Merge and Code Review

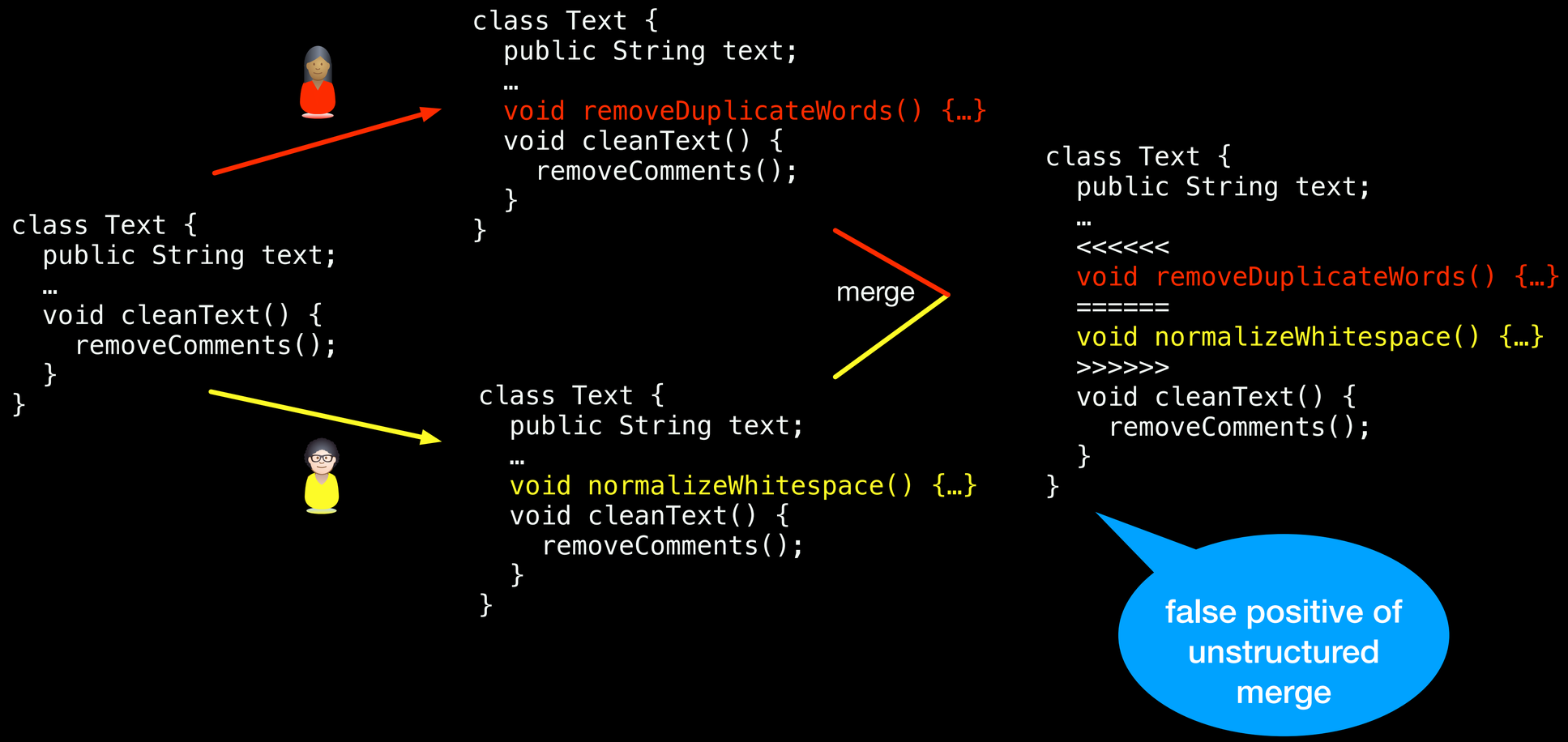
Paulo Borba
Informatics Center
Federal University of Pernambuco

pauloborba.cin.ufpe.br

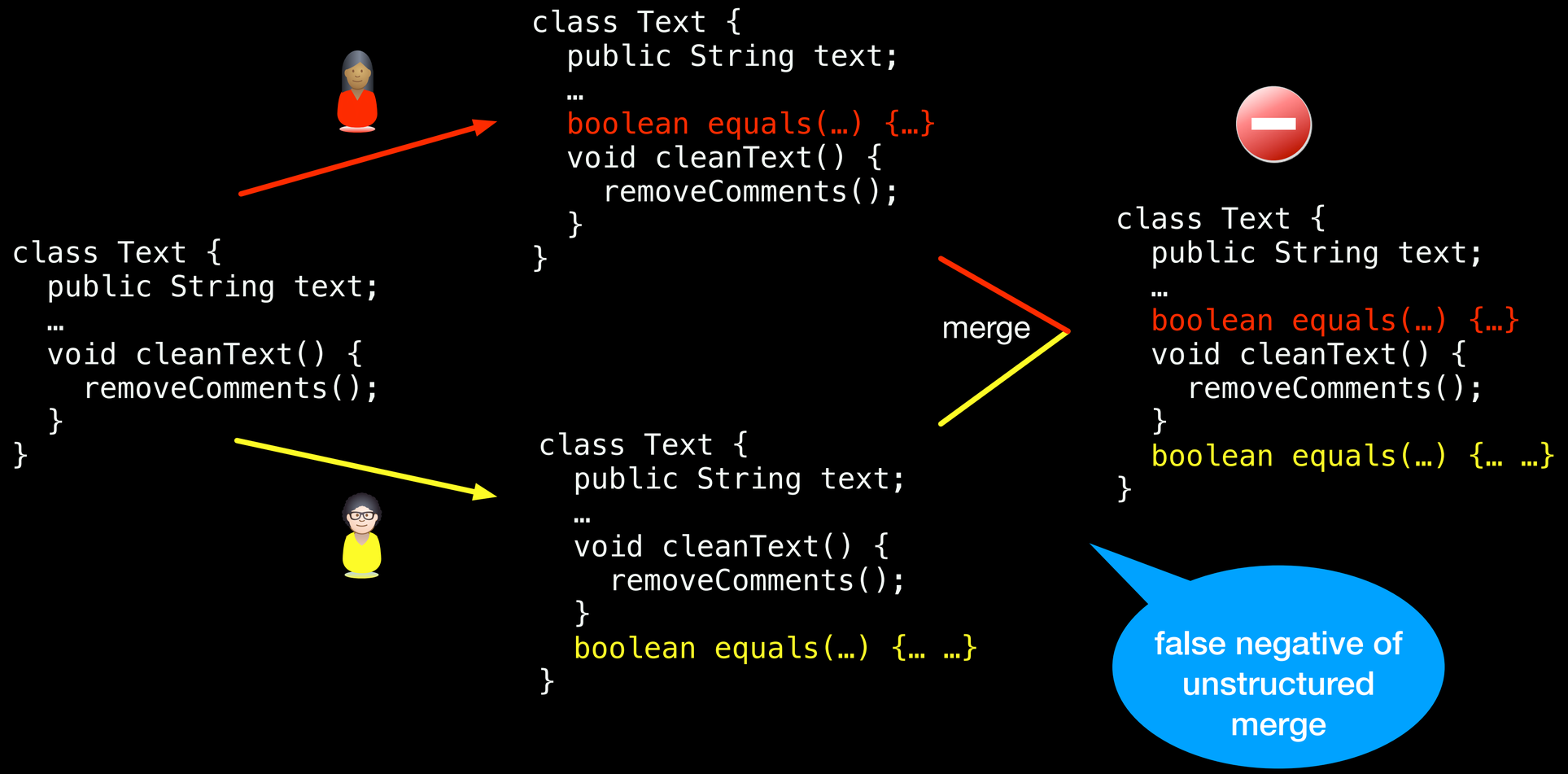
Semistructured merge

Trying to avoid unstructured merge false positives and negatives

Developers waste time by manually resolving conflicts that could be automatically solved



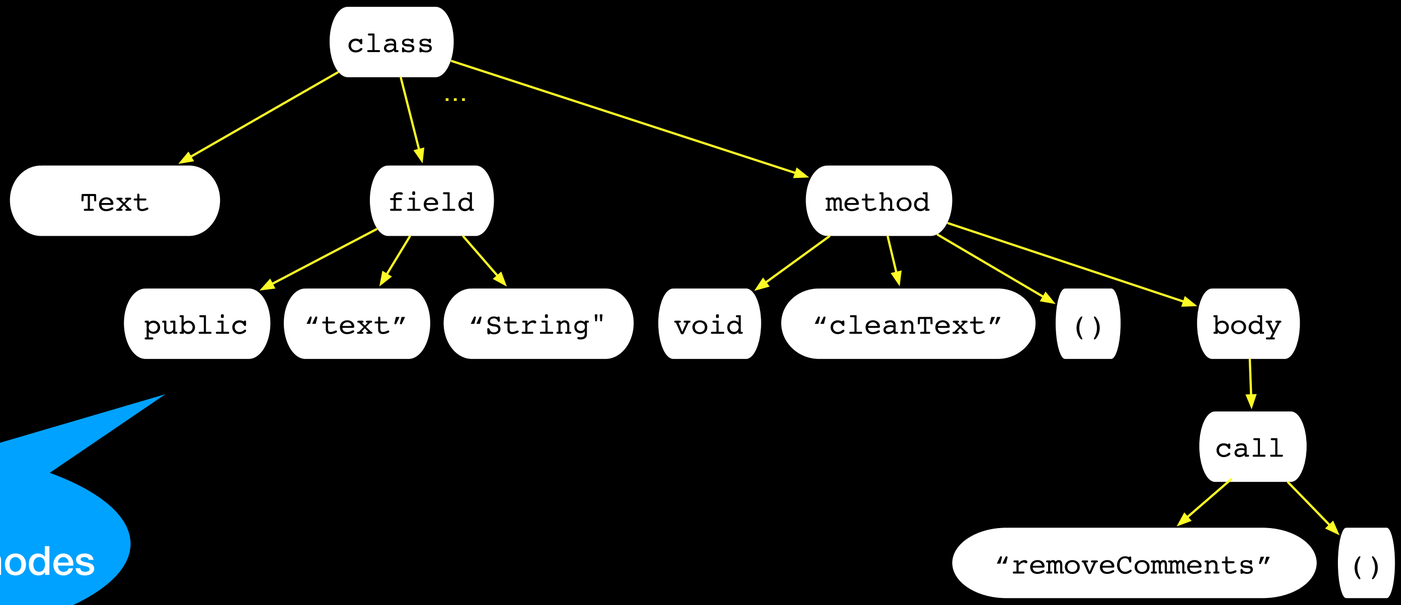
Current merge tools might integrate conflicting changes without warning developers



Structured
and
semistructured
merge tools to the rescue

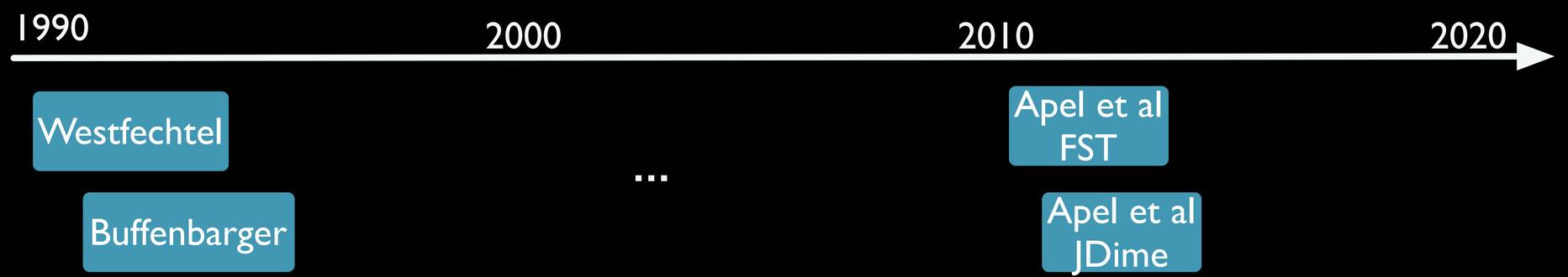
Structured merge works with ASTs, not a sequence of lines

```
class Text {              \n
  public String text;     \n
  …                       \n
  void cleanText() {      \n
    removeComments();     \n
  }                       \n
}                         \n
```



no conflicts for changes in different nodes

Structured merge tools timeline



Computationally expensive

Language dependent

No evaluation

Much less expensive

Much less dependent

Evaluation

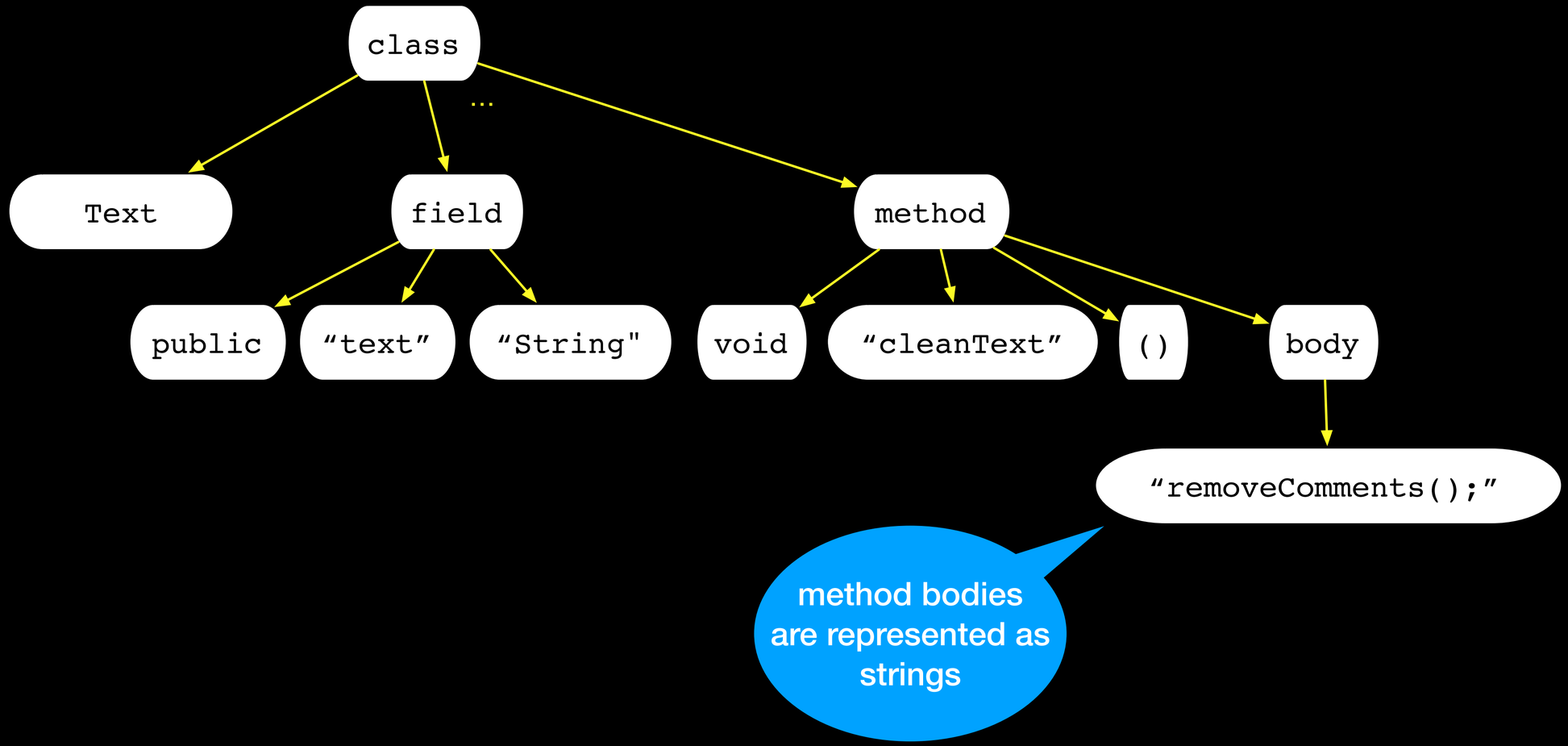
B. Westfechtel. Structure-Oriented Merging of Revisions of Software Documents. SCM 1991.

J. Buffenbarger. Syntactic Software Merging. SCM 1995.

S. Apel, J. Liebig, B. Brandl, C. Lengauer, and C. Kästner. Semistructured Merge: Rethinking Merge in Revision Control Systems. ESEC/FSE 2011.

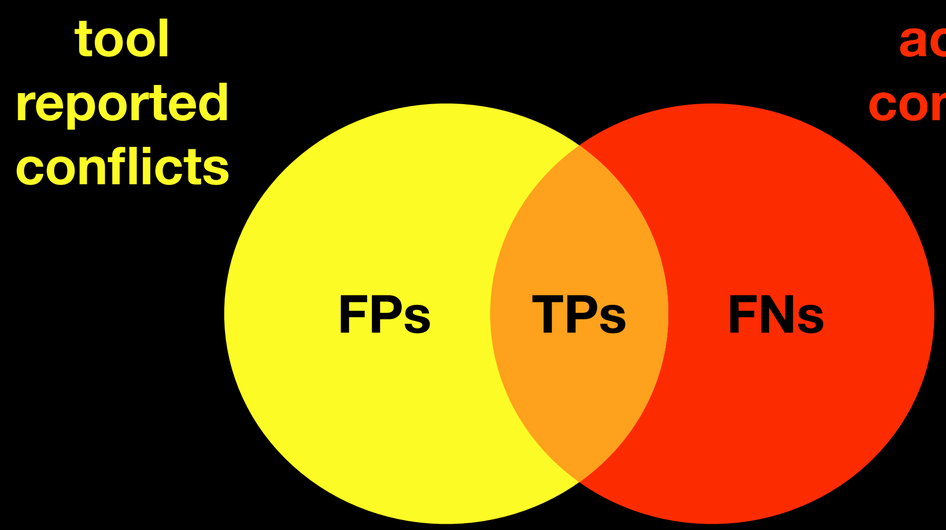
S. Apel, O. Leßenich, and C. Lengauer. Structured Merge with Auto-Tuning: Balancing Precision and Performance. ASE 2012.

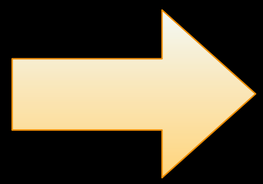
Semistructured merge works with partial ASTs



Evidence only about the reduction of reported conflicts

(others: 34% ± 21%, 39%) (here: 62% ± 24%, 71% ± 30%)





not for all projects and merge situations

Assessing Semistructured Merge in Version Control Systems: A Replicated Experiment

Guilherme Cavalcanti
Federal University of Pernambuco
Recife, Brazil
gjcc@cin.ufpe.br

Paola Accioly
Federal University of Pernambuco
Recife, Brazil
prga@cin.ufpe.br

Paulo Borba
Federal University of Pernambuco
Recife, Brazil
phmb@cin.ufpe.br

***Abstract*— *Context:* To reduce the integration effort arising from conflicting changes resulting from collaborative software development, unstructured merge tools rely on the textual similarity of the conflicts via textual similarity, whereas structured and semistructured merge tools try to go further by exploiting the syntactic structure of the involved artifacts. *Objective:* In this paper, aiming at increasing the existing body of evidence and assessing results for systems developed under an alternative version control paradigm, we replicate an experiment comparing the use of the unstructured, the semistructured and semistructured approach with respect to the occurrence of conflicts reported by both approaches. *Method:* We used both semistructured and unstructured merge in a sample 2.5 times bigger than the original study regarding the number of projects and we compared the occurrence of conflicts. *Results:* Similar to the original study, we observed that semistructured merge reduces the number of conflicts in 85% of the scenarios of the new sample. However, the observed average conflict reduction of 62% is 10 percentage points lower than the one observed in the study before. We also bring new evidence that the use of semistructured merge can reduce the occurrence of conflicting merge scenarios by half. *Conclusions:* Our findings reinforce the benefits of exploiting the syntactic structure of the artifacts involved in code integration. Besides, the reductions observed in the number and size of conflicts suggest that the use of semistructured merge, as implemented by FSTMerge, can significantly decrease integration effort without compromising correctness.

Keywords*— *replication study, collaborative development, software merging, semistructured merge, version control systems.

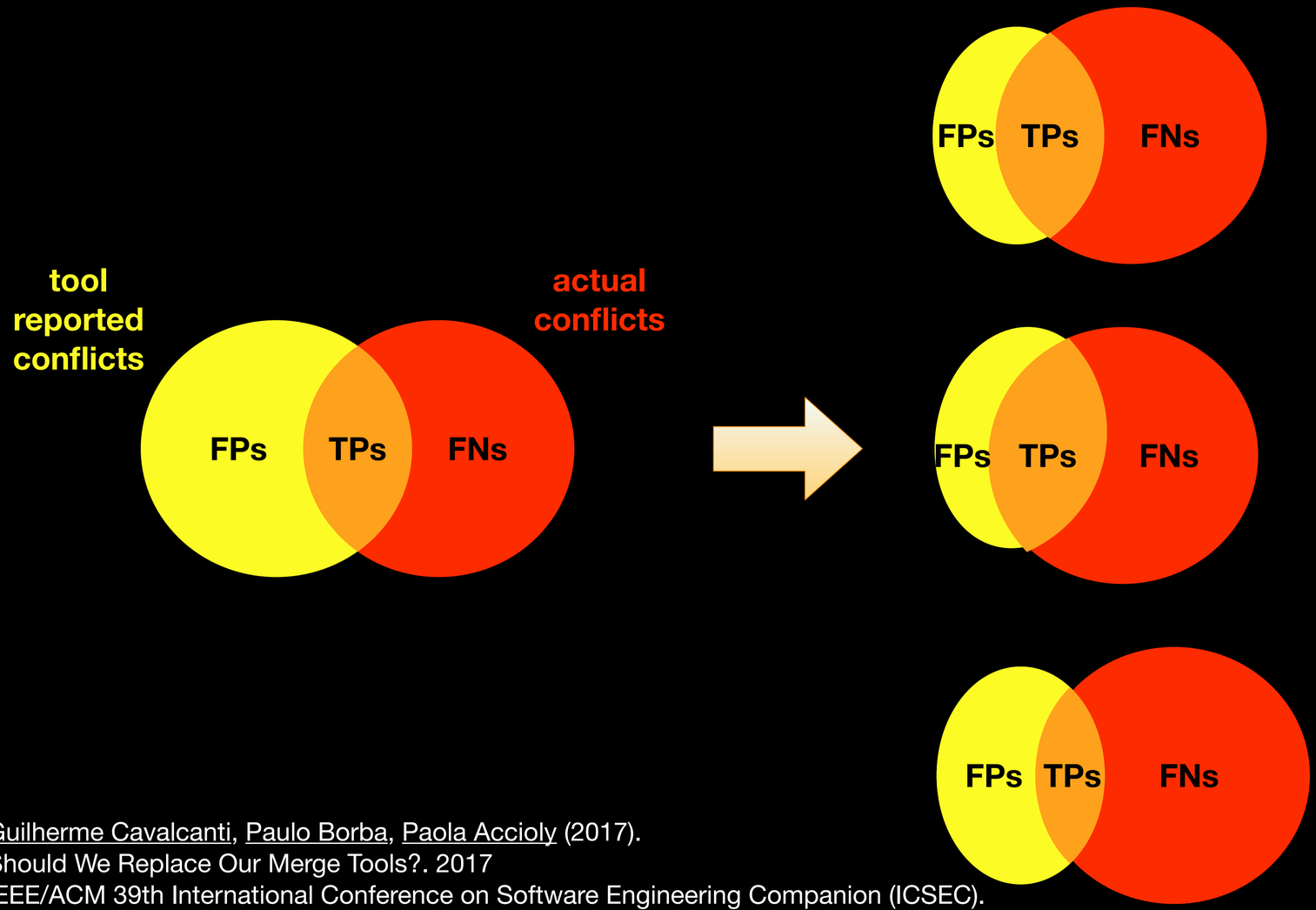
ing when developers detect conflicts, and how often conflicts occur. Zimmermann [4], for instance, describes that textual conflicts occurred in a range from 23% to 47% of all files' integration. Brun et al. [2] and Kasi and Sarma [5] found that textual conflicts occurred in an average of 15% of all merge scenarios – a set consisting of a common base revision and two of its subsequent revisions. In 33% of the occurrences of textual conflicts resulted in build or behavioral errors.

Such evidence motivates and guides the design of tools that use different strategies to both decrease integration effort and improve correctness during code integration. For example, to reduce the integration effort, unstructured merge tools are purely text-based and resolve conflicts via textual similarity. On the other hand, a structured merge tool is tailored to a specific programming language and uses knowledge of the syntactic structure of artifacts to merge [3, 6, 7, 8, 9, 10]. Finally, semistructured merge [1] attempts to combine the previous ones, so that it provides structural information about software artifacts to resolve conflicts automatically, and when this information is not sufficient, it applies the usual textual resolution to the conflict.

Apel et al. [1] in a previous empirical study found that the semistructured approach was promising if compared to the unstructured one. By studying 24 projects using Subversion, a Centralized Version Control System (CVCS), and analyzing a total of 180 merge scenarios, they found that, in 64% of their sample merge scenarios, the semistructured approach was able to reduce the number of textual conflicts by, on average

G. Cavalcanti, P. Borba, P. Accioly. Assessing Semistructured Merge in Version Control Systems: A Replicated Experiment. ESEM 2015.

Integration effort reduction?
Integration correctness impact?



2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion

Should we replace our merge tools?

Guilherme Cavalcanti, Paulo Borba and Paola Accioly
Informatics Center
Federal University of Pernambuco
Recife, Brazil
{gjcc,phmb,prga}@cin.ufpe.br

***Abstract*—While unstructured merge tools try to automatically resolve merge conflicts via textual similarity, semistructured merge tools try to go further by partially exploiting the syntactic structure and semantics of the involved artefacts. Previous studies compare these merge approaches with respect to the number of reported conflicts, showing, for most projects and merge situations, a reduction in favor of semistructured merge. However, these studies do not investigate whether this reduction actually leads to integration effort reduction (Productivity) without negative impact on the correctness of the merging process (Quality). To analyze this, and to better understand how these tools could be improved, we propose empirical studies to identify spurious conflicts reported by one approach but not by the other, and interference reported as conflict by one approach but missed by the other.**

***Keywords*-software merging; collaborative development; version control systems; empirical studies**

since the observed reduction might be obtained at the expense of missing actual interference between developers changes (false negatives). Moreover, given that the set of conflicts reported by semistructured merge in previous studies is often smaller but not a subset of the set reported by unstructured merge, semistructured merge could even be introducing other kinds of false positives that might be harder to resolve than the ones it eliminates. As even a minor disadvantage of a new tool can become a huge barrier for its adoption in practice, if we want to move forward on the state of the practice on merge tools, it is important to have solid evidence and further knowledge about false positives and false negatives resulting from those two merge approaches. This way developers can choose the merge approach to be used in terms of their impact on integration effort (Productivity) and correctness (Quality)—factors that are

Guilherme Cavalcanti, Paulo Borba, Paola Accioly (2017).
Should We Replace Our Merge Tools?. 2017
IEEE/ACM 39th International Conference on Software Engineering Companion (ICSEC).

A more comprehensive evaluation is not fully favourable for semistructured merge

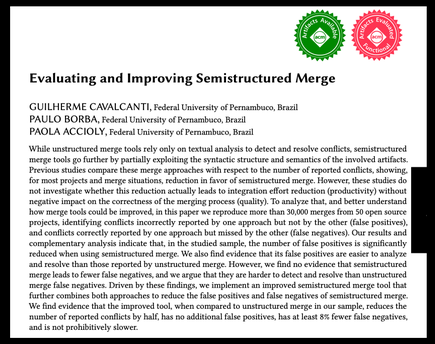
(34,030 merges from 50 open source projects)

FSTMerge reports less false positives (34%) than unstructured merge tools.

Easier to analyze and resolve.

No evidence that FSTMerge leads to fewer false negatives.

Harder to detect and resolve.



Evaluating and Improving Semistructured Merge

GUILHERME CAVALCANTI, Federal University of Pernambuco, Brazil
PAULO BORBA, Federal University of Pernambuco, Brazil
PAOLA ACCIOLY, Federal University of Pernambuco, Brazil

While unstructured merge tools rely only on textual analysis to detect and resolve conflicts, semistructured merge tools go further by partially exploiting the syntactic structure and semantics of the involved artifacts. Previous studies compare these merge approaches with respect to the number of reported conflicts, showing, for most projects and merge situations, reduction in favor of semistructured merge. However, these studies do not investigate whether this reduction actually leads to integration effort reduction (productivity) without negative impact on the correctness of the merging process (quality). To analyze that, and better understand how merge tools could be improved, in this paper we reproduce more than 30,000 merges from 50 open source projects, identifying conflicts incorrectly reported by one approach but not by the other (false positives), and conflicts correctly reported by one approach but missed by the other (false negatives). Our results and complementary analysis indicate that, in the studied sample, the number of false positives is significantly reduced when using semistructured merge. We also find evidence that its false positives are easier to analyze and resolve than those reported by unstructured merge. However, we find no evidence that semistructured merge leads to fewer false negatives, and we argue that they are harder to detect and resolve than unstructured merge false negatives. Driven by these findings, we implement an improved semistructured merge tool that further combines both approaches to reduce the false positives and false negatives of semistructured merge. We find evidence that the improved tool, when compared to unstructured merge in our sample, reduces the number of reported conflicts by half, has no additional false positives, has at least 8% fewer false negatives, and is not prohibitively slower.

G. Cavalcanti, P. Borba, P. Accioly. Evaluating and Improving Semistructured Merge. OOPSLA 2017. https://pauloborba.cin.ufpe.br/publication/2017evaluating_and_improving_semistructured_merge/

not uniformly across projects

s3m: an improved semistructured merge tool for Java

s3m reports less conflicts (51%)

no additional false positives

at least 8% fewer false negatives

not prohibitively slower
(32x, < 1s in 80% of the scenarios, > 5s in only 2%)

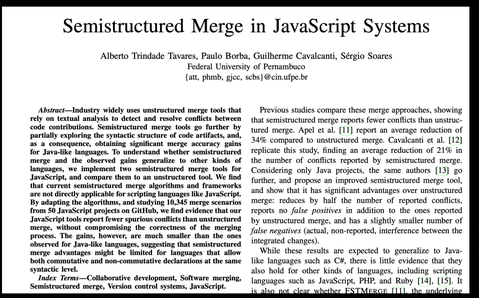
But the benefits do not generalize, as strongly, to Javascript

(10,345 merges from 50 projects)

s3m reports less conflicts (6%)

fewer false positives (87%) without compromising correctness (1 FN)

commutative and non-commutative elements at the same syntactic level



Semistructured Merge in JavaScript Systems

Alberto Trindade Tavares, Paulo Borba, Guilherme Cavalcanti, Sérgio Soares
Federal University of Pernambuco
{att, phmb, gjcc, scbs}@cin.ufpe.br

***Abstract*—Industry widely uses unstructured merge tools that rely on textual analysis to detect and resolve conflicts between code contributions. Semistructured merge tools go further by partially exploring the syntactic structure of code artifacts, and, as a consequence, obtaining significant merge accuracy gains for Java-like languages. To understand whether semistructured merge and the observed gains generalize to other kinds of languages, we implement two semistructured merge tools for JavaScript, and compare them to an unstructured tool. We find that current semistructured merge algorithms and frameworks are not directly applicable for scripting languages like JavaScript. By adapting the algorithms, and studying 10,345 merge scenarios from 50 JavaScript projects on GitHub, we find evidence that our JavaScript tools report fewer spurious conflicts than unstructured merge, without compromising the correctness of the merging process. The gains, however, are much smaller than the ones observed for Java-like languages, suggesting that semistructured merge advantages might be limited for languages that allow both commutative and non-commutative declarations at the same syntactic level.**

***Index Terms*—Collaborative development, Software merging, Semistructured merge, Version control systems, JavaScript.**

Previous studies compare these merge approaches, showing that semistructured merge reports fewer conflicts than unstructured merge. Apel et al. [11] report an average reduction of 34% compared to unstructured merge. Cavalcanti et al. [12] replicate this study, finding an average reduction of 21% in the number of conflicts reported by semistructured merge. Considering only Java projects, the same authors [13] go further, and propose an improved semistructured merge tool, and show that it has significant advantages over unstructured merge: reduces by half the number of reported conflicts, reports no *false positives* in addition to the ones reported by unstructured merge, and has a slightly smaller number of *false negatives* (actual, non-reported, interference between the integrated changes).

While these results are expected to generalize to Java-like languages such as C#, there is little evidence that they also hold for other kinds of languages, including scripting languages such as JavaScript, PHP, and Ruby [14], [15]. It is also not clear whether FSTMERGE [11], the underlying

A. Tavares, P. Borba, G. Cavalcanti, S. Soares. Semistructured Merge in JavaScript Systems. ASE 2019 https://pauloborba.cin.ufpe.br/publication/2019semistructured_merge_in_javascript_systems/

```
if ((window.onanimationend === undefined) && (window.onwebkitanimationend
  CSS_PREFIX = '-webkit-';
  ANIMATION_PROP = 'WebkitAnimation';
  ANIMATIONEND_EVENT = 'webkitAnimationEnd animationend';
} else {
  ANIMATION_PROP = 'animation';
  ANIMATIONEND_EVENT = 'animationend';
}

var DURATION_KEY = 'Duration';
var PROPERTY_KEY = 'Property';
var DELAY_KEY = 'Delay';
var TIMING_KEY = 'TimingFunction';
var ANIMATION_ITERATION_COUNT_KEY = 'IterationCount';
var ANIMATION_PLAYSTATE_KEY = 'PlayState';
var SAFE_FAST_FORWARD_DURATION_VALUE = 9999;

var ANIMATION_DELAY_PROP = ANIMATION_PROP + DELAY_KEY;
var ANIMATION_DURATION_PROP = ANIMATION_PROP + DURATION_KEY;
var TRANSITION_DELAY_PROP = TRANSITION_PROP + DELAY_KEY;
var TRANSITION_DURATION_PROP = TRANSITION_PROP + DURATION_KEY;

var ngMinErr = angular.$$minErr('ng');
function assertArg(arg, name, reason) {
  if (!arg) {
    throw ngMinErr('areq', 'Argument \'{0}\' is {1}', (name || '?'), (rea
  }
  return arg;
}
function concatWithSpace(a,b) {
  if (!a) return b;
  if (!b) return a;
  return a + ' ' + b;
}

var helpers = {
  blockTransitions: function(node, duration) {
    // we use a negative delay value since it performs blocking
    // yet it doesn't kill any existing transitions running on the
    // same element which makes this safe for class-based animations
    var value = duration ? '-' + duration + 's' : '';
    applyInlineStyle(node, [TRANSITION_DELAY_PROP, value]);
    return [TRANSITION_DELAY_PROP, value];
  }
};
```

statement

statement

statement

declaration

declaration

statement

Most semistructured merge conflicts happen because developers independently edit the same or consecutive lines of the same method

the probability of creating a merge conflict is approximately the same when editing methods, class fields, and modifier lists

most conflicting merge scenarios, and merge conflicts, involve more than two developers

Understanding Semistructured Merge Conflict Characteristics in Open-Source Java Projects

Paola Accioly · Paulo Borba · Guilherme Cavalcanti

Received: date / Accepted: date

Abstract Empirical studies show that merge conflicts frequently occur, impairing developers' productivity, since merging conflicting contributions might be a demanding and tedious task. However, the structure of changes that lead to conflicts has not been studied yet. Understanding the underlying structure of conflicts, and the involved syntactic language elements might shed light on how to better avoid merge conflicts. To this end, in this paper we derive a catalog of conflict patterns expressed in terms of the structure of code changes that lead to merge conflicts. We focus on conflicts reported by a semistructured merge tool that exploits knowledge about the underlying syntax of the artifacts. This way, we avoid analyzing a large number of spurious conflicts often reported by typical line based merge tools. To assess the occurrence of such patterns in different systems, we conduct an empirical study reproducing 70,047 merges from 123 GitHub Java projects. Our results show that most semistructured merge conflicts in our sample happen because developers independently edit the same or consecutive lines of the same method. However, the probability of creating a merge conflict is approximately the same when editing methods, class fields, and modifier lists. Furthermore, we noticed that most part of conflicting merge scenarios, and merge conflicts, involve more than two developers. Also, that copying and pasting pieces of code, or even entire files, across different repositories is a common practice and cause of conflicts. Finally, we discuss how our results reveal the need for new research studies and suggest potential improvements to tools supporting collaborative software development.

Paola Accioly, Paulo Borba, Guilherme Cavalcanti (2018).
Understanding semi-structured merge conflict characteristics in open-source Java projects.
Empirical Software Engineering.

More structure does not always improve merge accuracy

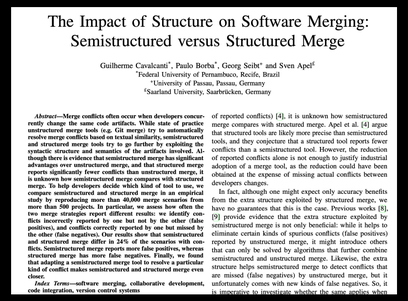
(43,509 merges from more than 500 projects)

tools do not often differ

(24% of the ~1K scenarios with conflicts)

semistructured merge reports more false positives (9x, 36)

structured merge has more false negatives (8x, 39)



The Impact of Structure on Software Merging: Semistructured versus Structured Merge

Guilherme Cavalcanti*, Paulo Borba*, Georg Seibt† and Sven Apel‡
*Federal University of Pernambuco, Recife, Brazil
†University of Passau, Passau, Germany
‡Saarland University, Saarbrücken, Germany

***Abstract*—Merge conflicts often occur when developers concurrently change the same code artifacts. While state of practice unstructured merge tools (e.g. Git merge) try to automatically resolve merge conflicts based on textual similarity, semistructured and structured merge tools try to go further by exploiting the syntactic structure and semantics of the artifacts involved. Although there is evidence that semistructured merge has significant advantages over unstructured merge, and that structured merge reports significantly fewer conflicts than unstructured merge, it is unknown how semistructured merge compares with structured merge. To help developers decide which kind of tool to use, we compare semistructured and structured merge in an empirical study by replaying 43,509 merge scenarios extracted from more than 500 projects. In particular, we assess the extent to which these strategies report different results, we identify conflicts incorrectly reported by one but not by the other (false positives), and conflicts correctly reported by one but missed by the other (false negatives). Our results show that semistructured and structured merge differ in 24% of the scenarios with conflicts. Semistructured merge reports more false positives, whereas structured merge has more false negatives. Finally, we found that adopting a semistructured merge tool to resolve a particular kind of conflict makes semistructured and structured merge even closer.**

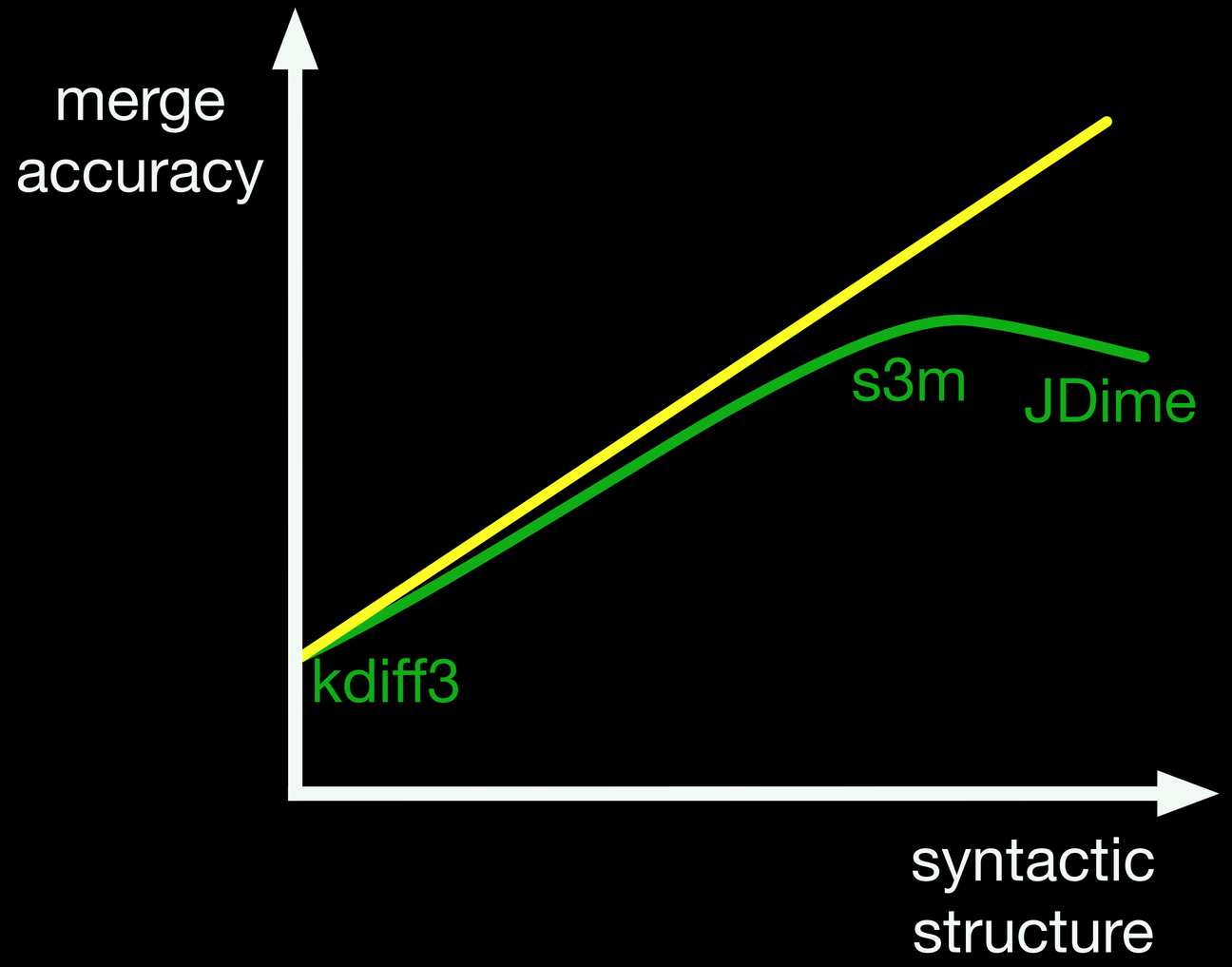
***Index Terms*—software merging, collaborative development, code integration, version control systems**

of reported conflicts) [4], it is unknown how semistructured merge compares with structured merge. Apel et al. [4] argue that structured tools are likely more precise than semistructured tools, and they conjecture that a structured tool reports fewer conflicts than a semistructured tool. However, the reduction of reported conflicts alone is not enough to justify industrial adoption of a merge tool, as the reduction could have been obtained at the expense of missing actual conflicts between developers changes.

In fact, although one might expect only accuracy benefits from the extra structure exploited by structured merge, we have no guarantees that this is the case. Previous works [8], [9] provide evidence that the extra structure exploited by semistructured merge is not only beneficial: while it helps to eliminate certain kinds of spurious conflicts (false positives) reported by unstructured merge, it might introduce others that can only be solved by algorithms that further combine semistructured and unstructured merge. Likewise, the extra structure helps semistructured merge to detect conflicts that are missed (false negatives) by unstructured merge, but it unfortunately comes with new kinds of false negatives. So, it is imperative to investigate whether the same applies when

G. Cavalcanti, P. Borba, G. Seibt, S. Apel. The Impact of Structure on Software Merging: Semistructured versus Structured Merge. ASE 2019 https://pauloborba.cin.ufpe.br/publication/2019the_impact_of_structure_on_software_merging__semistructured_versus_structured_merge/

Literature versus New results suggestions



Combining merge strategies and avoiding consecutive line conflicts show promising results

G. Seibt, F. Heck, G. Cavalcanti, P. Borba, S. Apel. Leveraging Structure in Software Merge: An Empirical Study. Submitted for publication, 2020.

Sesame = s3m + csdiff

	diff3 vs. SESAME	diff3 vs. s3m	s3m vs. SESAME
Total	1145	1019	675
(%)*	12.04%	10.72%	7.10%

* In relation to the total number of merged files.

Table 3: How often the merge tools differ.

	diff3	SESAME	s3m
Merge conflicts	2413	1413	1632
Conflicting files	1090	657	832

	diff3	SESAME
aFP	291	26
aFN	0	168

	s3m	SESAME
aFP	146	18
aFN	1	48

	diff3	s3m
aFP	169	31
aFN	4	124

Semistructured Merge with Language-Specific Syntactic Separators

Guilherme Cavalcanti
Federal Institute of Pernambuco
Belo Jardim, Brazil

Paulo Borba
Leonardo dos Anjos
Jonatas Clementino
Federal University of Pernambuco
Recife, Brazil

ABSTRACT

Structured merge tools exploit programming language syntactic structure to enhance merge accuracy by reducing spurious conflicts reported by unstructured tools. By creating and handling full ASTs, structured tools are language-specific and harder to implement. They can also be computationally expensive when merging large files. To reduce these drawbacks, semistructured merge tools work with partial ASTs that use strings to represent lower level syntactic structures such as method bodies, and rely on unstructured tools to merge them. This, however, results in merge accuracy loss. To improve accuracy without compromising semistructured merge benefits, we propose a tool that leverages language-specific syntactic separators to infer structure without parsing. We still resort to an unstructured tool to merge lower level structures, but only after preprocessing the code so that text in between separators such as curly braces appear in separate lines. This way we emulate the capabilities of structured merge tools while avoiding their drawbacks. By comparing our tool with a robust implementation of semistructured merge, we find that our tool substantially reduces the number of spurious conflicts. We also observe significant but less substantial reductions on the overall number of reported conflicts, and of files with conflicts. However, similar to structured tools, our tool lets more merge conflicts go undetected. Our tool shows significant improvements over unstructured tools widely used in practice. Finally we observe that exploiting language-specific syntactic separators introduces unique textual alignment challenges.

ACM Reference Format:
Guilherme Cavalcanti, Paulo Borba, Leonardo dos Anjos, and Jonatas Clementino. 2024. Semistructured Merge with Language-Specific Syntactic Separators. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3691620.3695483

1 INTRODUCTION

Unstructured merge tools have been in use for over 40 years [1], and are essential for team software development. These tools rely solely on textual analysis to detect and resolve merge conflicts [2]. They are fast and can handle any type of textual content, including source code in any language. However, they often report spurious merge conflicts, and fail to report conflicts that could lead to compilation and execution issues. For example, unstructured merge reports a conflict when asked to integrate rearrangeable declarations (such as method and field declarations) separately added by two developers to the same area of the source code [3]. Simply juxtaposing them in any order would be a correct merge, but an unstructured tool is not able to do that as it recognizes program lines instead of declarations.

To improve merge accuracy, researchers have proposed tools that consider the structure of the source code being merged [3–13]. Such structured merge tools go beyond simple textual, line-based, analysis by parsing the code and handling abstract syntax trees (ASTs). They employ tree matching and combination algorithms. Because of that, they are language-specific and demand significant

https://pauloborba.cin.ufpe.br/publication/2024semistructured_merge_with_language-specific_syntactic_separators/

Helping people focus on the merge conflicts that really matter!

http://is.gd/lSOeN7

Semistructured merge

Merge and Code Review

Paulo Borba
Informatics Center
Federal University of Pernambuco

pauloborba.cin.ufpe.br