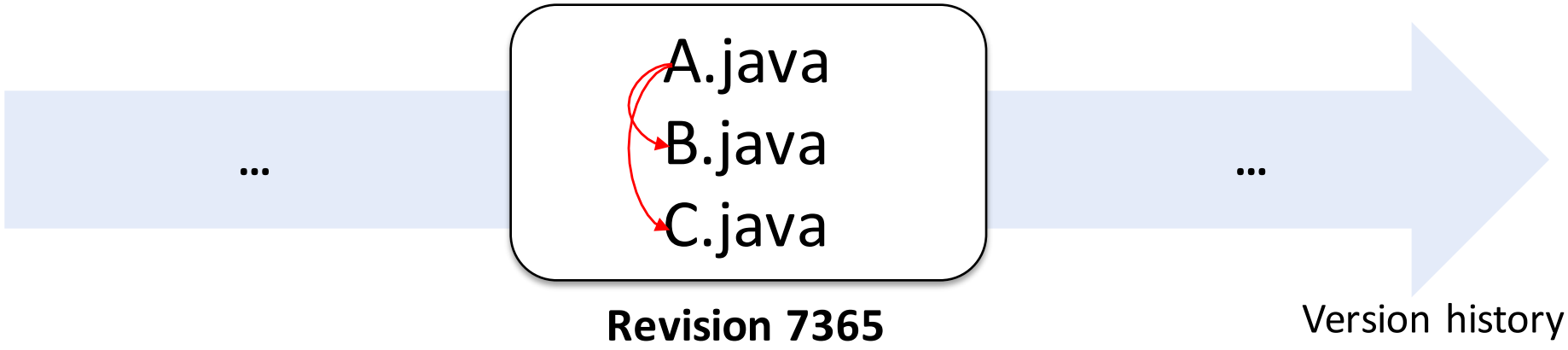
ASE 2014

An Empirical Study on Reducing Omission Errors in Practice

Jihun Park[1], Miryung Kim[2], Doo-Hwan Bae[1]

1. KAIST, South Korea
2. University of California, Los Angeles (UCLA), USA

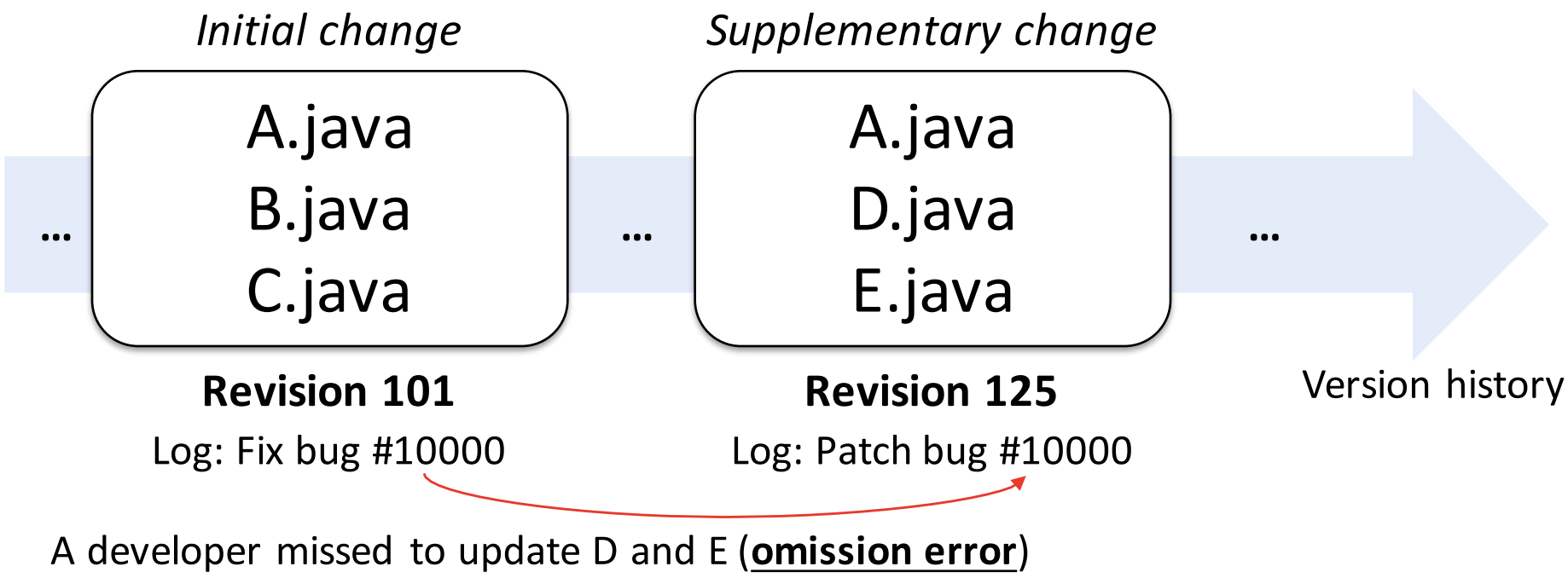
Predicting co-changed entities



Can we predict an additional change location in a transaction?

- **Change coupling (mining SW repositories)**: Zimmermann et al., Ying et al., Hassan and Holt, Herzig and Zeller
- **Structural dependency**: Robillard, Saul et al.
- **Cloning-based relationship**: Nguyen et al.

Predicting omission errors



How can we predict the supplementary change location, given the initial change location?

Key contributions

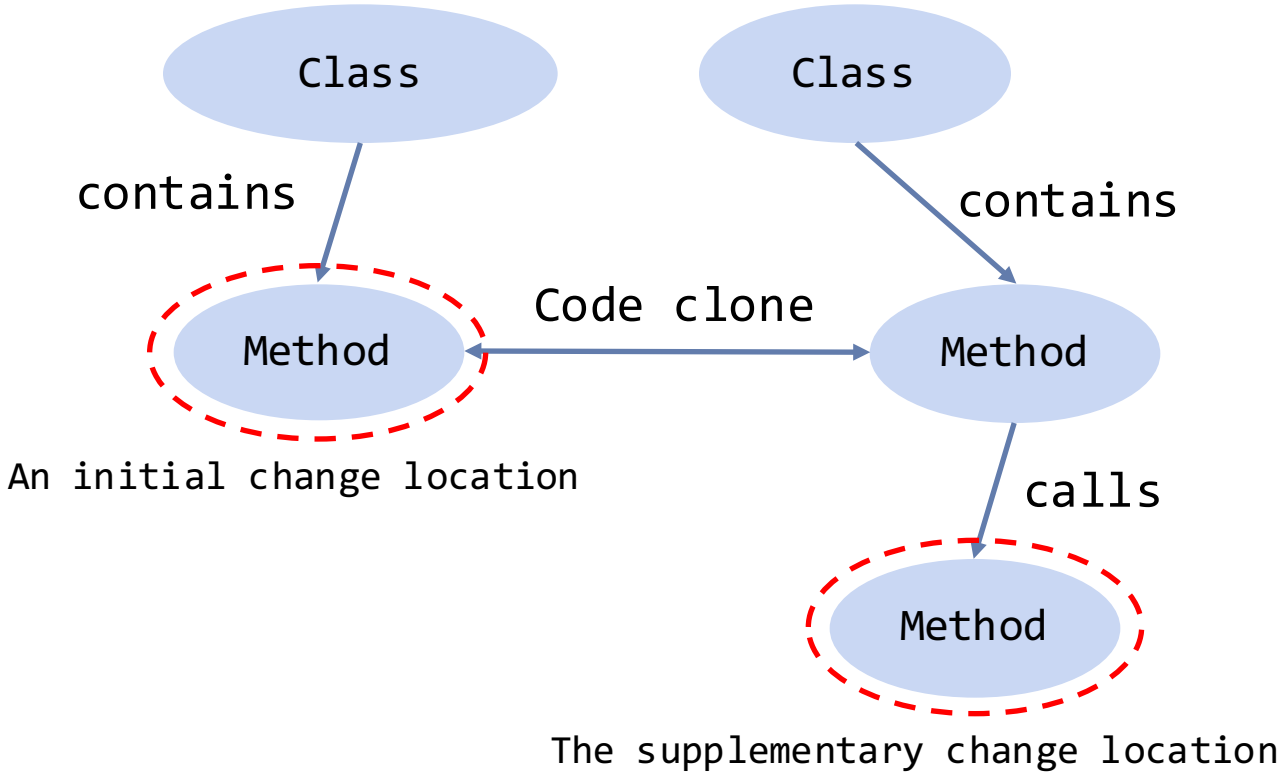
- To systematically investigate a real-world supplementary patch data set, we suggest a graph representation *change relationship graph (CRG)*.

1. While a single trait is inadequate, combining multiple traits is limited as well.
2. A boosting approach does not significantly improve the accuracy.
3. There is no package or developer specific pattern.
4. There is no repeated mistake.

Change Relationship Graph (CRG)

Study subjects: Eclipse JDT core, Eclipse SWT, and Equinox p2

- Graph Nodes
  - Classes
  - Methods
- Graph Edges
  - Extends
  - Contains
  - Method invocation (calls, called by)
  - Historical co-change
  - Code clone
  - Name similarity

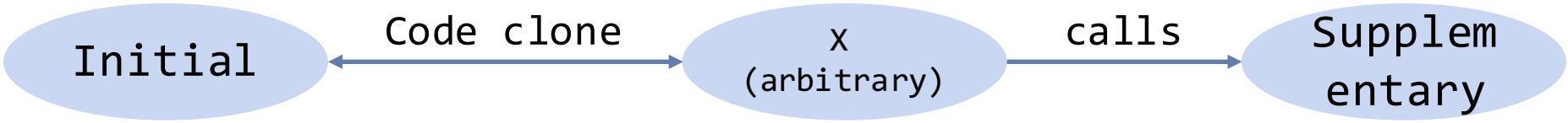


* M.K. Ripon Saha et al. A graph-based framework for reasoning about relationships among software modifications. TR 2014

Observation 1: While a single trait is inadequate, combining multiple traits is limited as well.

- Only 10% to 20% of supplementary change locations can be connected with *one edge* from initial change location.

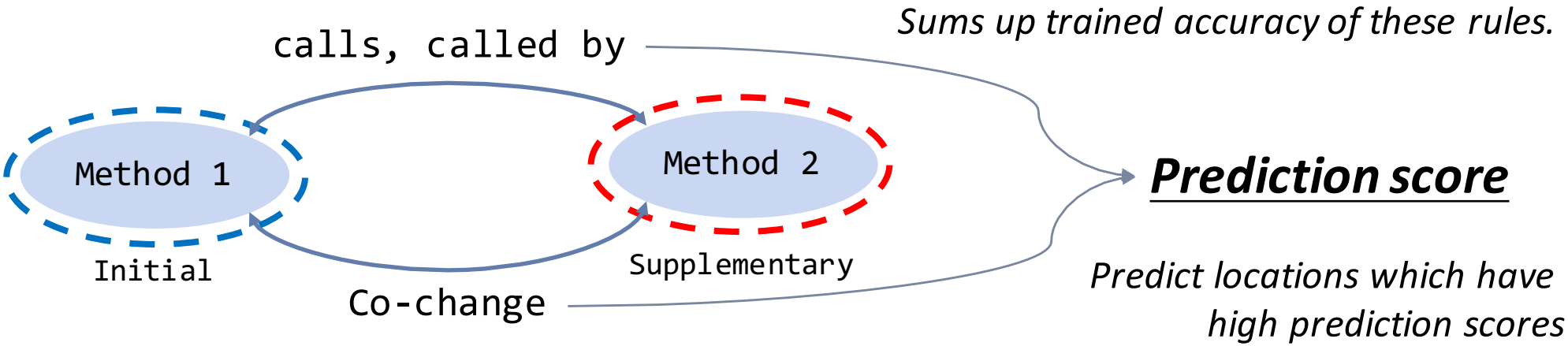
- Combining multiple traits as a prediction rule shows at most 10% accuracy



Combining multiple traits does not predict supplementary change locations accurately

Observation 2: A boosting approach does not improve the accuracy.

- We design a *boosting approach* that sums up trained accuracy of rules connecting initial and supplementary change locations to calculate ***prediction score***

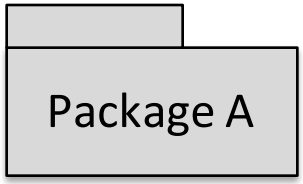


- This approach cannot accurately predict supplementary change location (at most 7% precision).

Boosting approach based on the past prediction accuracy also cannot accurately predict supplementary change locations.

Observation 3: There is no package or developer specific pattern.

- Package or developer specific rules might improve the prediction accuracy.



Accuracy of *code clone*: 40%
Accuracy of *co-change*: 10%

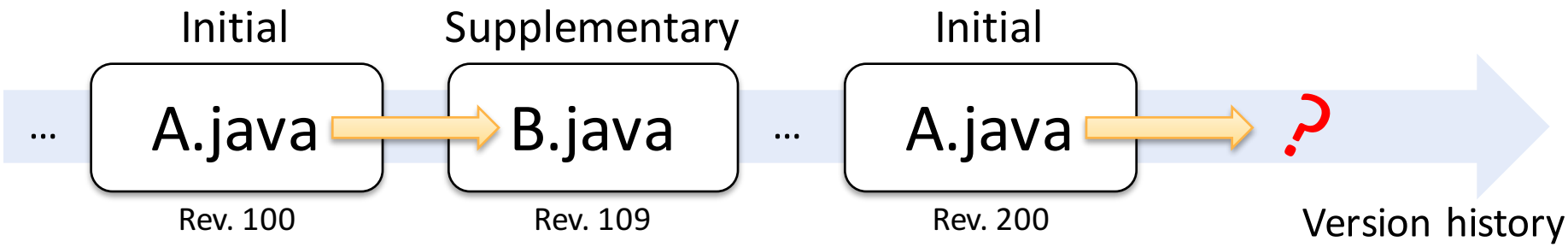
- We make boosting approaches based on package and developer specific prediction rules.

- The improvements is negligible; the highest accuracy improvement is only 1.2%

No package or developer specific pattern between initial and supplementary change locations exists.

Observation 4: There is no repeated mistake.

- There might be an uncovered relationship which can result in *repeated patterns*.



- The majority of patterns (78% ~ 96%) appear only once.
- 69% to 84% of initial change locations appear only once.

Developers rarely make repeated mistakes at the same location

Conclusion

- We systematically study omission errors using a real-world supplementary patch data set.
- Version history based pattern mining cannot be accurate at finding supplementary change locations.
- Past prediction accuracy, and package or developer specific information does not help.
- We share our skepticism that reducing real-world omission errors is inherently challenging.

ASE 2014

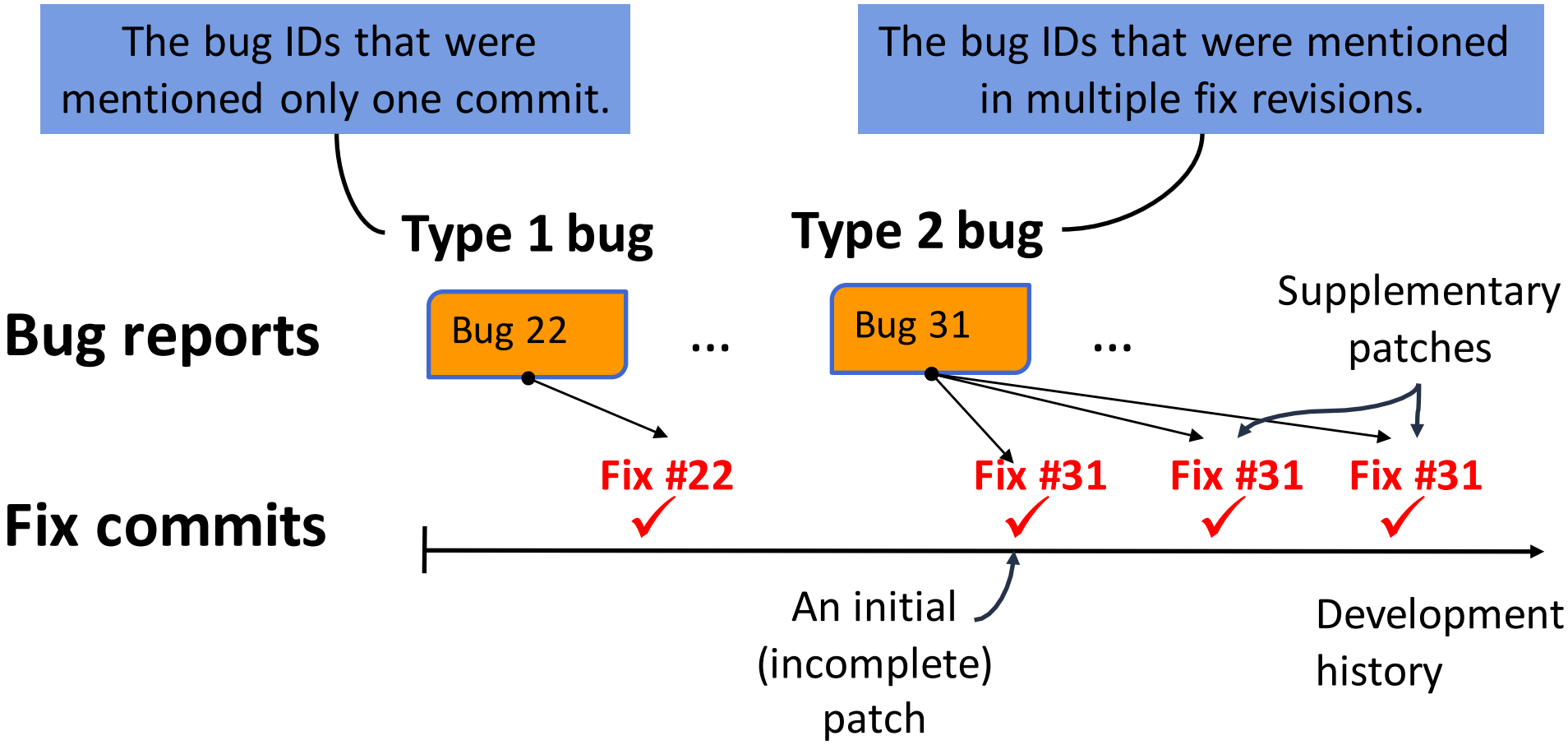
Thank you for listening

An Empirical Study on Reducing Omission Errors in Practice

Jihun Park[1], Miryung Kim[2], Doo-Hwan Bae[1]

1. KAIST, South Korea
2. University of California, Los Angeles (UCLA), USA

Supplementary Data Set



- We use Eclipse JDT core, Eclipse SWT, and Equinox p2
- Total 16 years, 13259 bugs (24.8% are Type 2 bugs on average)

Subject projects

	Eclipse JDT core	Eclipse SWT	Equinox p2
Study period	2001/06 ~ 2007/12	2001/05 ~ 2008/12	2006/01 ~ 2009/12
Total revisions	17009 revisions	21530 revisions	6761 revisions
# of bugs	1812	1256	1783
Type 1 bugs	2930 (77.04%)	3458 (74.00%)	1328 (74.48%)
Type 2 bugs	873 (22.96%)	1215 (26.00%)	455 (25.52%)

Evaluating a prediction method

- Precision, recall, and f-score
  - Predicted set P and Suggested set S
  - $Precision = \frac{|P \cap S|}{|P|}, Recall = \frac{|P \cap S|}{|S|}$
  - $F - score = 2 * precision * recall / (precision + recall)$

- Feedback
  - What portion of initial changes can obtain at least one suggestion?
  - P_b^m is derived using a prediction method *m* for bug *b,*
  - $Feedback = \frac{|\{\, b \in TypeIIbugs \mid 1 \leq |\{P_b^m\}| \,\}|}{|\{TypeIIbugs\}|}$