Performance of Defect Prediction in Rapidly Evolving Software

Davide G. Cavezza, Roberto Pietrantuono, Stefano Russo
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Università degli Studi di Napoli Federico II

Motivations (1/2)

- Defect prediction gives insight into product quality
  - Useful to make decisions on when to release
- Rapidly evolving development paradigms
  - Agile methods
    - Continuous Integration, Continuous Delivery
  - Short release-cycle required

Motivations (2/2)

- Classical “static” defect prediction: choose a model and cross-validate it on all the available data
  - There is no insight on how long the model remains valid
  - This is a key concern in rapidly changing software
- We propose a dynamic prediction model
  - The model is periodically retrained with the most recent data

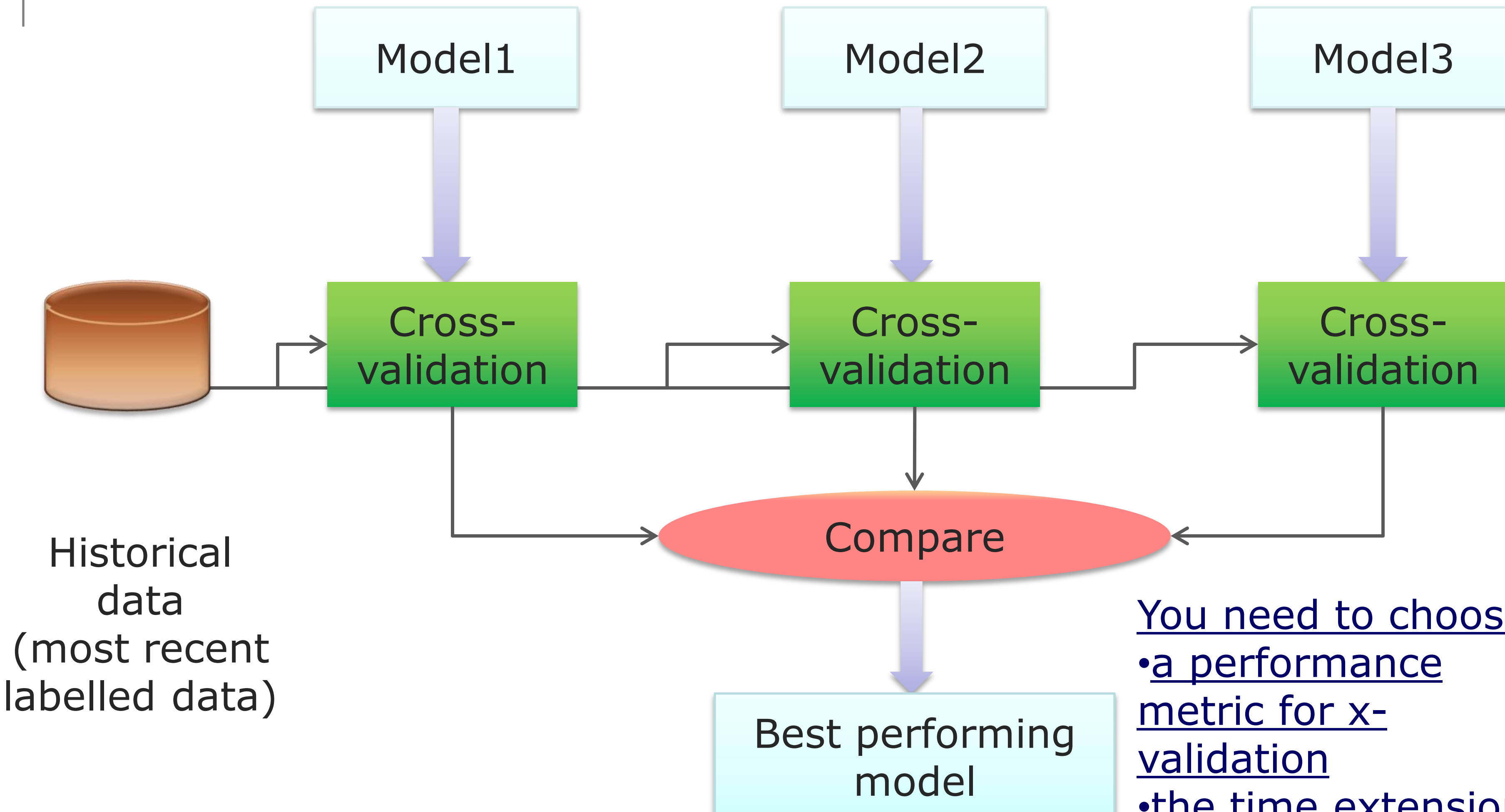
Commit-level defect prediction

- Relationship between a commit's features and its defectiveness
- Learning algorithms are used to predict if a commit is defective given its feature values
  - Supervised learning: the training set consists of commits whose defectiveness has been assessed

Dynamic prediction phases

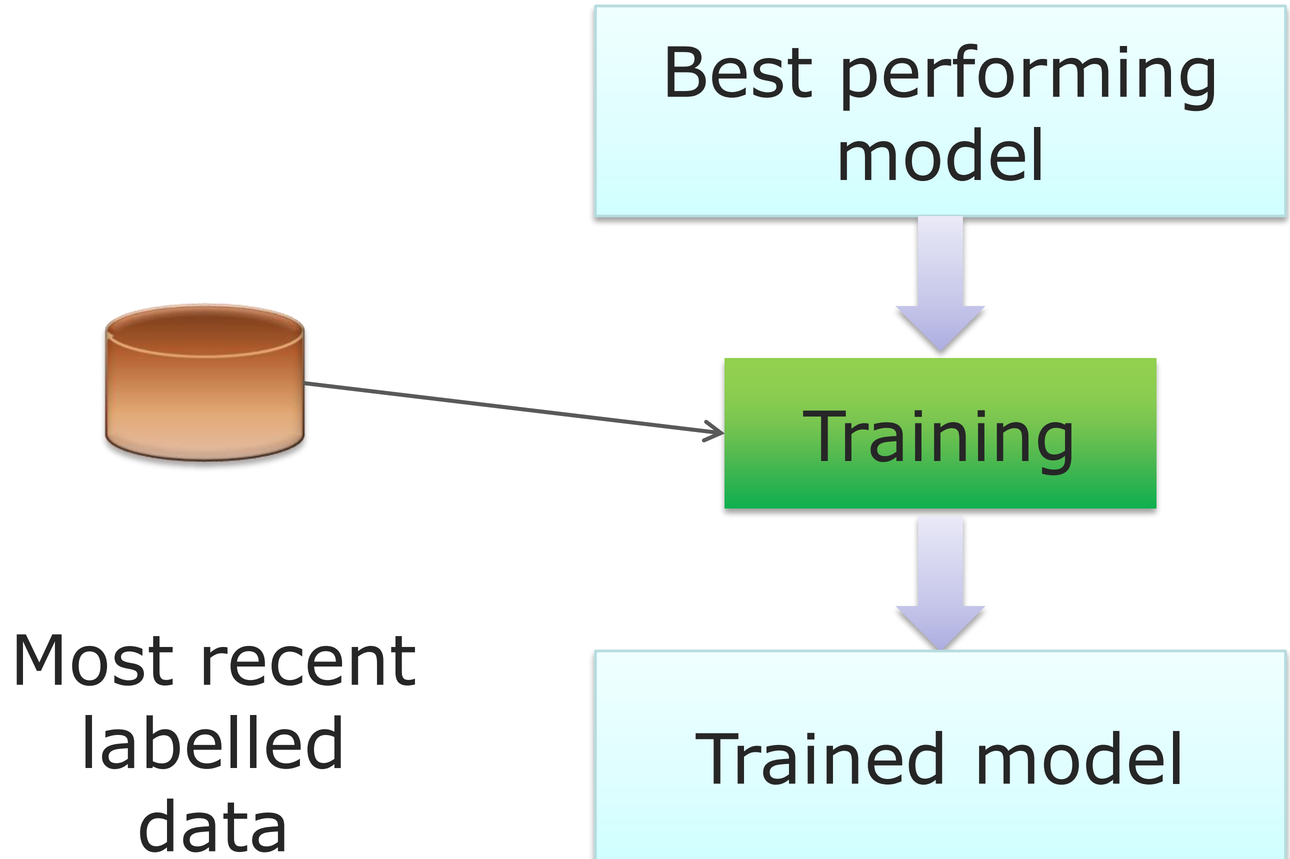
1. Model selection
2. (Re)training
3. Prediction
4. Evaluation

Model selection

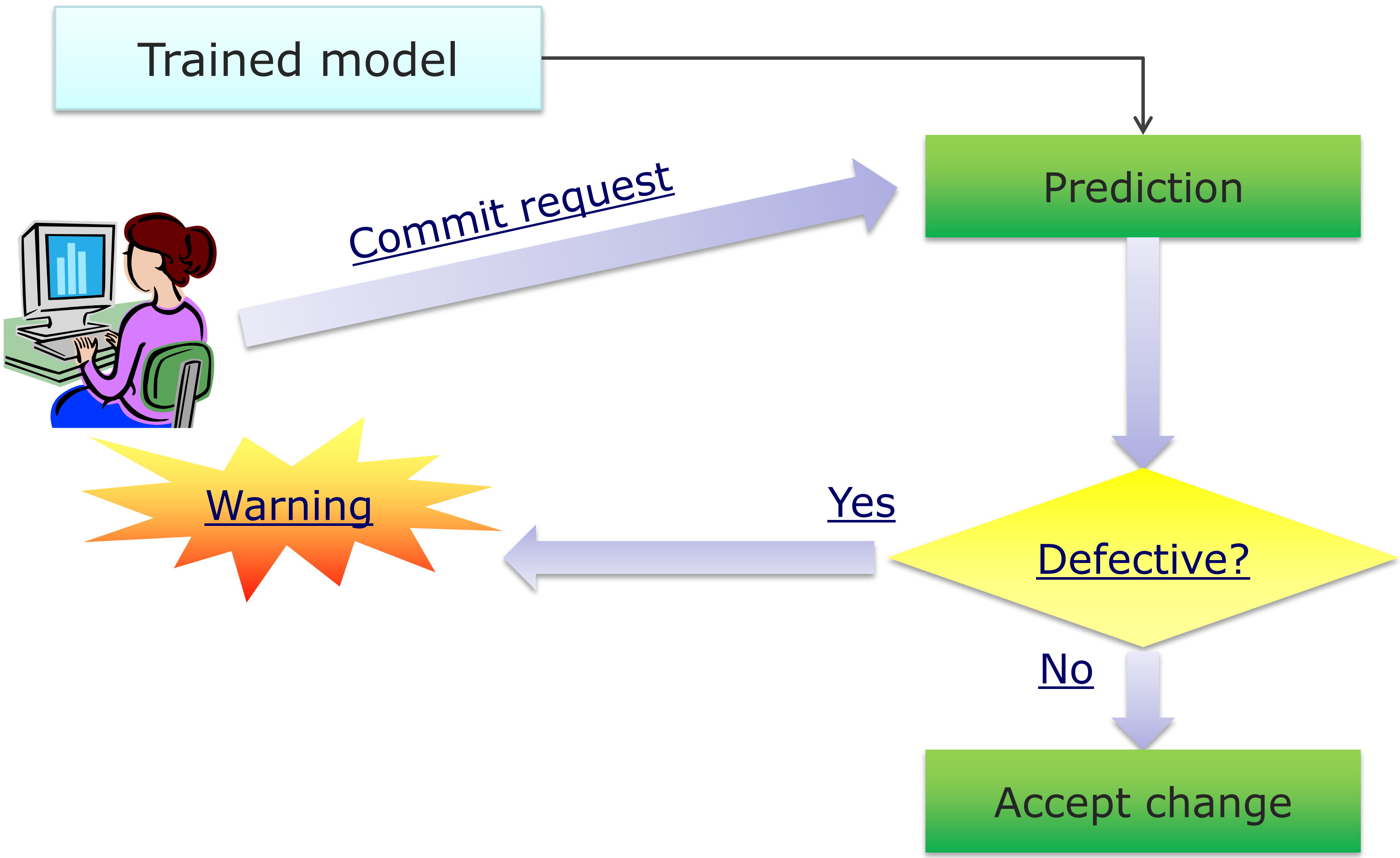


You need to choose:
- a performance metric for x-validation
- the time extension of the historical data

(Re)training

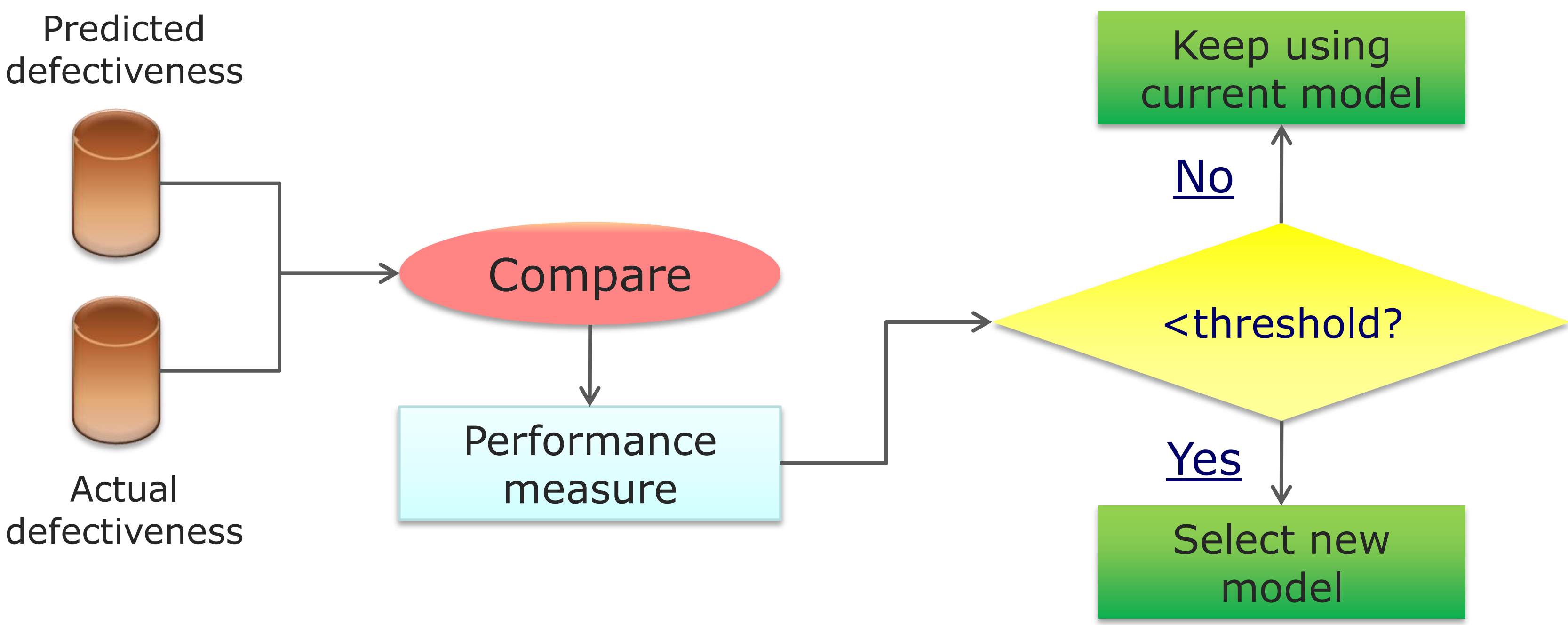

Best performing model
Training
Trained model
Most recent labelled data

Prediction


Trained model
Prediction
Commit request
Defective?
Yes
Warning
No
Accept change

Evaluation

- Executed periodically
  - Time interval between two evaluations must be chosen



Experimental setting (1/4)

- Eclipse JDT
- Commit data extracted from Git repository
- SZZ algorithm to distinguish defective and non-defective commits

Total commits	Timespan	Defective commits	Non-defective commits
26,009	From 2001-06-05 To 2014-12-13	13,984 (53.77%)	12,025 (46.23%)

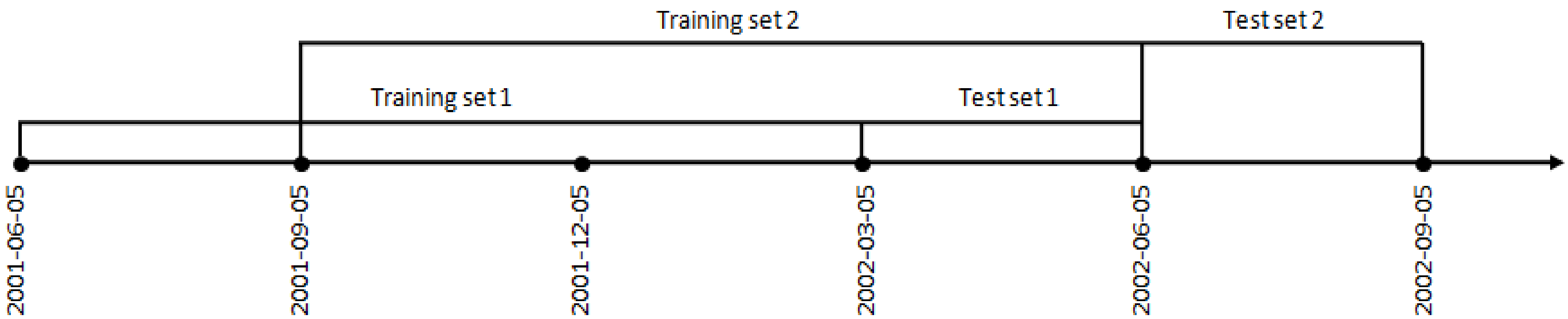
Experimental setting (2/4)

- Commit-level features

Number of modified files (NF)	Number of files modified in the commit
Entropy	Scattering of modifications throughout the modified files
Lines added (LA)	Number of lines added in the commit
Lines deleted (LD)	Number of lines deleted in the commit
FIX	Binary value indicating whether or not the commit is a bug fix
Number of developers (NDEV)	Number of developers that changed the files touched by the commit before the commit was issued
AGE	Average time interval between the current and the last change across all the involved files
Number of unique changes (NUC)	Number of unique commits that last changed the involved files
Experience (EXP)	Experience of the developer, measured as the number of changes previously committed by him
Recent experience (REXP)	Number of past commits of the same developer, each weighted proportionally to the number of years between that commit and the measured one

Experimental setting (3/4)

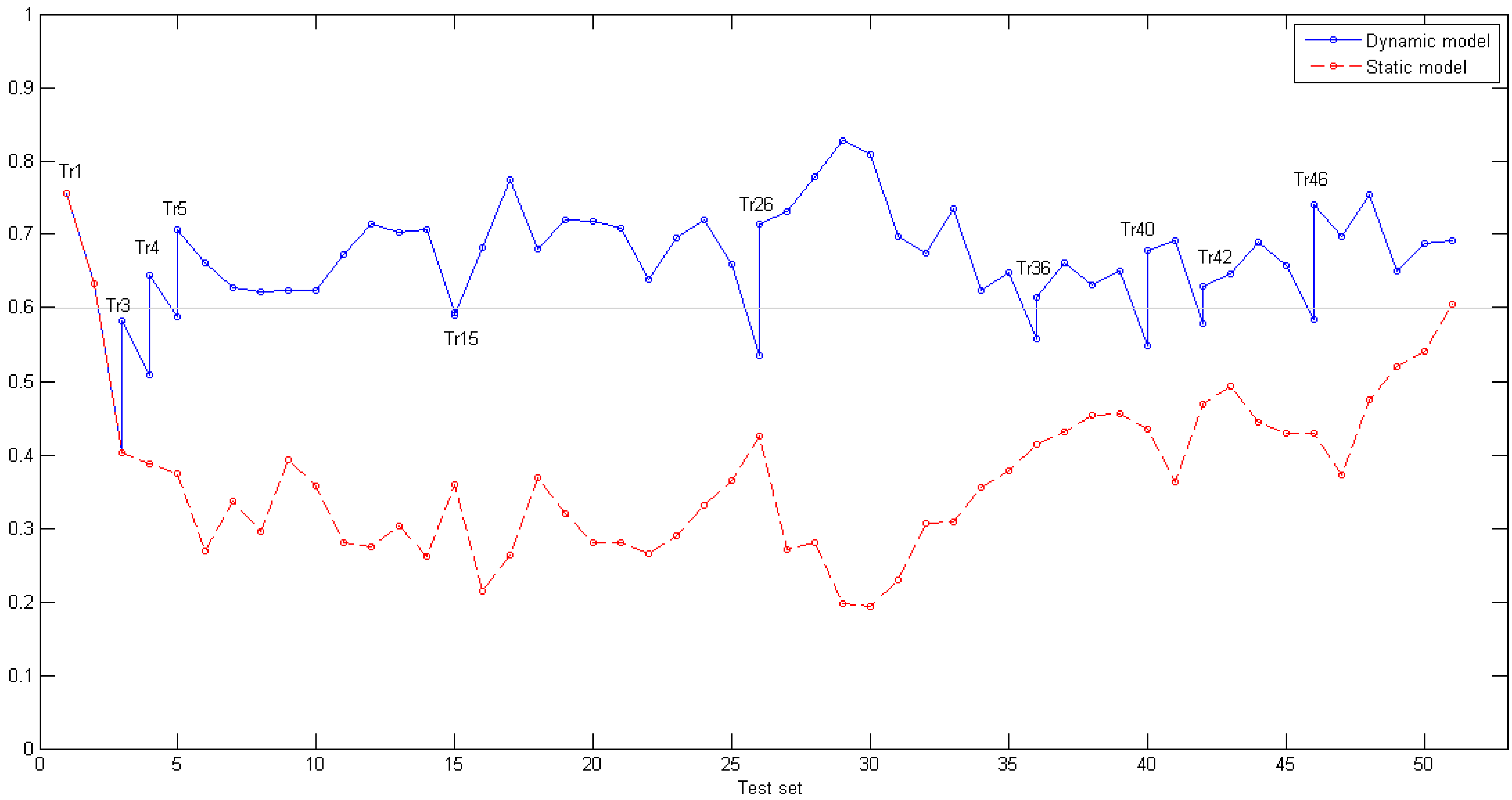
- Repartition of training and test sets:
  - Training sets duration: 9 months
  - Test sets duration: 3 months



Experimental setting (4/4)

- Models used:
  - J48
  - OneR
  - NaiveBayes
- Performance metric:
  - F-measure = $2 * \frac{precision * recall}{precision + recall}$

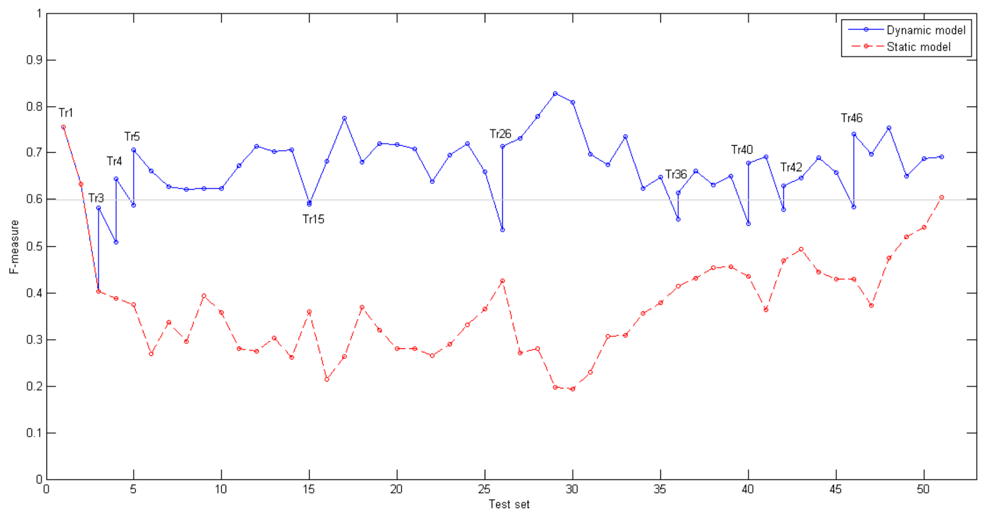
Results: Static vs Dynamic model



Discussion

Dynamic model outperforms static

But there are two situations in which neither can predict defectiveness with sufficient accuracy



Future challenges

- Assessment of the influence of parameters like
  - Training windows extension
  - Frequency of evaluations
  - Performance measure choice
- Problem: lack of knowledge on recent commit defectiveness

Thank you!

Questions?