Building Implicit Vector Representations of Individual Coding Style

Vladimir Kovalenko
JetBrains Research

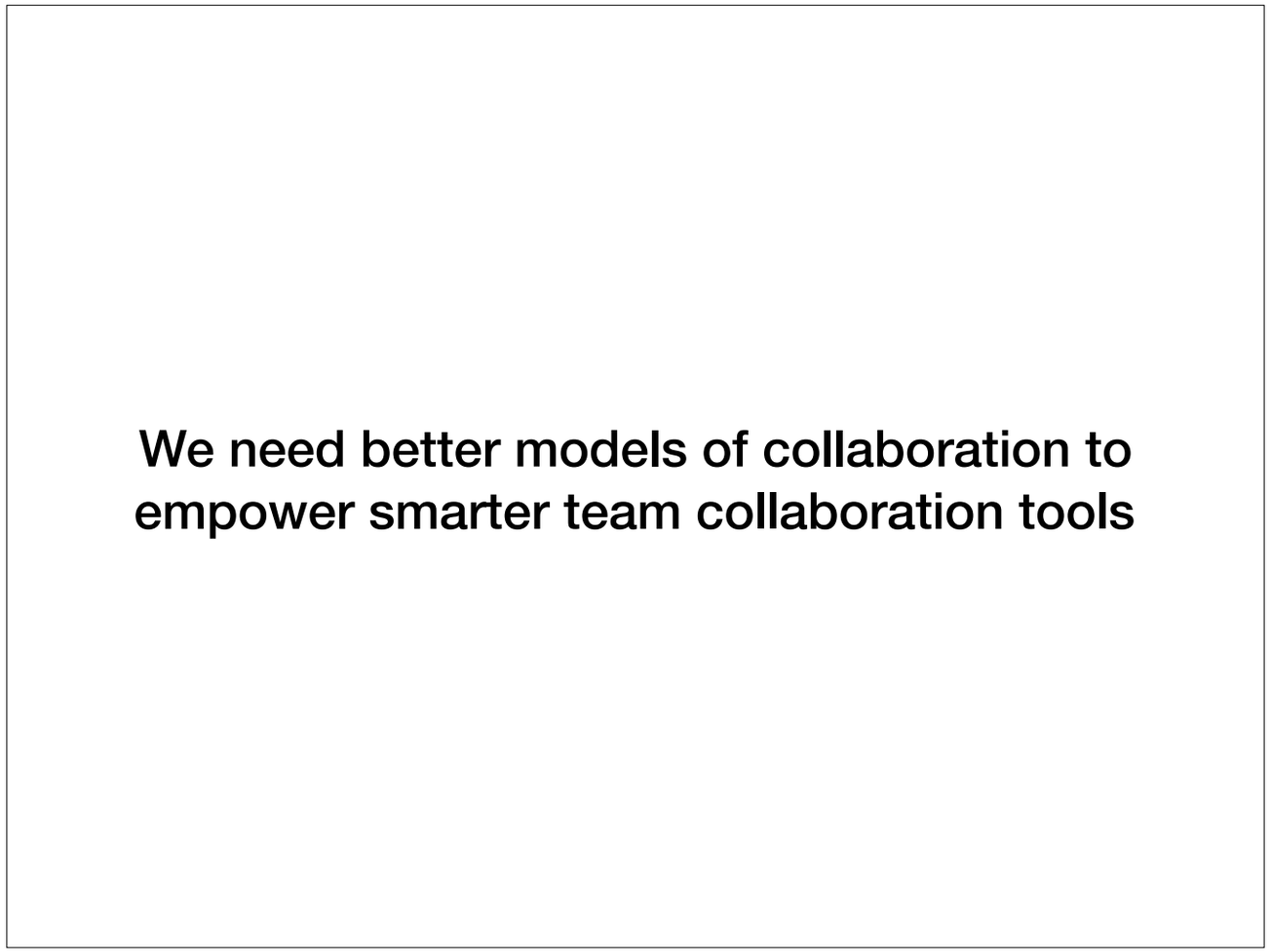
Egor Bogomolov
JetBrains Research

Timofey Bryksin
JetBrains Research

Alberto Bacchelli
University of Zurich

Hello everyone,

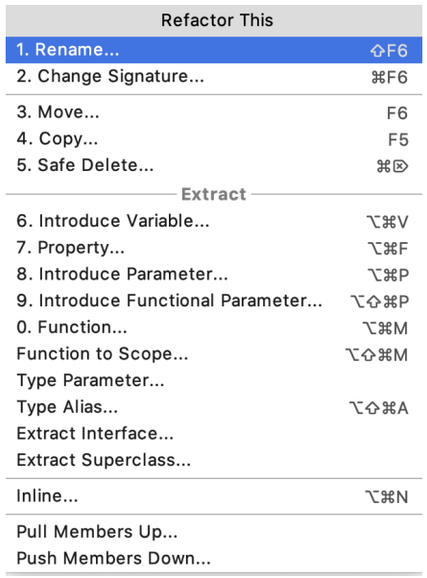
My name is Vladimir. I'm going to present a work titled "Building Implicit Vector Representations of Individual Coding Style", which is a collaboration between my colleagues and myself at JetBrains Research in Saint Petersburg and Amsterdam, and University of Zurich.

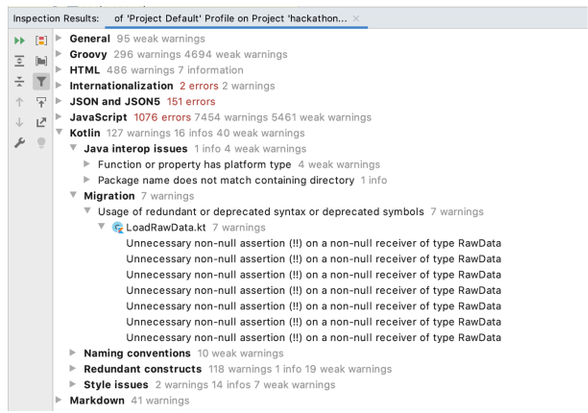


Let me start from afar. We don’t have time to go deep into detail anyway, so I’m going to use a good chunk of the 10 minutes we have to share — and motivate — one important message.
My goal with this talk is not to tell you many details about this piece of work, but rather to convey this single message — and spark up a discussion.

We need better models of collaboration to empower smarter team collaboration tools.

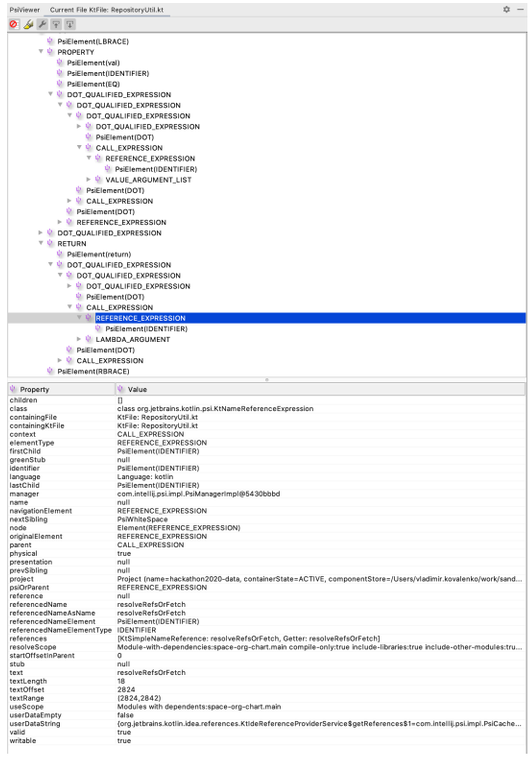
IDEs are powerhouses of code analysis





We are all familiar with IDEs, that offer plenty of options for code manipulation and analysis.
One can refactor code, or find and fix a thousand issues, with just a single click.

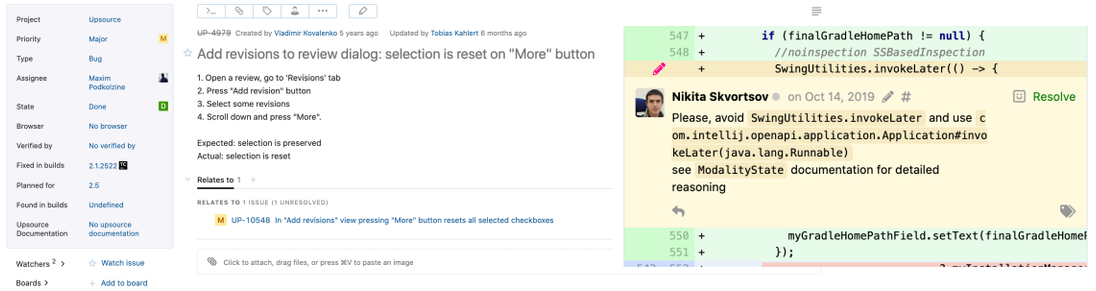
IDEs are powerhouses of code analysis



Thanks to rich internal representations of code

What enables IDEs to do incredible things with code are rich internal models of code and various connections between its elements.
For example, the IntelliJ platform and IDEs that rely on it all use PSI, a tree-based representation, that powers most of the refactoring and analysis features.

What about team collaboration tools?



Team collaboration tools mostly “just” present data as is

However, software engineering is not just about writing, reading, and maintaining code.
I’m happy I don’t have to sell the idea to this audience.
Software engineers use tools to collaborate, manage tasks, and socialize.

One thing that makes collaboration tools different from IDEs is relative simplicity. The vast majority of these tools and their features, essentially, provide a way to manipulate and display rather simple data like text, and don’t build or use any models of processes they support. As the result, they don’t really provide any smart features to users.

Smarter team collaboration tools

Smarter team collaboration tools require **rich models of interpersonal communication** in teams

To sum things up, IDEs are about code, and have great code models. Team collaboration tools are about communication and collaboration, but they don’t have any models of these processes.

And here comes my message again!

To make team collaboration tools smarter, we need to empower them with rich models of interpersonal communication and collaboration.

Smarter team collaboration tools

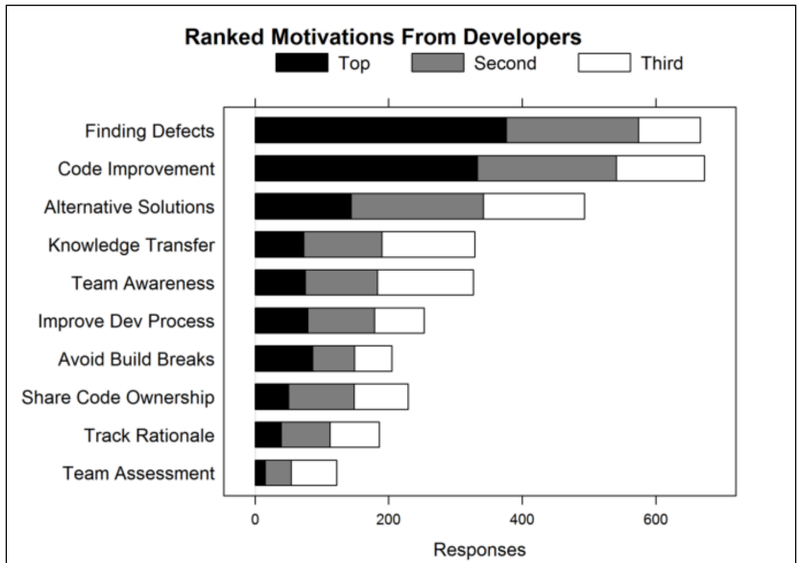


Figure 3. Developers' motivations for code review.

Bacchelli and Bird, ICSE 2013

When I say `smarter`, I don't just mean `more automated`. I also mean "better tailored to users' needs".

This is a picture from a rather famous paper on expectations and outcomes of code review, by Alberto Bacchelli and Chris Bird. It presents the most frequent developers' motivations from the code review process.

Smarter team collaboration tools

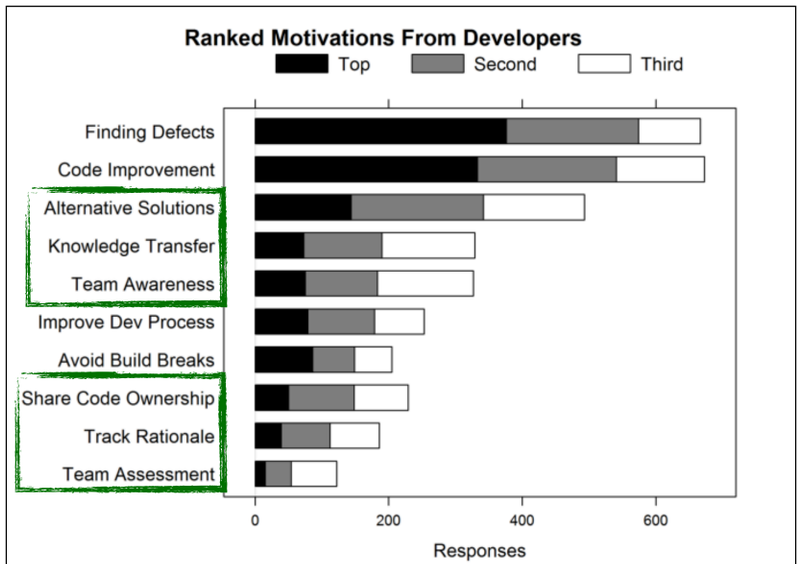


Figure 3. Developers' motivations for code review.

Bacchelli and Bird, ICSE 2013

Please note how most of the reported motivations are not exactly about code, but rather about sharing knowledge within the team.
Modern code review tools are far from offering substantial dedicated support in these processes, with one underlying reason that they just don't model these processes.

This finally brings me to our work.

Goal: build vector representations of individual coding style

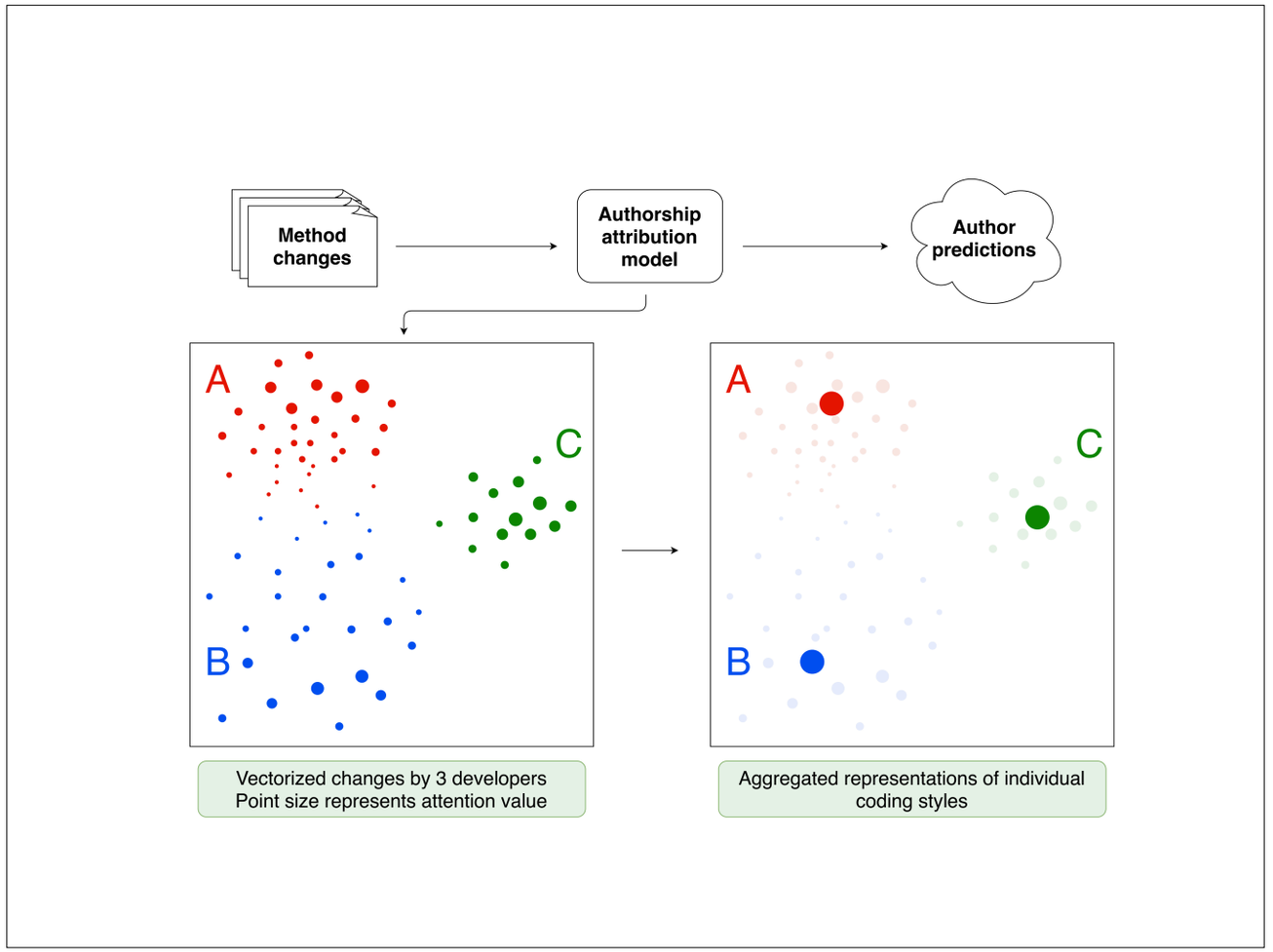
Why: to trace style evolution and learning

Coding style is *whatever makes one's code different from peers' code*

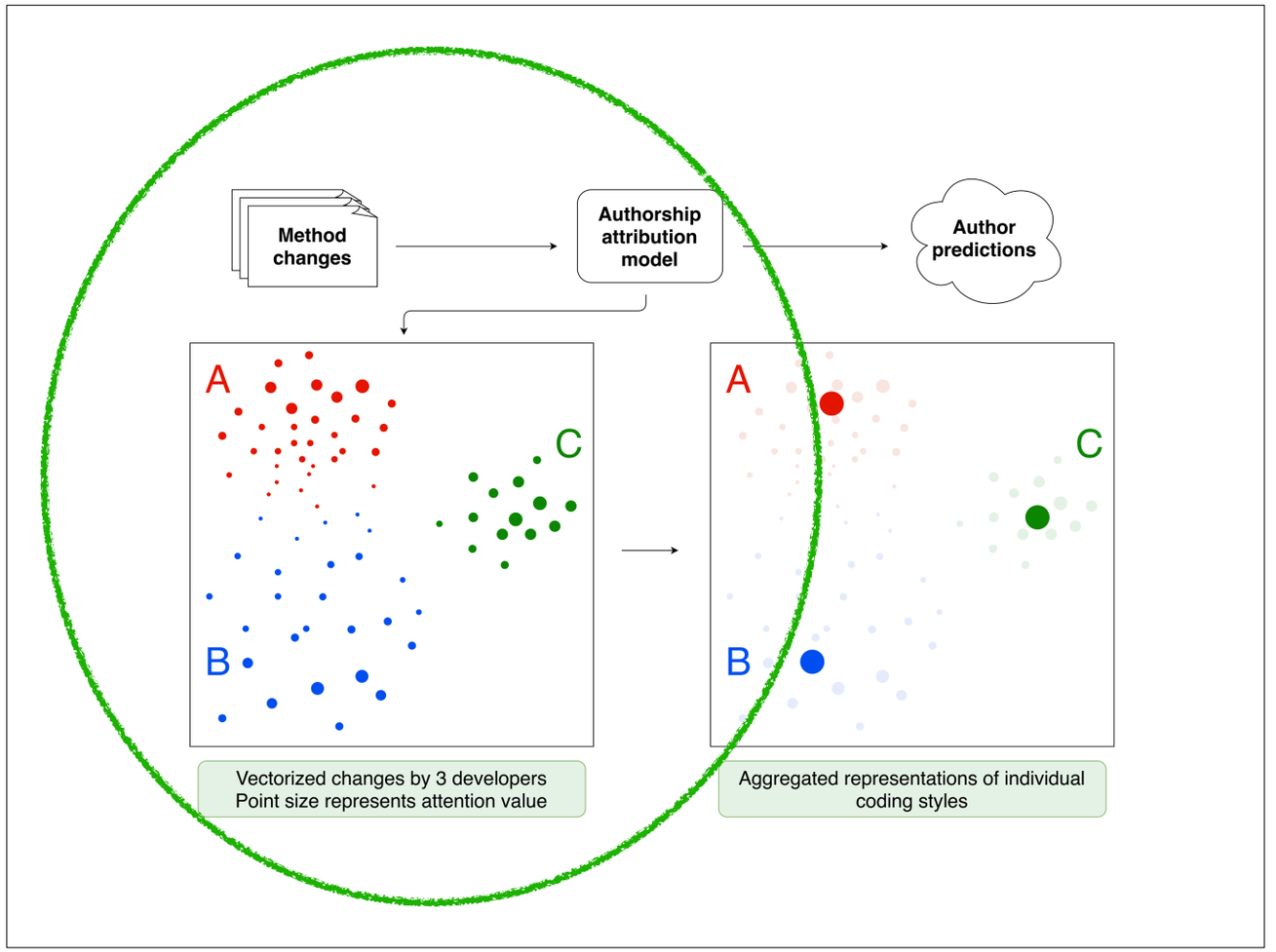
The goal of this work was to try to build representations of people's individual coding style. This is an important element of the collaboration models that I've talked about earlier.

With such representations in place, it would be possible to trace evolution of individual coding style and even detect learning between individuals, which brings the tools closer to offering assistance or steering this process where necessary. Think of things like bus factor: tools of the future may be able to assist with this.

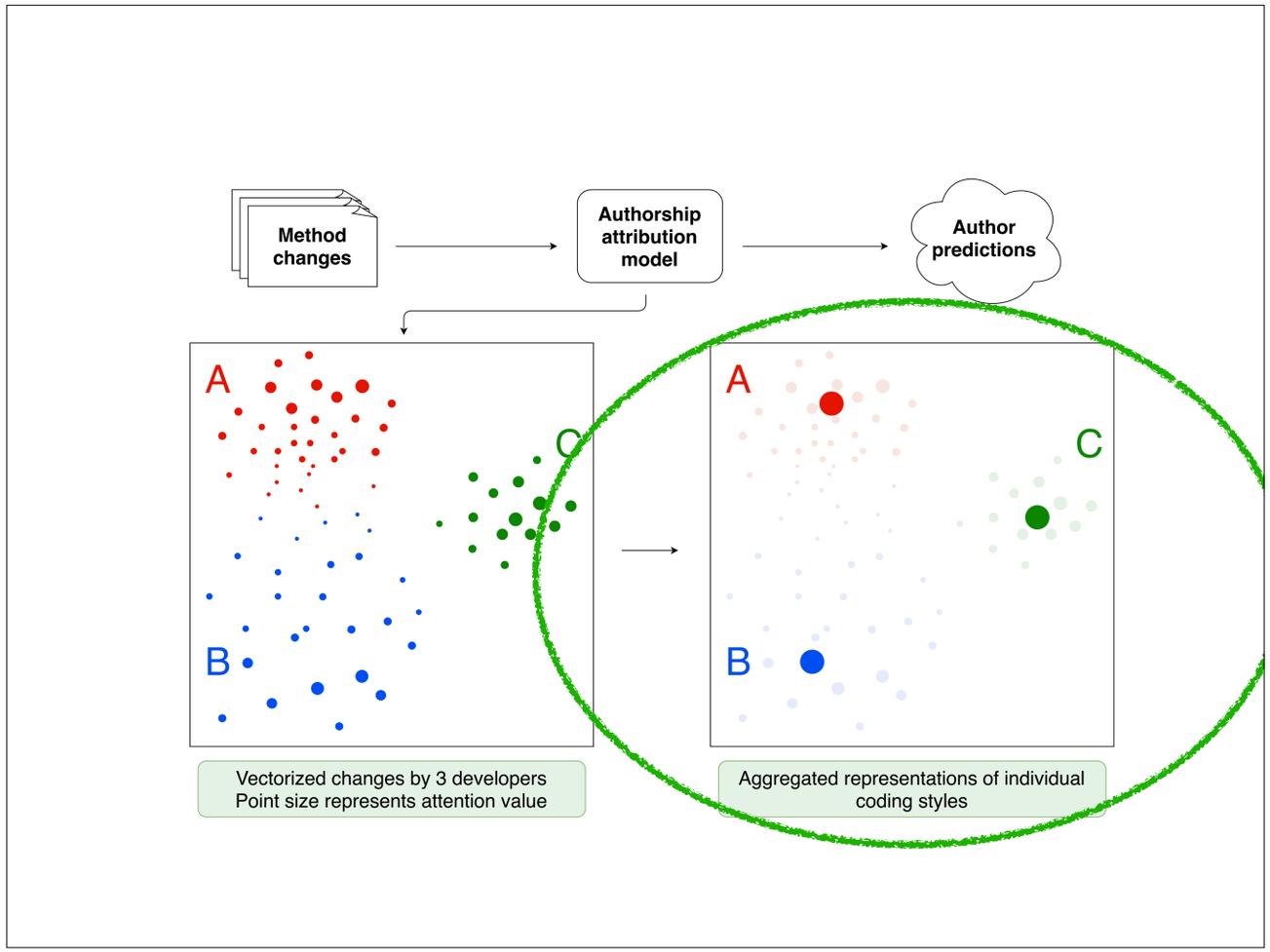
In this work, we define coding style as whatever makes one's code different from peers'.
This way, we are not limited to any particular set of metrics, unlike some existing work on code stylometry: every set of metrics is naturally limited and may miss some important aspects.



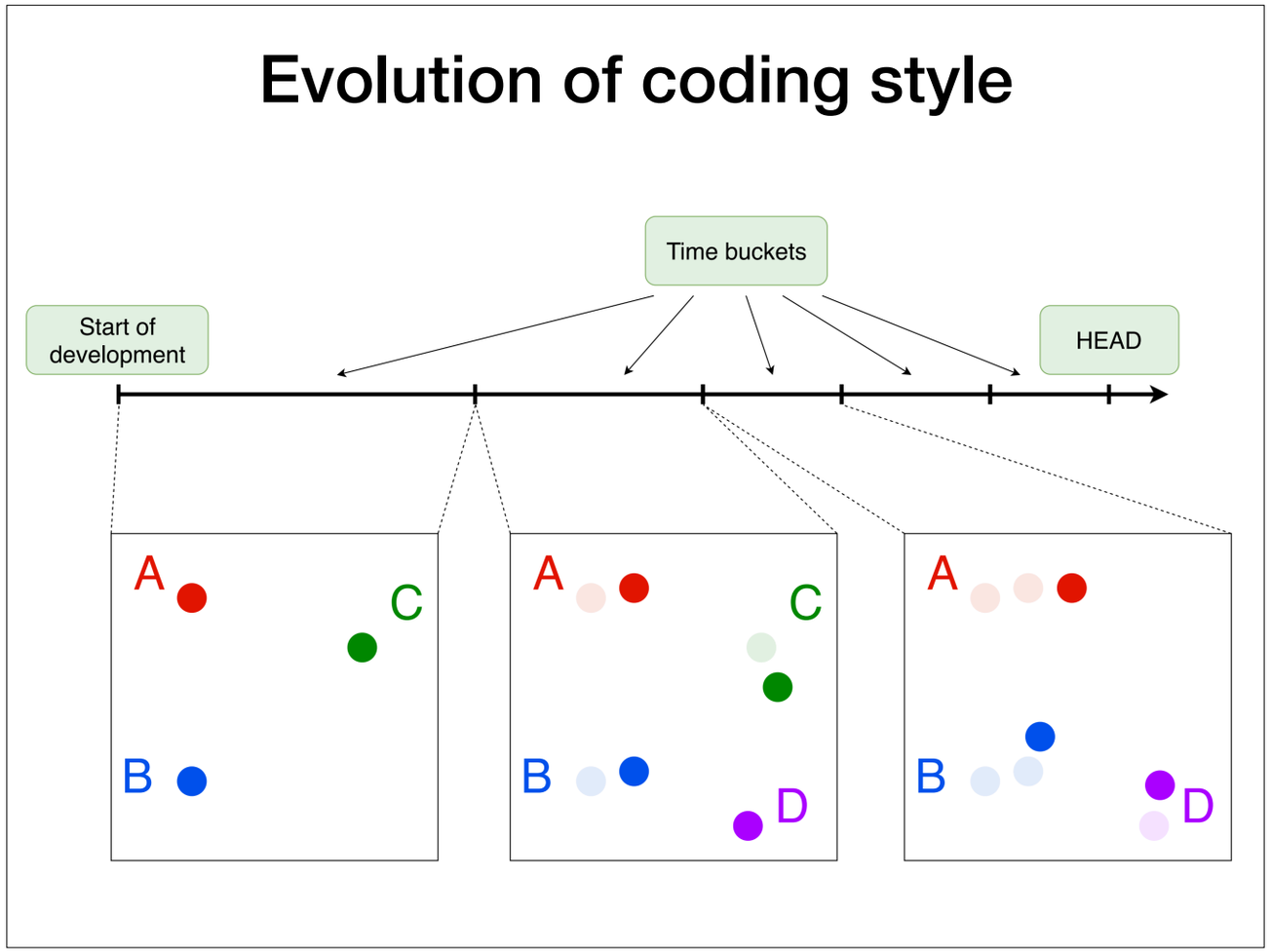
Here is an overview of our pipeline.



First, we train a code authorship attribution model. Essentially, the model learns to vectorize code changes so that changes authored by different developers are separated as well as possible.

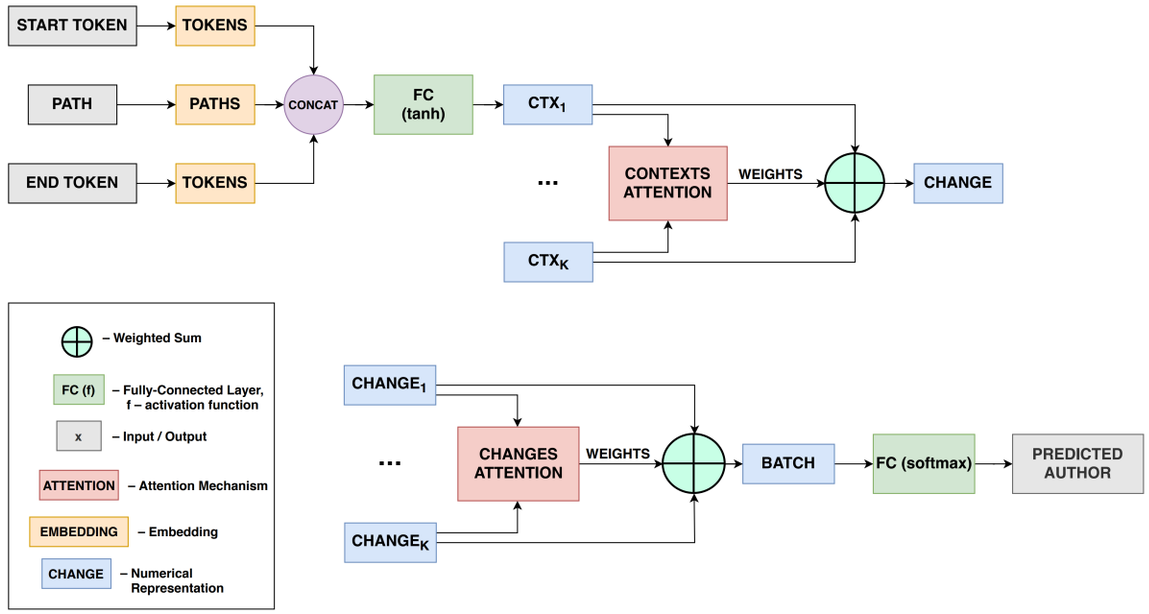


Later, we aggregate representations of changes over a certain period of time into a vector that represents the corresponding developer.

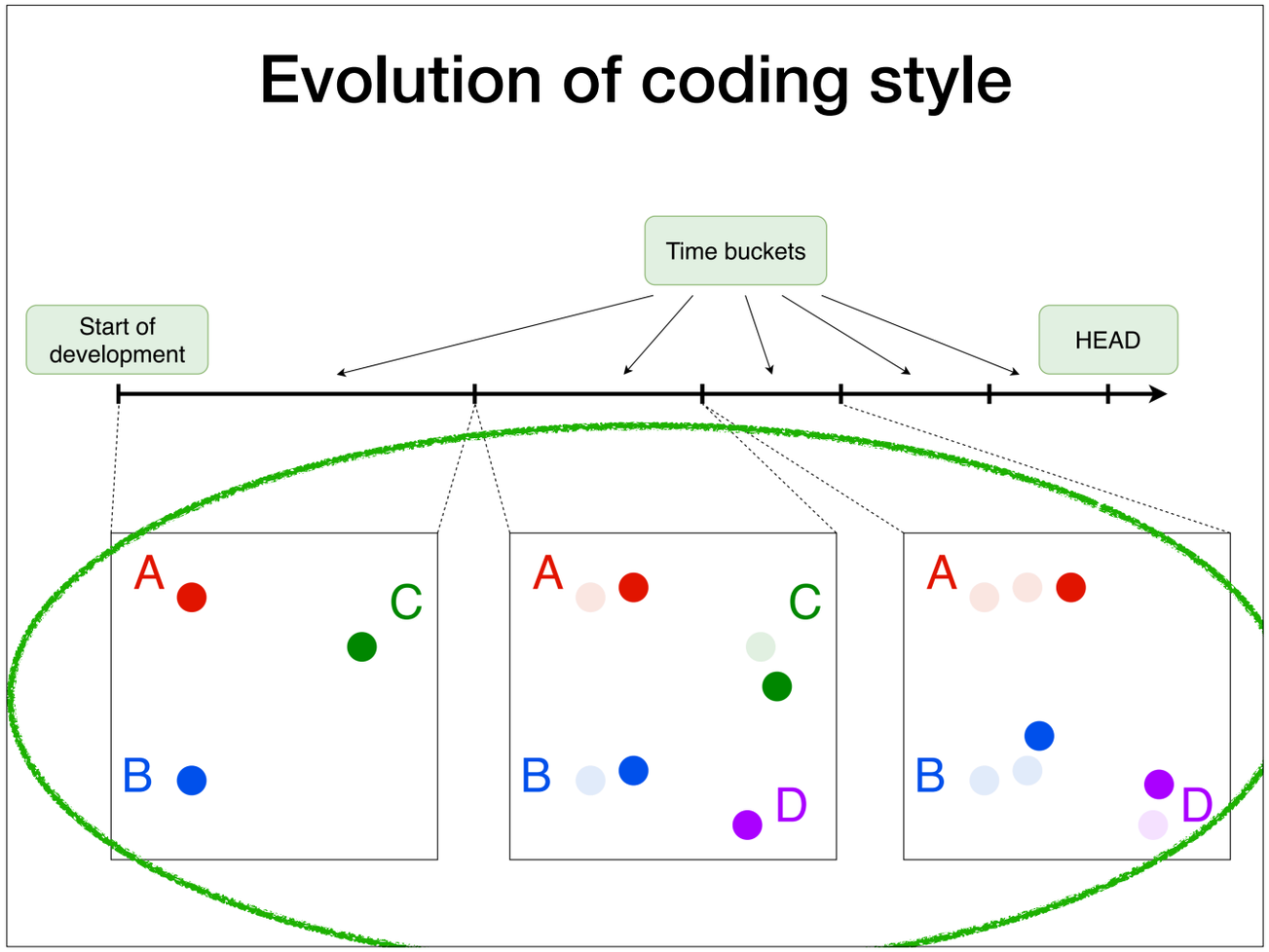


As the result, we have multiple snapshots of coding styles from multiple developers in the team, with each shapshot corresponding to a particular time period. Our hope is, we can make sense of relative positions of representations and their relative movements and interpret them as proximity of individual coding styles, or their evolution to become more similar or different.

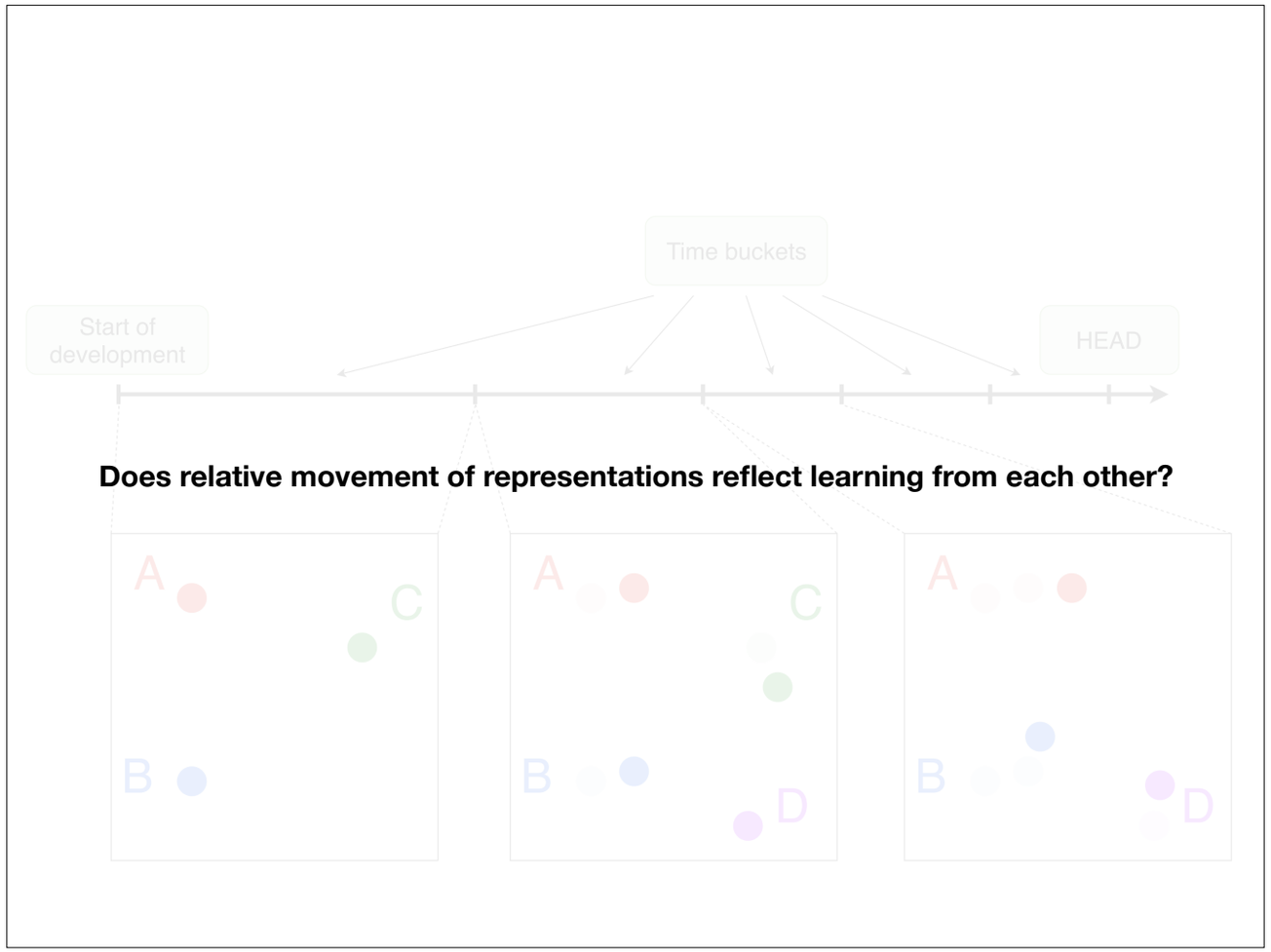
Authorship Attribution Model



I won’t go too deep into details of the authorship attribution model. I will only say that it’s modified code2vec, and encourage you to read more in the paper.



So, the interesting part here is relative positions and relative movement of representations. We hope to be able to make sense of it, but can we, really? If A and B are close, or are moving closer to each other, does that mean that there is learning between them?



We investigated it in the evaluation part of this study.

Developer survey

IntelliJ IDEA data and team survey

- **Name a colleague in your team from whom you have learned some elements of coding**
- **To what extent have you learned from the person above?**
  1 (Very little: I can barely identify any particulars)

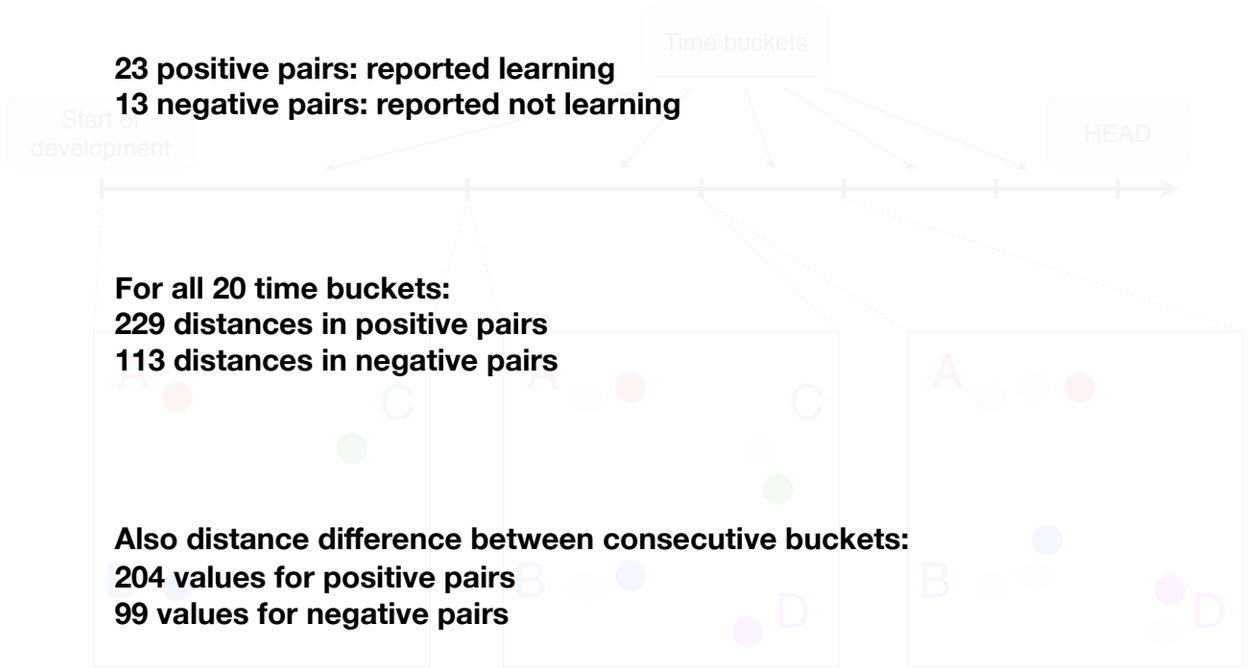
  ---

  7 (Very much: I owe several elements of my coding to learning from this person)
- **When do you think you have been learning from them the most actively?**

Also asked to name people they didn't learn from

We computed the representations from contribution history of a large project — IntelliJ IDEA Community, and asked the developers to indicate peers whom they have learned from, and those they are sure they didn't learn from.

Mapping survey results on data

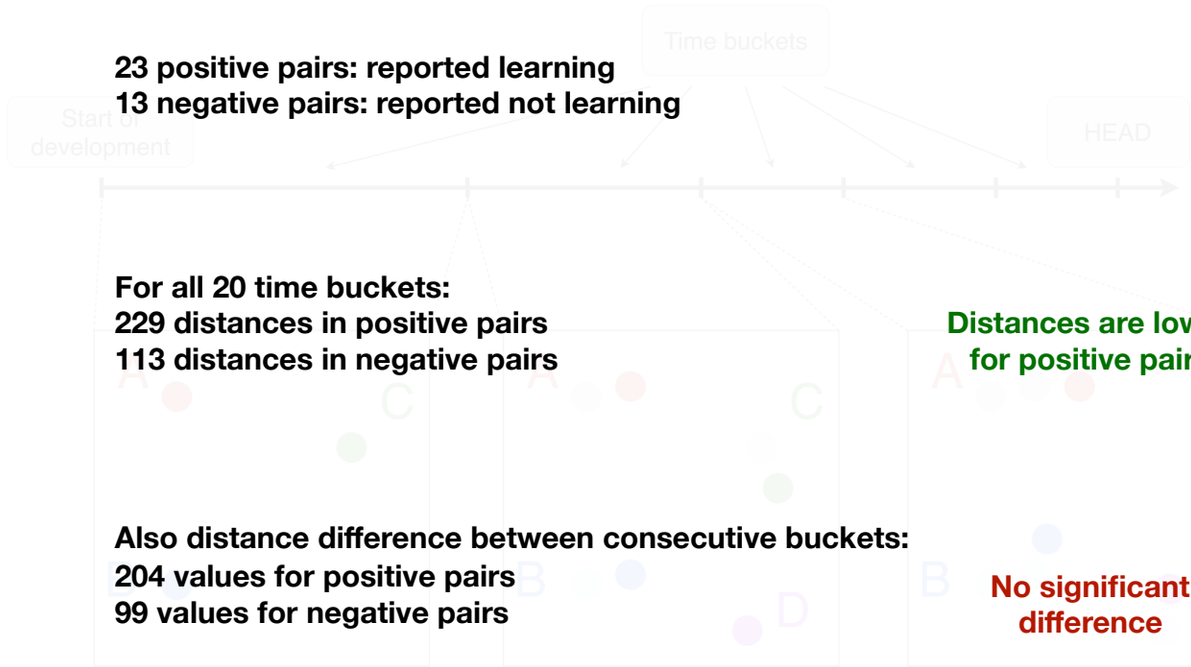


The survey yielded 23 pairs of people who reported learning, and 13 pairs of people who reported not learning from each other.
For every pair, we calculated distances between their representations in 20 different time buckets, and changes in these distances over time.

Finally, we compared distributions of distances and their differences to see if they are different if people learn from each other.

Mapping survey results on data

23 positive pairs: reported learning
13 negative pairs: reported not learning



For all 20 time buckets:
229 distances in positive pairs
113 distances in negative pairs

Distances are lower for positive pairs

Also distance difference between consecutive buckets:
204 values for positive pairs
99 values for negative pairs

No significant difference

We found that distances between people one of whom reported learning from the other are lower than in pairs where one reported not learning from the other.

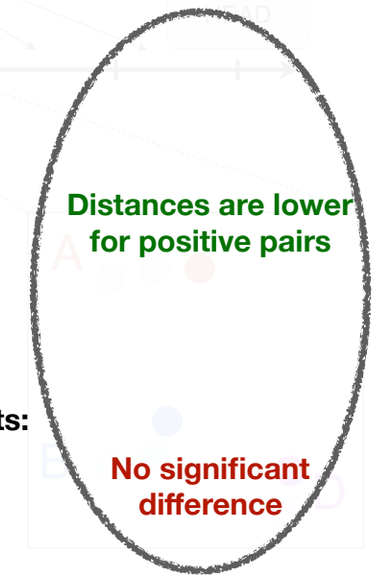
We found no difference in distribution of differences of distances.

Mapping survey results on data

23 positive pairs: reported learning
13 negative pairs: reported not learning

For all 20 time buckets:
229 distances in positive pairs
113 distances in negative pairs

Also distance difference between consecutive buckets:
204 values for positive pairs
99 values for negative pairs



Learning is represented to an extent

From this, we conclude that our code style representations may capture learning between individual developers to an extent.

Summary

- **We need rich models of collaboration to improve team tools: they are not so smart yet**
- **We build representations of individual contribution style purely from technical records**
- **Our representations capture learning between developers to an extent**
- **This is an exciting research direction: team tools will evolve one day**

Just read the bullet points :)

Problems

- **Noise**
- **Resource consumption**
- **Low evaluation quality**
- **Context**

There are a few issues I have to mention in the approach.

First off, the representations are noisy, thanks to the random nature of the ML model initialization. We ran the whole pipeline 30 times and did some filtering to smoothen the data.
This made the whole pipeline, that is already rather hungry for computing power, very slow to run. The whole process takes about a day to compute on a normal machine.

Our evaluation involves human input, so we had to limit it to 1 project. It could have been broader for more reliable results.

Finally, the authorship attribution model distinguishes between people not just on style, but also on context of their changes. While we took measures to account for that, it is not clear whether context can be eliminated completely. We explore this rather deeply in the paper.

That’s it.

Thank you!