causes} a_t\} \qquad (1)$$

Conversely, for a given state a_{t-n} identified as a cause, the set of known caused fixes for a_{t-n} is defined as:

$$a_{t-n}^{\text{CAUSES}} = \{a_t \in A : a_{t-n} \xrightarrow{causes} a_t\} \qquad (2)$$

A cause-fix graph can be constructed by utilising information extracted from version control systems. This can be accomplished automatically by applying any of the approaches for tracking the location of modified fragments across revisions already described in the literature [WS08, CCDP09] and transforming their output. The resulting graph at the project (or global) level of abstraction represents the cause-fix relationships between states of the whole project. An example for such a graph for five states of a project p (p_1 to p_5) is shown on Figure 1.

3 Weights and Factors

A simplified binary classification of nodes in the graph as causes for events of interest presents some limitations. The basic example from Figure 1 already raises two questions related to the significance of the classifications:

- Given that both p_3 and p_4 are identified as causes for the fix in p_5, are they both equally likely causes and thus to be considered of equal importance?
- Given that p_3 is identified as causing both p_4 and p_5, is it then considered a less likely cause for p_5, and thus to be considered of less importance?

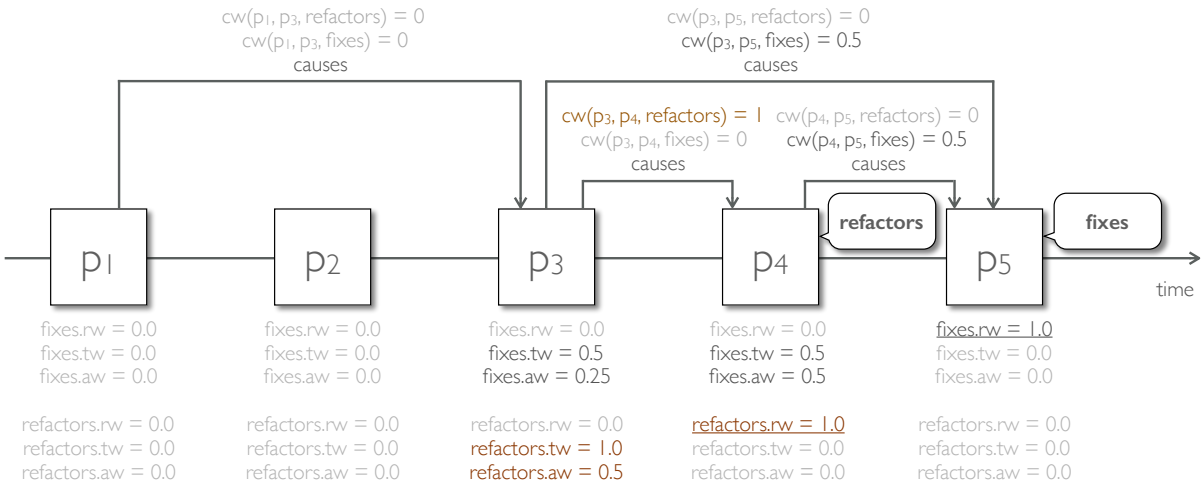

Figure 1: Multi-factor cause-fix graph example

In order to be able to reason about these questions, we need means to quantify the relationships between fixes and causes. We can establish that cause-fix relationships are many-to-many, that is a revision may be the cause for many subsequent revisions, and a revision may fix multiple previous revisions. Conceptually, we consider a fix as an activity that is "removing a weight" from a state of an artifact. Consequently the activities that contributed to the causes for the fix "added weight" to the corresponding states of the artifact. Our approach to quantifying the degree to which a revision can be considered as the cause for another revision is based on this conceptual premise. In addition, there may be different types of "weights" based on different characteristics of the fixing revision, e.g. "fixing an issue", "refactoring code", etc., reflecting the different kinds of events of interest. In order to accommodate this, we extend the notion to "removing a weight related to a weight factor *wf*" where $wf \in \{\text{fixes}, \text{refactors}, \ldots\}$. Thus, we speak of a fixing revision a_t as having *removed weight* (rw) with respect to weight factor wf where:

$$rw(a_t, wf) = \begin{cases} 1 & \text{if } wf \text{ property holds for } a_t \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

Each of the causes a_{t-n} can be regarded as contributing to that weight, thus for each cause-fix relationship $a_{t-n} \xrightarrow{\text{causes}} a_t$ and for each weight factor wf, we define the notion of *contributed weight* (cw) of a causing revision a_{t-n} to a fixing revision a_t with regard to a weight factor wf as:

$$cw(a_{t-n}, a_t, wf) = rw(a_t, wf) \frac{1}{|a_t^{\text{FIXES}}|} \quad (4)$$

For each fix a_t caused by a causing state a_{t-n}, the causing state a_{t-n} is then said to accumulate a *total weight* (tw) with regard to weight factor wf, defined as:

$$tw(a_{t-n}, wf) = \sum_{a_t \in a_{t-n}^{\text{CAUSES}}} cw(a_{t-n}, a_t, wf) \quad (5)$$

For example, the fix in p_5 is removing a weight $rw(\text{p}_5, \text{fixes}) = 1$ with respect to the weight factor "fixes". In the set P denoting all states of p, there are two states $\text{p}_5^{\text{FIXES}} = \{\text{p}_{5-n} \in \text{P} : \text{p}_{5-n} \xrightarrow{causes} \text{p}_5\} = \{\text{p}_3, \text{p}_4\}$ identified as causes for this fix. They are considered to be contributing equally to that weight. In this case, each cause-fix relationship is contributing a weight $cw(\text{p}_{5-n}, \text{p}_5, \text{fixes}) = 0.5$. On the other hand, p_4 can be considered "neutral" with respect to the "fixes" weight factor (i.e. $rw(\text{p}_4, \text{fixes}) = 0$), as it is not identified as an event of interest. Hence, p_3 does not contribute any weight to p_4 (i.e. $cw(\text{p}_3, \text{p}_4, \text{fixes}) = 0$). In this case, we speak of p_3 and p_4 as having a $tw(\text{p}_3, \text{fixes}) = tw(\text{p}_4, \text{fixes}) = 0.5$. Thus, at first glance it may seem that p_3 and p_4 can be considered equally important.

In order to reason about the second question, we need to contemplate the inverse relationship. Considering p_3 in the example, it causes both p_4 and p_5, i.e. $\text{p}_3^{\text{CAUSES}} = \{\text{p}_{3+n} \in \text{P} : \text{p}_3 \xrightarrow{causes} \text{p}_{3+n}\} = \{\text{p}_4, \text{p}_5\}$, whereas p_4 only causes p_5, i.e. $\text{p}_4^{\text{CAUSES}} = \{\text{p}_5\}$. To

take this into account, we define the notion of *average weight* (aw) with regard to weight factor wf as:

$$aw(a_{t-n}, wf) = \frac{tw(a_{t-n}, wf)}{|a_{t-n}^{\text{CAUSES}}|} \quad (6)$$

In the example above, this yields $aw(\text{p}_3, \text{fixes}) = 0.25$ and $aw(\text{p}_4, \text{fixes}) = 0.5$, respectively. Thus, we can state that while both p_3 and p_4 can be considered important as causes for the fix in p_5 with respect to the weight factor "fixes", since p_3 is also a cause for p_4, it is less important than p_4 as it also caused a "neutral" change with respect to the weight factor "fixes" in addition to the fixing change. If we consider the "refactors" weight factor, we observe that the weights are distributed differently since it is p_4 where the weight related to that factor is removed ($rw(\text{p}_4, \text{refactors}) = 1$) and hence p_3 is the only identified cause contributing all the removed weight ($cw(\text{p}_3, \text{p}_4, \text{refactors}) = tw(\text{p}_3, \text{refactors}) = aw(\text{p}_3, \text{refactors}) = 1$). The corresponding weighting is also shown in Figure 1. The weight-related values are calculated for each weight factor for each node. Note, that while information about the causing revisions can be considered definitive, information about the fixing revisions is only partially known. Future revisions may still include fixes for existing revisions, thus altering their weights.

4 Layers and Granularities

In the examples discussed so far, only the project level of granularity was considered. In practice, a revision at the project level of granularity can be decomposed to revisions at the file and logical levels of granularity, where multiple related artifacts at these levels are changed together as part of a development activity. In this case, the challenge of transferring weights between the different levels arises. Furthermore, while a set of related artifacts may be changed within a causing revision, only a subset of these artifacts and possibly a set of additional artifacts may be changed within a corresponding fixing revision. Thus, the causes and fixes for a revision of an artifact at a finer level of granularity may be a subset of the causes and fixes for the containing artifact. Consequently, the weight distribution may vary across the different levels of granularity. This raises two fundamental challenges:

- Given a revision that is the cause for a fix, where the cause affects multiple artifacts at a finer level of granularity, are all of these artifacts contributing equally to the cause for the fix?
- Given a revision that is considered a fix, which affects multiple artifacts at a finer level of granularity, are all these artifacts equally important for the fix?

To illustrate the first challenge, consider a different scenario, sketched in Figure 2. In this scenario, there are three files, x, y, and z, two of which are modified as part of p_3, p_4, p_5 on the project level of granularity. There are two states at the file level for each state at the project level of granularity. The naive approach would be to simply copy the weights from the project level to the file level. With regard to the first challenge, the question arises whether the states y_3 and y_4 at the file level are contributing at all to the cause for the fix in p_5, given that in p_5 only x and z have been modified. In other words, shall y_3 and y_4 be assigned any weights at all? The same is also applicable at the logical level.

Even from this simplified example, we can observe that the naive copy approach can potentially result in a lot of noise since the sets of states of artifacts at a finer level of granularity may vary between the causing and the fixing states at the coarser level of granularity. A more adequate approach is to construct a distinct cause-fix graph at each layer corresponding to a given level of granularity based on the cause-fix relationships among the states at that level. This enables weight redistribution within the corresponding layers, yielding more accurate weighting for each layer. Consider the same scenario from Figure 2, where instead of copying the weights from the project layer, we calculate the weights at the file layer based only on the cause-fix relationships at that layer, as illustrated in Figure 3. This approach yields more accurate weight distribution, taking into account that only x and z were modified as part of the fix in p_5. Hence, the corresponding states x_3 and z_4 carry the full responsibility for causing the fix in p_5 and thus shall be assigned the corresponding weights, whereas the states y_3 and y_4 can be considered neutral in this case and shall be assigned no weights at all.

This brings us to the second challenge, which can be exemplified in the given scenario as follows: given that both states x_5 and z_5 at the file level are considered as part of the fix in p_5 at the project level, are both x_5 and z_5 contributing equally to the fix in p_5? So far, states at finer levels of granularity simply inherited the removed weights from the containing state at a coarser level of granularity, that is $rw(\text{x}_5, \text{fixes}) = rw(\text{z}_5, \text{fixes}) = rw(\text{p}_5, \text{fixes})$. Inheriting the removed weights from the containing state does not take into account potential dilution of the contribution of each individual state at the finer level of granularity. If there is a single state at the finer level of granularity, it can be considered solely responsible for the fix, but if there are a large number of states at the finer level of granularity, each one of them may be contributing only a small part to the fix.

Even in this simple artificial scenario, we need to account for both the number of states at a finer level

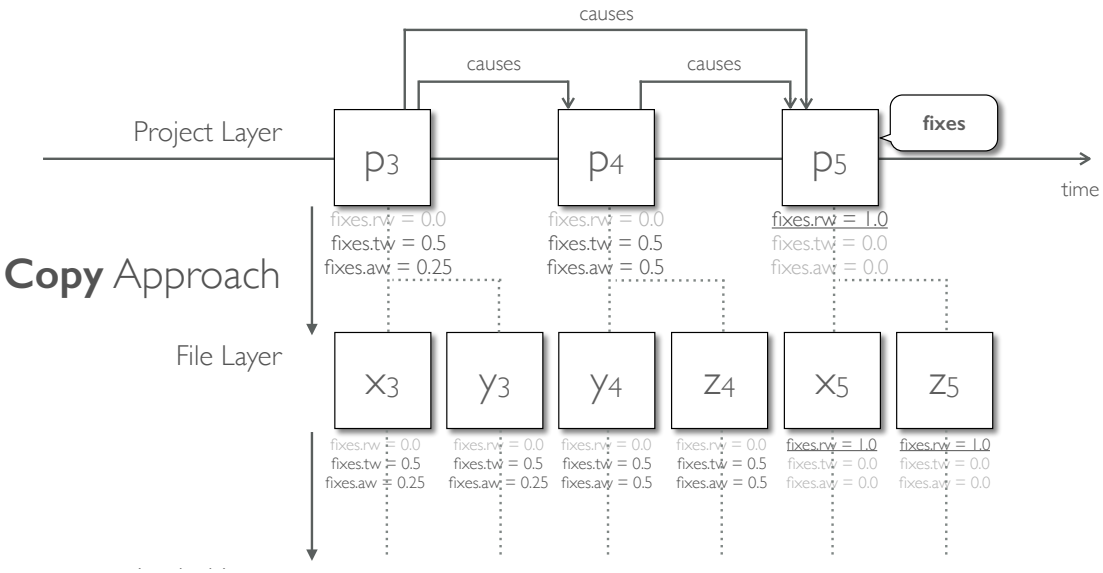

Figure 2: Copy approach for distributing weights across different levels of granularity

of granularity involved in a fix and potentially also other characteristics of each state in order to obtain a more accurate picture. This raises some concerns that need to be taken into account, such as the following:

- Does the number of states of artifacts at a finer level of granularity involved in a fix dilute the contribution of each individual state to the fix?
- Do states of certain types of artifacts contribute more to a fix than others (e.g. states of code vs. image artifacts)?
- Do states of larger artifacts contribute more to a fix than states of smaller artifacts?
- Do states of artifacts containing larger changes contribute more to a fix than states of artifacts containing smaller changes?

In order to take these concerns into account in the weighting approach, we define different weight distribution strategies, which distribute removed weights in fixing states across artifact states at finer levels of granularity depending on their contribution to a fix. Consequently, the weights calculated for the causing states are also updated according to the strategy being used.

5 Weight Distribution Strategies

As noted in Section 4, when we consider the contribution of each state of an artifact at a finer level of granularity to a fix in a state of a containing artifact at a coarser level of granularity, we need to take different aspects into account, such as the number of states at the finer level of granularity involved in the fix, the type and size of the corresponding artifacts, as well as the amount of change to each corresponding artifact. To address these concerns, we exemplify four weight distribution strategies. Additional strategies may be added to emphasise the importance of other characteristics of corresponding artifacts, such as their complexity, documentation availability, etc. The weight distribution strategies refine the notion of *removed weight* (rw) to *distributed removed weight* (drw). The distributed removed weight according to a distribution strategy ds for a state a_t of artifact a contained in a state c_t of containing artifact c is defined based on the following expression:

$$drw(a_t, wf, ds) = rw(c_t, wf) \cdot df(a_t, ds) \qquad (7)$$

where the *distribution factor* for a distribution strategy ds ($df(ds)$) determines the proportion of the removed weight from the containing state c_t allocated to the contained state a_t according to the distribution strategy of choice. As a baseline, the distribution factor for the *inherit* strategy discussed in Section 4 and shown in Figure 3 can be defined as:

$$df(a_t, \text{inherit}) = 1 \qquad (8)$$

Substituting the removed weight with the distributed removed weight in the calculation of the con-

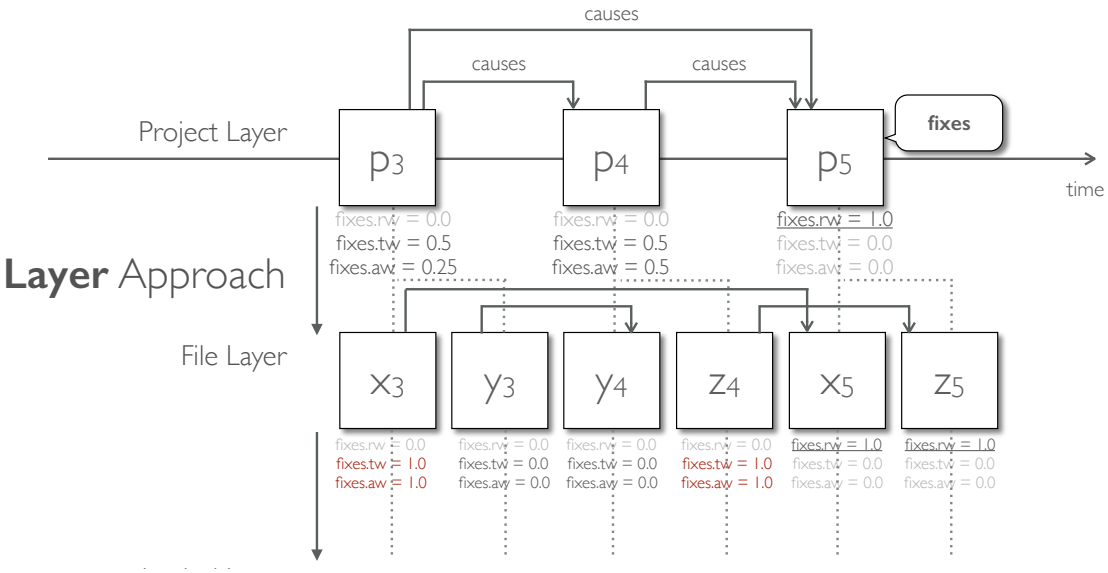

Figure 3: Layer approach for distributing weights across different levels of granularity

tributed weights enables the support for distributed removed weights according to a given strategy throughout the approach.

5.1 Shared Strategy

The *shared* strategy takes into account number of states of artifacts at a finer level of granularity involved in a fix based on the assumption that a large number of states dilutes the contribution of each individual state to the fix. This strategy distributes the removed weight equally, assuming that each state at a finer granularity contributes equally to the fix. As a consequence, the more states contributing to a fix the less impact each individual state has. Given the set of states at a finer level of granularity contained in a state c_t, defined as:

$$c_t^{\text{CONTENTS}} = \{a_t : c_t \xrightarrow{contains} a_t\} \tag{9}$$

the distribution factor for the *shared* strategy is defined as:

$$df(a_t, \text{shared}) = \frac{1}{|c_t^{\text{CONTENTS}}|} \tag{10}$$

The application of the *shared* strategy to the running example from Figures 2–3 and the resulting weight redistribution is shown in Figure 4. Since two states at the file level of granularity are involved in the fix at the project level of granularity, the $df(\text{x}_5, \text{shared}) = df(\text{z}_5, \text{shared}) = 0.5$ and hence $drw(\text{x}_5, \text{fixes}, \text{shared}) = drw(\text{z}_5, \text{fixes}, \text{shared}) = 0.5$. Consequently, the total and average weights of the corresponding causing states at the file level of granularity are also adjusted. Thus, the dilution of the contribution of each state at the finer level of granularity to the fix is also extended to the total and average weights of the corresponding causing states.

While we exemplify only the application of the strategy to the project and file levels of granularity, this strategy is also applicable at different logical levels of granularity. Note, however, that it shall be applied at each logical level of granularity (e.g. Class, Method, Function, etc.) separately, which makes its application at that level more similar to the *type* strategy.

5.2 Type Strategy

The *type* strategy takes into account how much states of artifacts at a finer level of granularity contribute to a fix based on the artifact type (at) of the corresponding artifact. This strategy distributes the removed weight equally among states of artifacts of a selected type (indicated as a parameter), while states of artifacts of other types do not get any removed weight assigned. It can be used to emphasise the importance of states of code artifacts and de-emphasise the importance of image artifacts, for example. The distribution factor for the *type* strategy for a given type T is defined as:

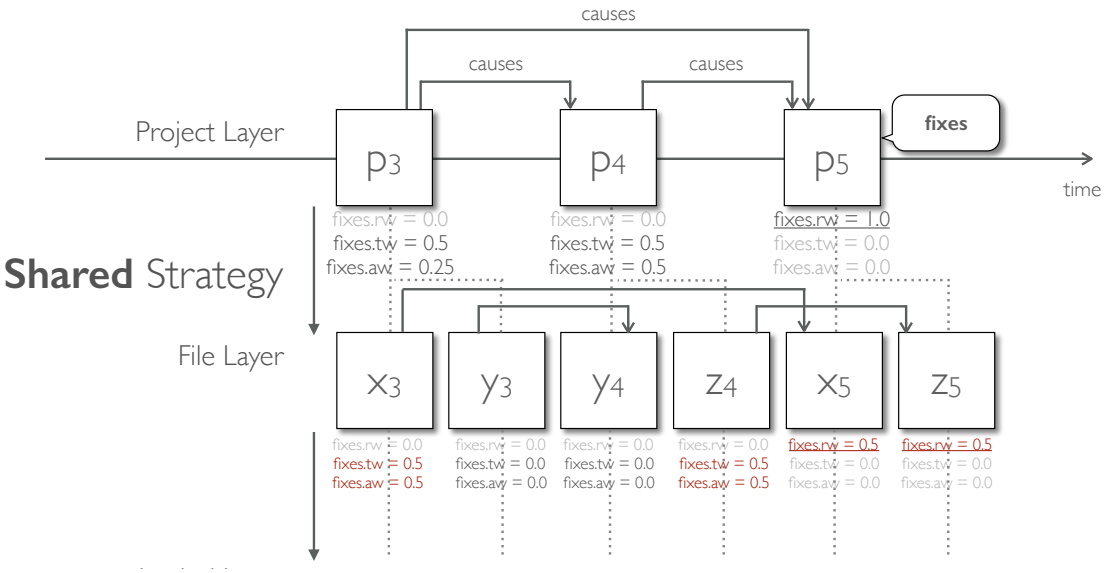

Figure 4: Shared strategy for distributing removed weights across layers

$$df(a_t, \text{type:}T) = \begin{cases} \frac{1}{|\{s_t \in c_t^{\text{CONTENTS}} : at(s_t) = T\}|} & \text{if } at(a_t) = T \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

where $\{s_t \in c_t^{\text{CONTENTS}} : at(s_t) = T\}$ denotes the set of states of artifacts of type T contained in c_t.

The application of the *type* strategy for the type *code* to the running example from Figures 2–4 and the resulting weight redistribution is shown in Figure 5. Of the two states at the file level of granularity involved in the fix at the project level of granularity, only x_5 is of type *code*, hence $df(x_5, \text{type:code}) = 1$, whereas $df(z_5, \text{type:code}) = 0$ since $at(z_5) = image$. Consequently, $drw(x_5, \text{fixes}, \text{type:code}) = 1$, whereas $drw(z_5, \text{fixes}, \text{type:code}) = 0$. The total and average weights of the corresponding causing states at the file level of granularity are adjusted respectively. Thus, the emphasis on the contribution of states of code artifacts to the fix is also extended to the total and average weights of the corresponding causing states.

This strategy can be applied multiple times for different types of artifacts, essentially resulting in a distribution of removed weights "within type", i.e. the removed weight of a fixing state at the project level of granularity is distributed once among all states of code artifacts, then again independently among all states of test artifacts, and so on. In a similar manner, it can also be applied at the different logical levels of granularity (e.g. Class, Method, Function, etc.) individually in order to obtain the equivalent of the *shared* strategy at the file level of granularity applied at the logical levels of granularity.

5.3 Size Strategy

The *size* strategy emphasises the impact of the size of an artifact (as) in a given state that is considered as a part of a fixing state at a coarser level of granularity. The underlying assumption is that larger artifacts require more time and effort to maintain [ABJ10] and thus more emphasis shall be placed on such artifacts and their contribution to the occurrence of an event of interest, such as a fix. Hence, if there is weight to be removed in a fix, the chunk of that weight to be removed from a given artifact is assumed to be proportional to the size of the artifact. The size of an artifact is generally measured in terms of lines of code, however other measures may be used as well. The distribution factor for the *size* strategy is defined as:

$$df(a_t, \text{size}) = \frac{as(a_t)}{as(c_t)} \quad (12)$$

The application of the *size* strategy to the running example from Figures 2–5 and the resulting weight redistribution is shown in Figure 6. Given the artifact sizes $as(x_5) = 40$ and $as(z_5) = 60$, the corresponding distribution factors are $df(x_5, \text{size}) = 0.4$ and $df(z_5, \text{size}) = 0.6$, which are also identical to the respective distributed removed weights for x_5 and z_5. The total and average weights of the corresponding causing states at the file level of granularity are also

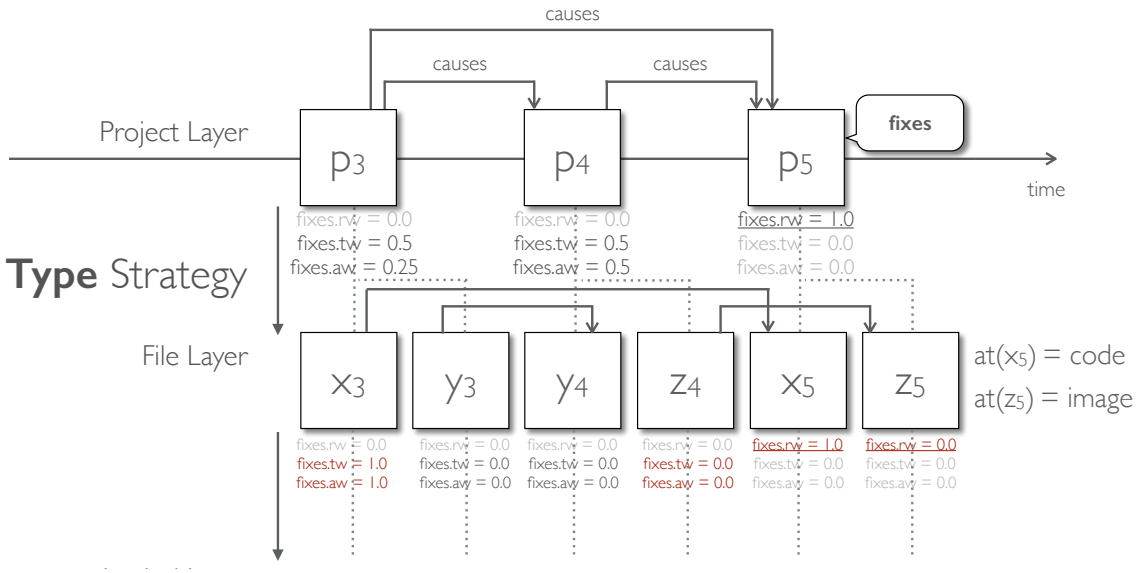

Figure 5: Type strategy for distributing removed weights across layers

adjusted respectively, emphasising the impact of the size of the corresponding artifacts in the fixing state on their contribution to the fix as indicated by the removed weight assigned to them, and also on the total and average weights of the corresponding causing states.

Similar to the *shared* strategy, the *size* strategy shall be applied at each logical levels of granularity (e.g. Class, Method, Function, etc.) separately, which effectively results in a refinement of the *size* strategy that also integrates the *type* strategy. In that case, the *size* strategy takes a parameter T denoting the type of artifacts it shall be applied to. Given the *typed artifact size* (*tas*) for a state of an artifact c_t and an artifact type T defined as the sum of the sizes of all artifacts of type T in the states contained in c_t:

$$tas(c_t, T) = \sum_{a_t \in c_t^{\text{CONTENTS}} : at(a_t) = T} as(a_t) \qquad (13)$$

the *size* strategy is refined by integrating the *tas* in the distribution factor resulting in:

$$df(a_t, \text{size:}T) = \begin{cases} \frac{as(a_t)}{tas(c_t, T)} & \text{if } at(a_t) = T \\ 0 & \text{otherwise} \end{cases} \qquad (14)$$

Apart from the application at the logical levels of granularity, this refinement also combines the emphasis on the type and the size of the artifact. When applied at the file level of granularity, only the size of artifacts of the given type is taken into consideration. If a fixing state at the project level includes states of artifacts of different types, e.g. *code* and *test*, and we are interested primarily in artifacts of type *code*, the typed *size* strategy distributes the removed weight according to the size of code artifacts only. Thus, even if the fixing state contains large test artifacts, they will have no impact on the weight distribution among the code artifacts. Similar to the *type* strategy, the typed *size* strategy can be applied multiple times for different types of artifacts, essentially resulting in a distribution of removed weights "within type".

5.4 Churn Strategy

The *churn* strategy emphasises the impact of the amount of change (churn) of an artifact (*ac*) in a given state that is considered as a part of a fixing state at a coarser level of granularity. The underlying assumption is that larger changes in artifacts require more time and effort to perform and potentially contribute more to the occurrence of an event of interest, such as a fix. Hence, if there is weight to be removed in a fix, the chunk of that weight to be removed from a given artifact is assumed to be proportional to the amount of change that needed to be performed in the artifact. The distribution factor for the *churn* strategy is defined as:

$$df(a_t, \text{churn}) = \frac{ac(a_t)}{ac(c_t)} \qquad (15)$$

The application of the *churn* strategy to the running example from Figures 2–6 and the resulting

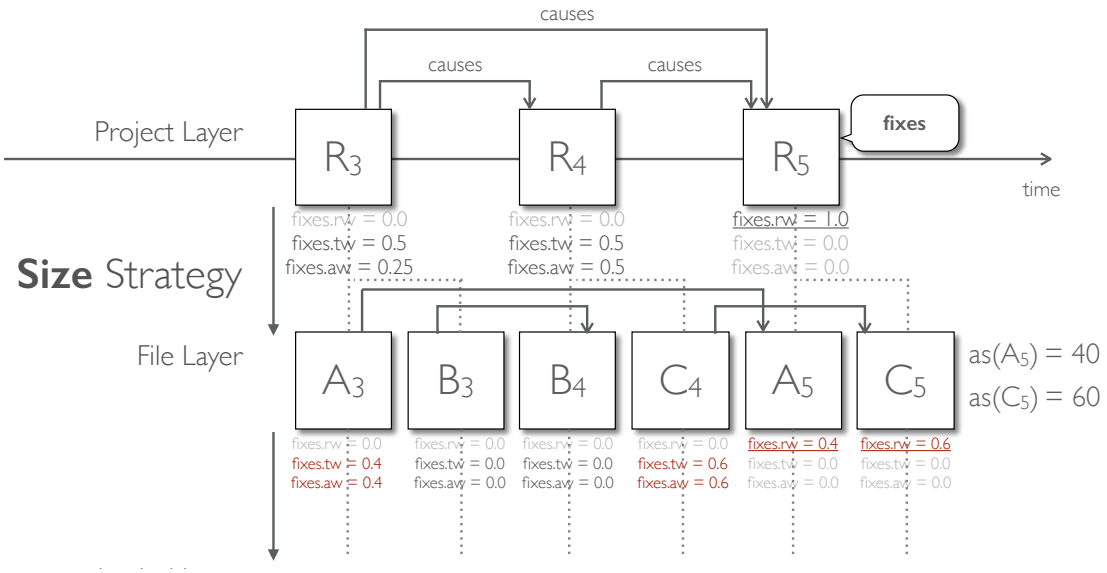

Figure 6: Size strategy for distributing removed weights across layers

weight redistribution is shown in Figure 7. Given that $ac(x_5) = 4$ and $ac(z_5) = 1$, the corresponding distribution factors are $df(x_5, \text{churn}) = 0.8$ and $df(z_5, \text{churn}) = 0.2$, which are also identical to the respective distributed removed weights for x_5 and z_5. The total and average weights of the corresponding causing states at the file level of granularity are also adjusted respectively. This emphasises the impact of the amount of change in the states of the corresponding artifacts in the fixing state on their contribution to the fix. Their contribution is indicated by the removed weight assigned to them. By extension, this also emphasises the impact of the amount of change on the total and average weights of the corresponding causing states.

Contemplating the application of both the *size* and the *churn* strategies, as illustrated in Figure 6 and Figure 7, respectively, we may observe a contradiction in the weight distributions. The *size* strategy indicates that z_5 is contributing more to the fix in p_5 due to its larger size and hence its causing state z_4 is the more likely cause for the fix in p_5. On the other hand, the *churn* strategy indicates that x_5 is contributing more to the fix in p_5 due to the larger amount of change in x_5 and hence its causing state x_3 is the more likely cause for the fix in p_5. The different strategies ultimately enable emphasising different characteristics of events of interest. Which one is to be used depends on the application context and the assessment task. If the size of artifacts is perceived as resulting in more effort involved in maintenance and development tasks, then the *size* strategy will be more adequate for identifying and emphasising the states of artifacts that contribute both to events of interest and to their likely causes based on the relative importance of these states with respect to the effort involved in understanding them. On the other hand, if the amount of change in states of artifacts is considered more critical with respect to the effort involved in maintenance and development tasks, then the *churn* strategy will be more adequate. The states of artifacts that contribute both to events of interest and to their likely causes can be identified and emphasised based on their relative importance with respect to the effort involved in modifying them.

There are different kinds of churn measures described in the literature [KAG+96, KS94, MGP13, NB05]. We consider a rather simple absolute measure of churn defined as the sum of additions and removals in terms of lines (*churned lines of code* in [NB05]), where a modification is considered both a removal and an addition of one or more lines that are part of the modification. Other notions of churn can also be used in the *churn* strategy, however if a relative churn measure is used, such as the ones described in [NB05], the distribution factor may need to be adjusted as well.

Similar to the *shared* and the *size* strategy, the *churn* shall be applied at each logical levels of granularity (e.g. Class, Method, Function, etc.) separately, which effectively results in a refinement of the *churn* strategy that also integrates the *type* strategy, analogous to the *size* strategy. In that case, the *churn* strategy takes a parameter T denoting the type of artifacts it shall be applied to. Given the *typed artifact churn* (*tac*) for a state of an artifact c_t and an artifact

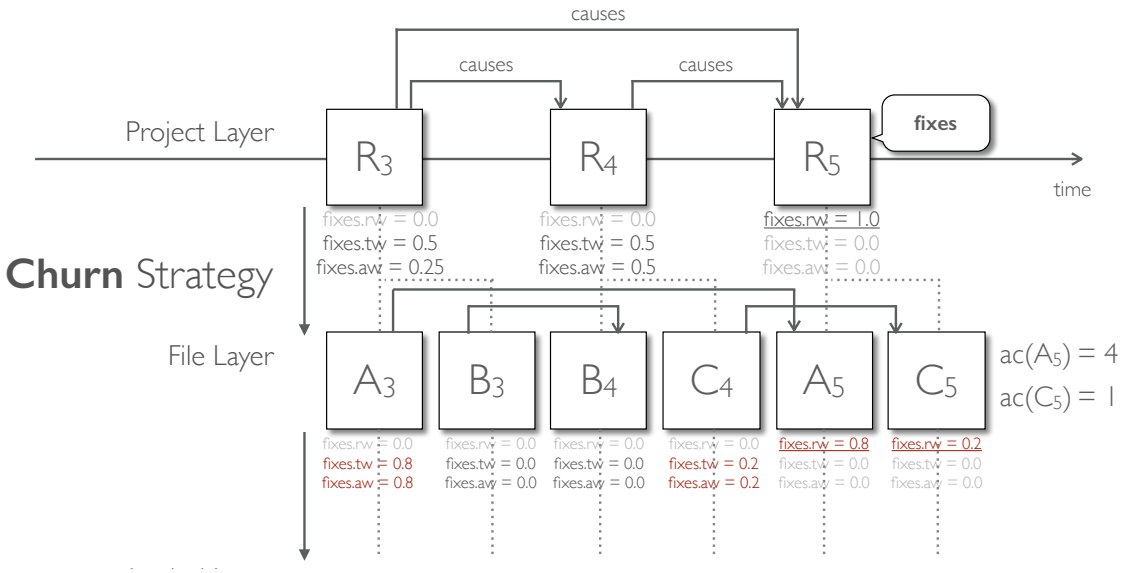

Figure 7: Churn strategy for distributing removed weights across layers

type T defined as the sum of the churn of all artifacts of type T in the states contained in c_t:

$$tac(c_t, T) = \sum_{a_t \in c_t^{\text{CONTENTS}} : at(a_t) = T} ac(a_t) \qquad (16)$$

the *churn* strategy is refined by integrating the *tac* in the distribution factor resulting in:

$$df(a_t, \text{churn:}T) = \begin{cases} \frac{as(a_t)}{tac(c_t, T)} & \text{if } ac(a_t) = T \\ 0 & \text{otherwise} \end{cases} \qquad (17)$$

Apart from the application at the logical levels of granularity, this refinement also combines the emphasis on the type of the artifact and the amount of change in the artifact. When applied at the file level of granularity, only the churn of artifacts of the given type is taken into consideration. If a fixing state at the project level includes states of artifacts of different types, e.g. *code* and *test*, and we are interested primarily in artifacts of type *code*, the typed *churn* strategy distributes the removed weight according to the churn of code artifacts only. Thus, even if the fixing state contains large changes to test artifacts, they will have no impact on the weight distribution among the code artifacts. Similar to the *type* and the typed *size* strategy, the typed *churn* strategy can be applied multiple times for different types of artifacts, essentially resulting in a distribution of removed weights "within type".

6 Related Work

Existing approaches are typically based on some form of origin analysis [GT02], involving line tracking and annotation graphs [KZPW06], line histories [CC06], line mapping [MHC14], as well as refinements to these [WS08, CCDP09] in order to map and track entities across revisions. Historage [HMK11] is an approach for tracing fine-grained artifact histories including renaming changes. The approach presented in this chapter builds on top of these approaches, applying origin analysis to events of interest in order to determine their potential causes and then quantifying the cause-fix relationships by means of weights. Our approach also considers different levels of granularity. Any of the existing approaches can be used as a foundation and generally the accuracy of the weighting depends in part on the quality of the results from the underlying origin analysis approach.

Different applications for the existing approaches have been discussed in the literature, ranging from finding fix-inducing changes [SZZ05] and understanding the role of authorship on implicated code [RD11] to defect-insertion circumstance analysis [PP14]. While in a sense such applications do serve a similar purpose — identifying potential causes for events of interest, they are more focused on identifying such causes before the event of interest has occurred. Such applications generally require sufficient information about known causes for events of interest, which serves as training data in order to build pattern recognition models that are then used to identify potential causes for events

of interest. Both, the training and the validation of such pattern recognition models requires data annotated with known causes for events of interest. The approach discussed in this chapter can be applied to produce such data emphasising different characteristics across multiple levels of granularity for different kinds of events of interest.

The challenge of "tangled changes" [HZ13] is somewhat related to topic of this chapter, where the authors study the prevalence of such changes that are unrelated or loosely related to events of interest and apply a multi-predictor approach to untangle them, based on different confidence voters. The approach discussed in this article relies on weighting and different weight distribution strategies to emphasise certain characteristics of changes related to events of interest, that are considered to be of importance in a given context. It can further benefit from a more sophisticated untangling approach, such as the one described in [HZ13], which can be incorporated as an additional weight distribution strategy to refine the distribution of weights among fixing and causing states of artifacts across the different levels of granularity.

To the best of our knowledge none of the existing approaches has incorporated quantification of the extent to which a change in one state contributes to a subsequent fix in a later state of an artifact. Also, none of the approaches has explored how to apply cause-fix analysis across multiple levels of granularity.

7 Conclusion

In this article, we explored a weight-based approach for finding potential causes for events of interest in software repositories. An event of interest can be any occurrence that may be of relevance for an assessment task, such as fixing issues and problems, improving properties, adding features and functionality, and refactoring code. The approach adds quantitative information on top of existing approaches for origin analysis, such as ones based on line tracking. The quantitative information is in the form of weights, where an event of interest regarded as a fix is considered to be removing a weight, and the potential causes for the event of interest are considered to be contributing to the presence of that weight. Distinct weights can be calculated across different dimensions, based on the kind of event of interest, such as a bug fix, refactoring, etc., designated by a distinct factor for each kind of interest. The approach accommodates weight redistribution across multiple layers corresponding to different levels of granularity in order to provide more accurate information at these levels of granularity. We outlined different strategies for weight redistribution across the different levels of granularity, which enable emphasising different characteristics of the states of artifacts involved in an event of interest, such as their type, size, or the amount of change they have undergone. The emphasis on different characteristics allows us to account for the importance of these characteristics in the effort involved in performing an activity that leads to an event of interest or its causes. Further weight distribution strategies may be defined in order to emphasise other characteristics or combinations of characteristics of events of interest.

There are different related approaches described in the literature, which seek to establish relationships between fixes and their likely causes. However, none of them have incorporated quantification of the extent to which a likely cause contributes to a subsequent fix, especially across multiple levels of granularity. The presented approach builds on top of these approaches and generally any of them can serve as a foundation, providing the relationships between fixes and their likely causes. Based on these relationships, the proposed approach can be used to calculate the corresponding weights and quantify the cause-fix relationships. There are also different related applications discussed in the literature which can be used for similar purposes. However, their scope and focus is mostly on identifying potential causes for events of interest, where the event of interest has not yet occurred. The approach discussed in this article can be applied to provide necessary information for the configuration, validation, and refinement of such applications.

While the set of revisions identified as causes for a given revision is definitive, meaning that no additional causes may be added for that revision, the set of revisions identified as fixes for a given revision reflects the state of knowledge at a given point in time, meaning that future revisions may also fix issues introduced in that revision. This affects the reliability of the calculated weights. In future work, a suitable cut-off point in time needs to be defined, after which the calculated weights for causing states can be considered unreliable. Such a cut-off point may be based on release tags, or on the distance between causing and fixing states with respect to a particular factor, or on the distance between causing and fixing states in general.

Establishing the real causes for events of interest is a hard task. The presented approach provides a foundation for the quantification of potential causes. The next step is to investigate the extent to which the presented approach can be used to determine the real causes for events of interest, and in particular the role of different weight distribution strategies and combinations of strategies towards that goal.

References

[ABJ10] Erik Arisholm, Lionel C. Briand, and Eivind B. Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *J. Syst. Softw.*, 83(1):2–17, 2010.

[CC06] G. Canfora and L. Cerulo. Fine grained indexing of software repositories to support impact analysis. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 105–111, Shanghai, China, 2006. ACM.

[CCDP09] G. Canfora, L. Cerulo, and M. Di Penta. Tracking Your Changes: A Language-Independent Approach. *Software, IEEE*, 26(1):50 –57, February 2009.

[GT02] M. Godfrey and Q. Tu. Tracking structural evolution using origin analysis. In *Proceedings of the international workshop on Principles of software evolution - IWPSE '02*, page 117, Orlando, Florida, 2002.

[HMK11] Hideaki Hata, Osamu Mizuno, and Tohru Kikuno. Historage: Fine-grained Version Control System for Java. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11, pages 96–100, New York, NY, USA, 2011. ACM.

[HZ13] Kim Herzig and Andreas Zeller. The impact of tangled code changes. MSR '13, page 121130, Piscataway, NJ, USA, 2013. IEEE Press.

[KAG+96] T.M. Khoshgoftaar, E.B. Allen, N. Goel, A. Nandi, and J. McMullan. Detection of software modules with high debug code churn in a very large legacy system. In *Seventh International Symposium on Software Reliability Engineering, 1996. ISSRE 1996*, pages 364–371, October 1996.

[KS94] T.M. Khoshgoftaar and R.M. Szabo. Improving code churn predictions during the system test and maintenance phases. In *International Conference on Software Maintenance, 1994. ICSM 1994*, pages 58–67, September 1994.

[KZPW06] S. Kim, T. Zimmermann, K. Pan, and E.J. Whitehead. Automatic Identification of Bug-Introducing Changes. In *Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on*, pages 81–90, 2006.

[MGP13] Philip Makedonski, Jens Grabowski, and Florian Philipp. Quantifying the evolution of TTCN-3 as a language. *International Journal on Software Tools for Technology Transfer*, 16(3):227–246, July 2013.

[MHC14] P. Marinescu, P. Hosek, and C. Cadar. Covrig: A Framework for the Analysis of Code, Test, and Coverage Evolution in Real Software. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 93–104, New York, NY, USA, 2014. ACM.

[NB05] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *27th International Conference on Software Engineering, 2005. ICSE 2005*, pages 284–292. IEEE, May 2005.

[PP14] L. Prechelt and A. Pepper. Why Software Repositories Are Not Used for Defect-insertion Circumstance Analysis More Often: A Case Study. *Inf. Softw. Technol.*, 56(10):1377–1389, October 2014.

[RD11] F. Rahman and P. Devanbu. Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 491–500, New York, NY, USA, 2011. ACM.

[SZZ05] J. Sliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, St. Louis, Missouri, 2005. ACM.

[WS08] C. Williams and J. Spacco. SZZ revisited: verifying when changes induce fixes. In *Proceedings of the 2008 workshop on Defects in large software systems*, DEFECTS '08, pages 32–36, New York, NY, USA, 2008. ACM. ACM ID: 1390826.