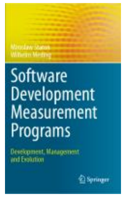
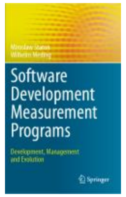


PREDICTING TEST CASE VERDICT USING TEXTUAL ANALYSIS OF COMMITTED CODE CHURNS

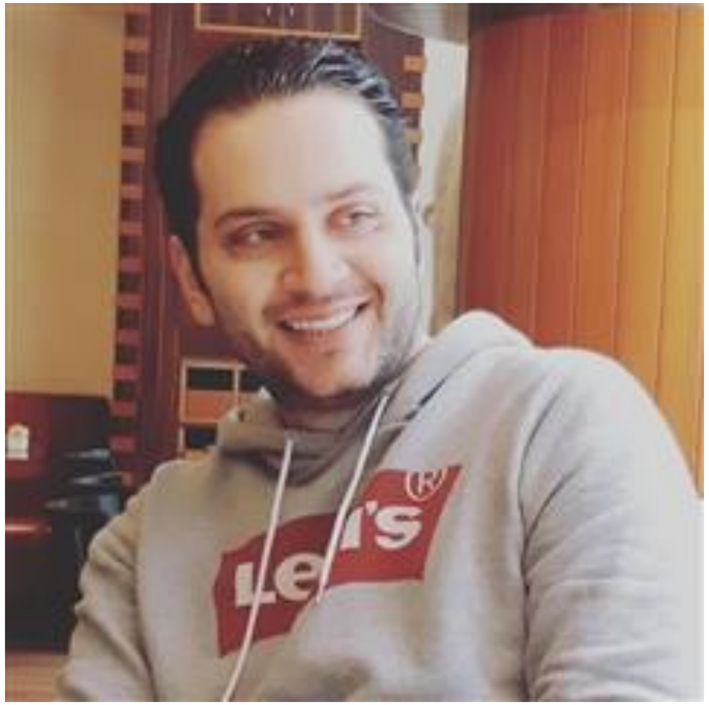
KHALED AL SABBAGH, CHALMERS | UNIVERSITY OF GOTHENBURG

REGINA HEBIG, CHALMERS | UNIVERSITY OF GOTHENBURG

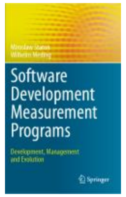
MIROSLAW STARON, CHALMERS | UNIVERSITY OF GOTHENBURG



About me



- **_Personal profile_**
  - ❑ *Khaled Al-Sabbagh*
  - ❑ *Syria*
- **_Academic background_**
  - ❑ *MSc degrees Management*
  - ❑ *MSc degree in Software Engineering*
  - ❑ *BSc Information Technology Engineering*
- **_Current work_**
  - ❑ *2nd year PhD student in Gothenburg University, Sweden*
- **_Contact Info_**
  - ❑ **_Khaled.al-sabbagh@gu.se_**
  - ❑ **_+46729250522_**



Context: testing in continuous integration

- Continuous integration often includes (regression) testing after every build
  - Frequent (every 10 minutes) integrations results in large number of test executions
  - Regression suites need to be small in order to reduce the cost of testing

- Continuous testing is often organized in several suites
  - Minimal suite after every build
  - Larger daily suite
  - Even larger weekend suite

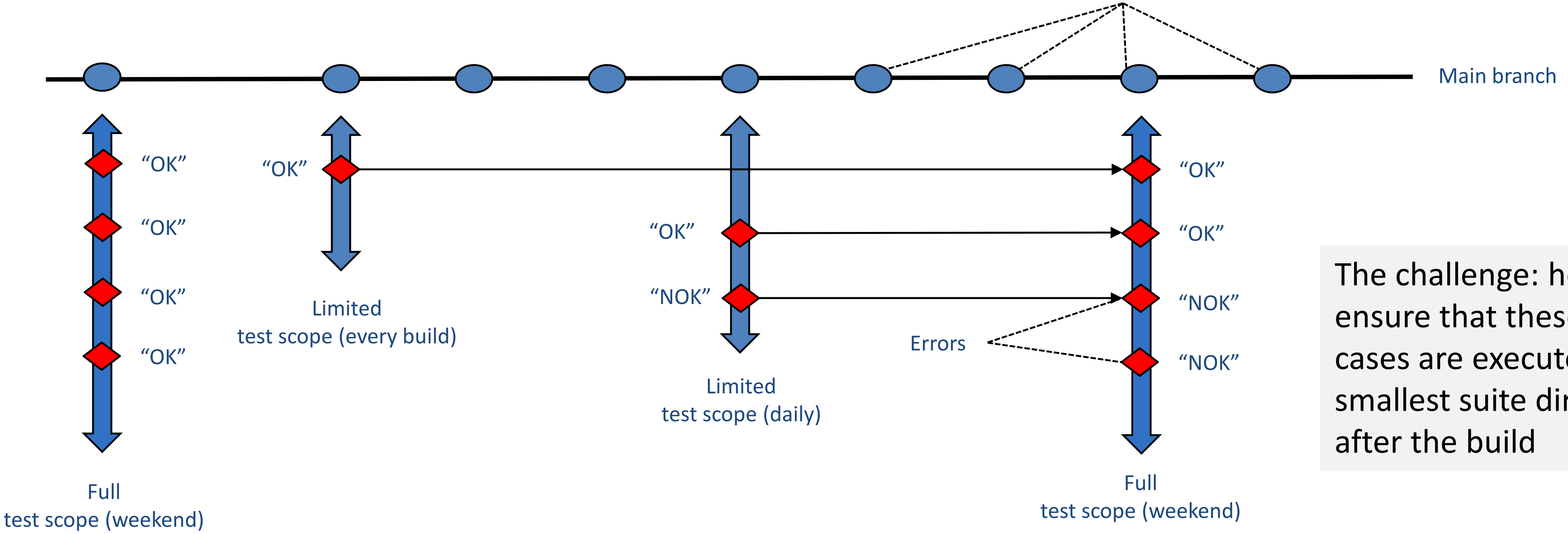
- Developers need feedback about their code (from testing) as soon as possible
  - We should strive to execute the test that have the highest probability of failure as quickly as possible after code commit

Goals for this research

- To reduce the time for testing?
  - Reduce the time for test execution to shorten the feedback loop
  - Reduce the risk of re-introducing new defects when fixing the existing ones

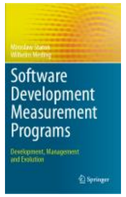
- To increase the rate of fail/executed test cases?
  - Reduce the number of test cases that are executed and do not trigger any failures

Illustration of the context



The challenge: how to ensure that these test cases are executed in the smallest suite directly after the build

Full scope executes "OK" → Build regression executes "OK" → Daily scope executes "OK" → Full scope does not execute "OK" for all test cases



Problem formulation:

- How to predict which test case would fail for a given line of code?
- How can we predict whether a given test case will fail/pass for a given line of code?
- How do we optimize the “limited test scope” for <u>each</u> build, so that no unknown errors are found when we run “full test scope”?

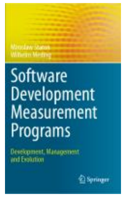
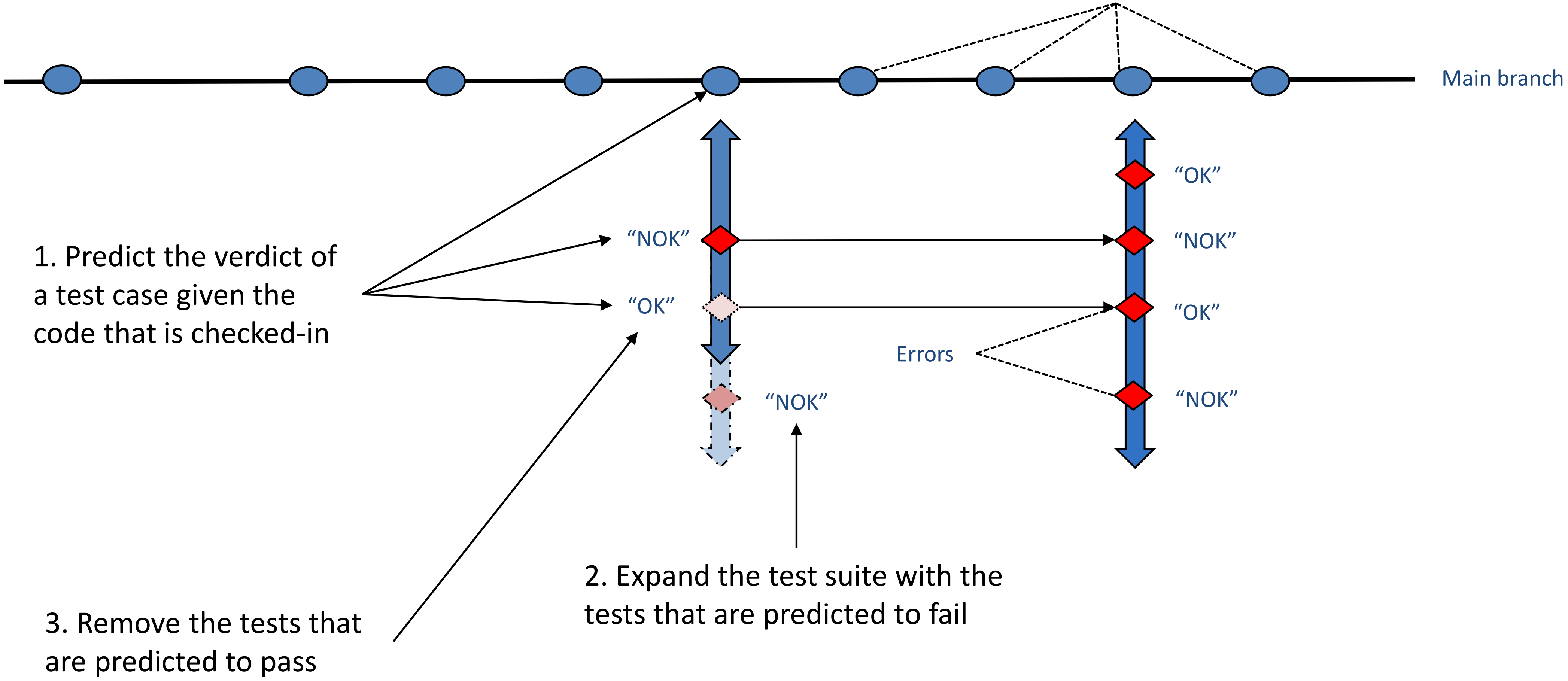
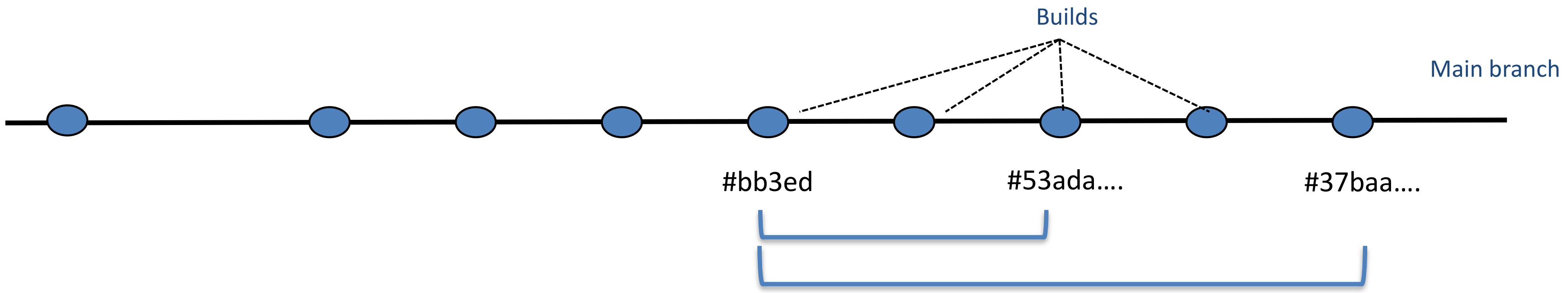


Illustration of the solution



Code Churn



The amount of changes made to software between two points in time is referred to as code churn.

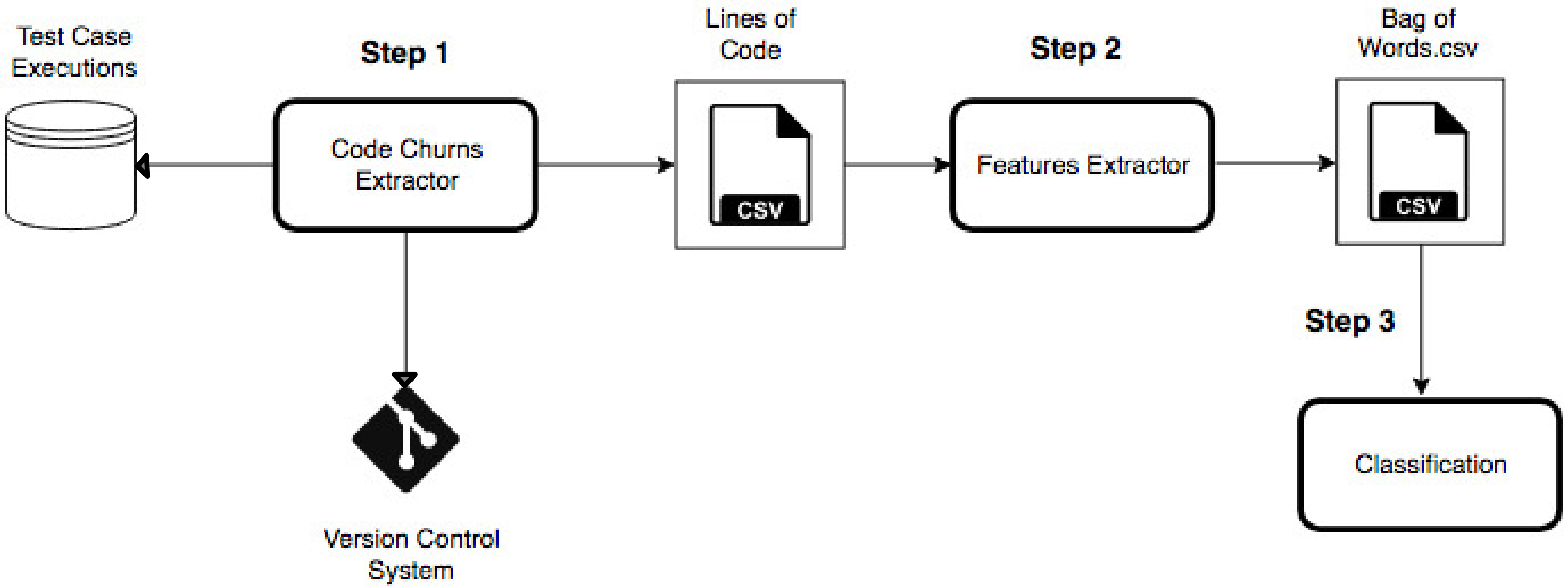
```
//pointer declaration.
Int *p;
….
 int age[100]
 …..
char vowels[][5] = { {'A', 'E', 'I', 'O', 'U'}, {'a', 'e', 'i', 'o', 'u'} };
```

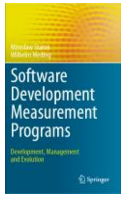
```
//pointer declaration.
Int *p;
….
 int age[100]
 …..
char vowels[][5] = { {'A', 'E', 'I', 'O', 'U'}, {'a', 'e', 'i', 'o', 'u'} };

//array declarations
int person[100]

person[0]= p
…………
………….
```

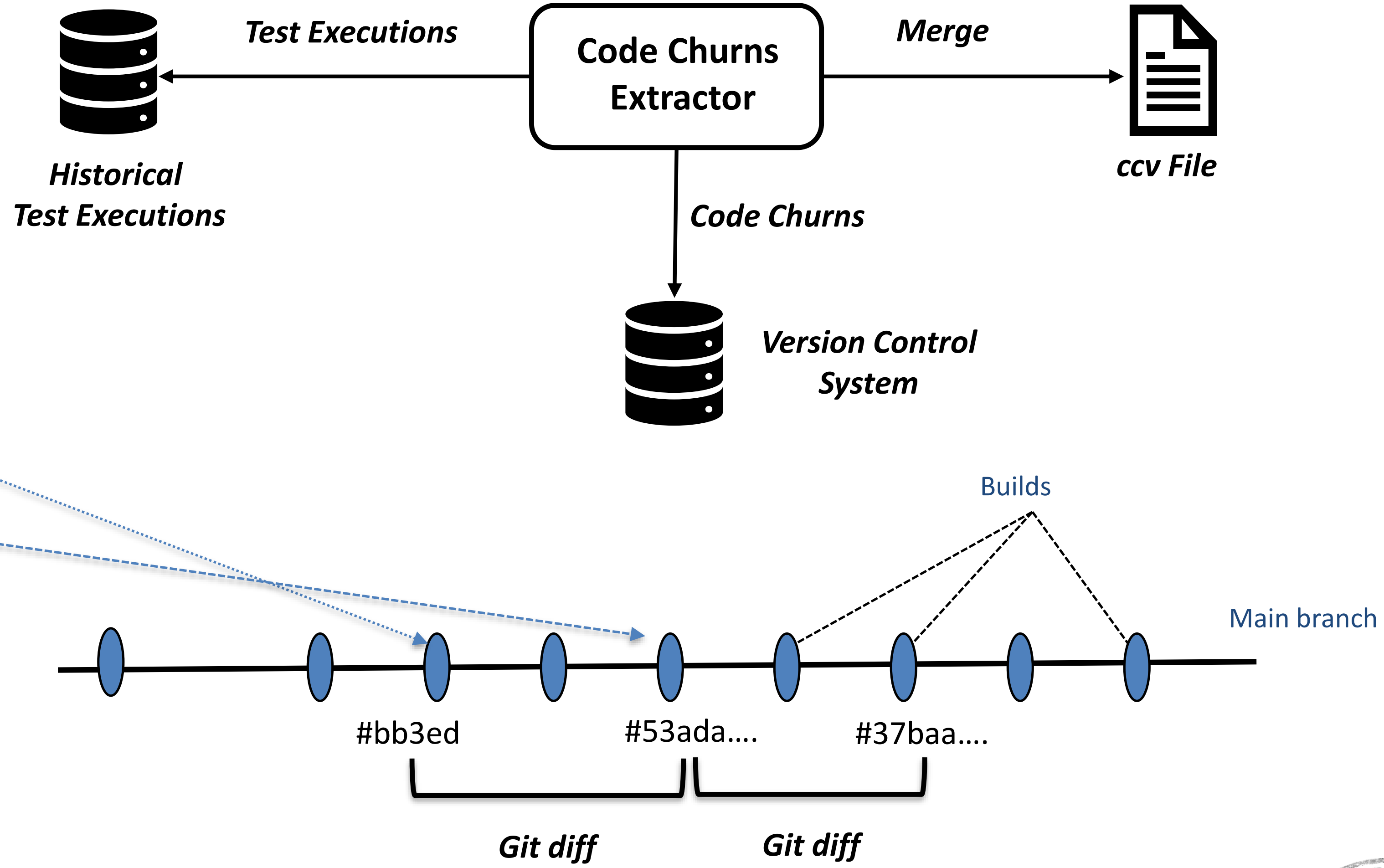
Method using Bag of Words for Test Selection (MeBoTS)



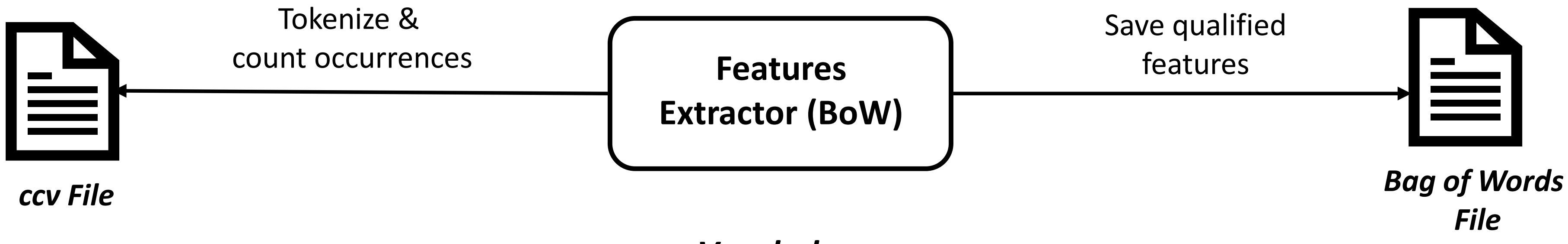


Step 1: (Data Extraction)

Baseline	Test Case Name	Verdict
#33bda….	ST-case 22	Failed
#bb3ed…	ST-case 22	Failed
#53ada…	FT-case 22	Passed
#37baa….	FT-case F2	Failed
#37baa….	FT-case F2	Failed



Step 2: (Features Extraction)



Vocabulary

if	for	(	;	//	[	]	*	else	..	..

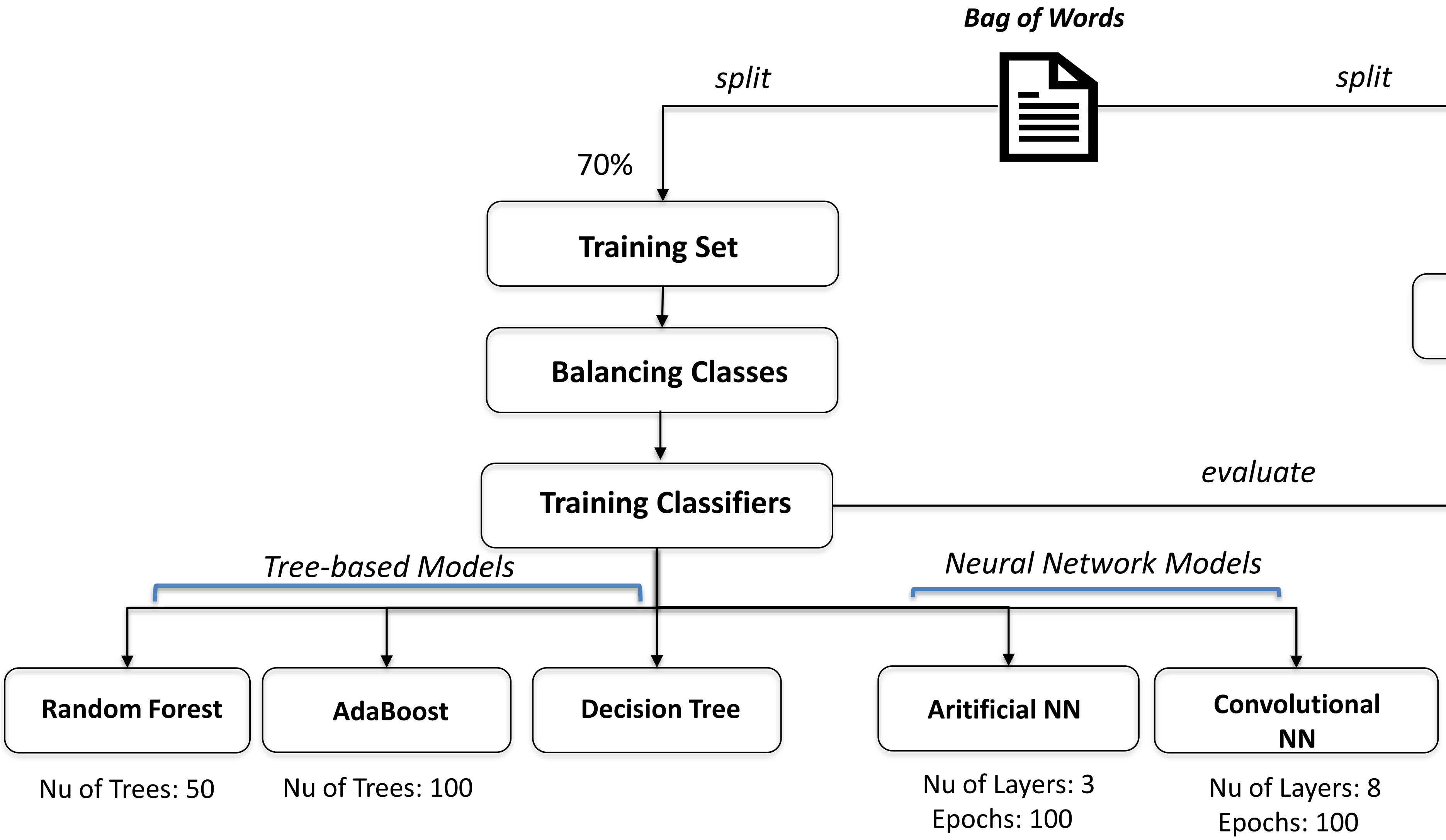
//pointer declaration. ***Int *p;*** …. ***int age[100]*** ….. char vowels[][5] = { {'A', 'E', 'I', 'O', 'U'}, {'a', 'e', 'i', 'o', 'u'} };	*Pass* *Pass* *Pass* *Fail* *Fail* *Fail* *Pass* *Pass* …… …..

csv file

Line #	//	a	*	[	….	Class (0=Fail)
1	1	2	0	0	….	**1**
2	0	1	1	0	….	1
….	….	…	…	…	…	…

Bag of Words csv file

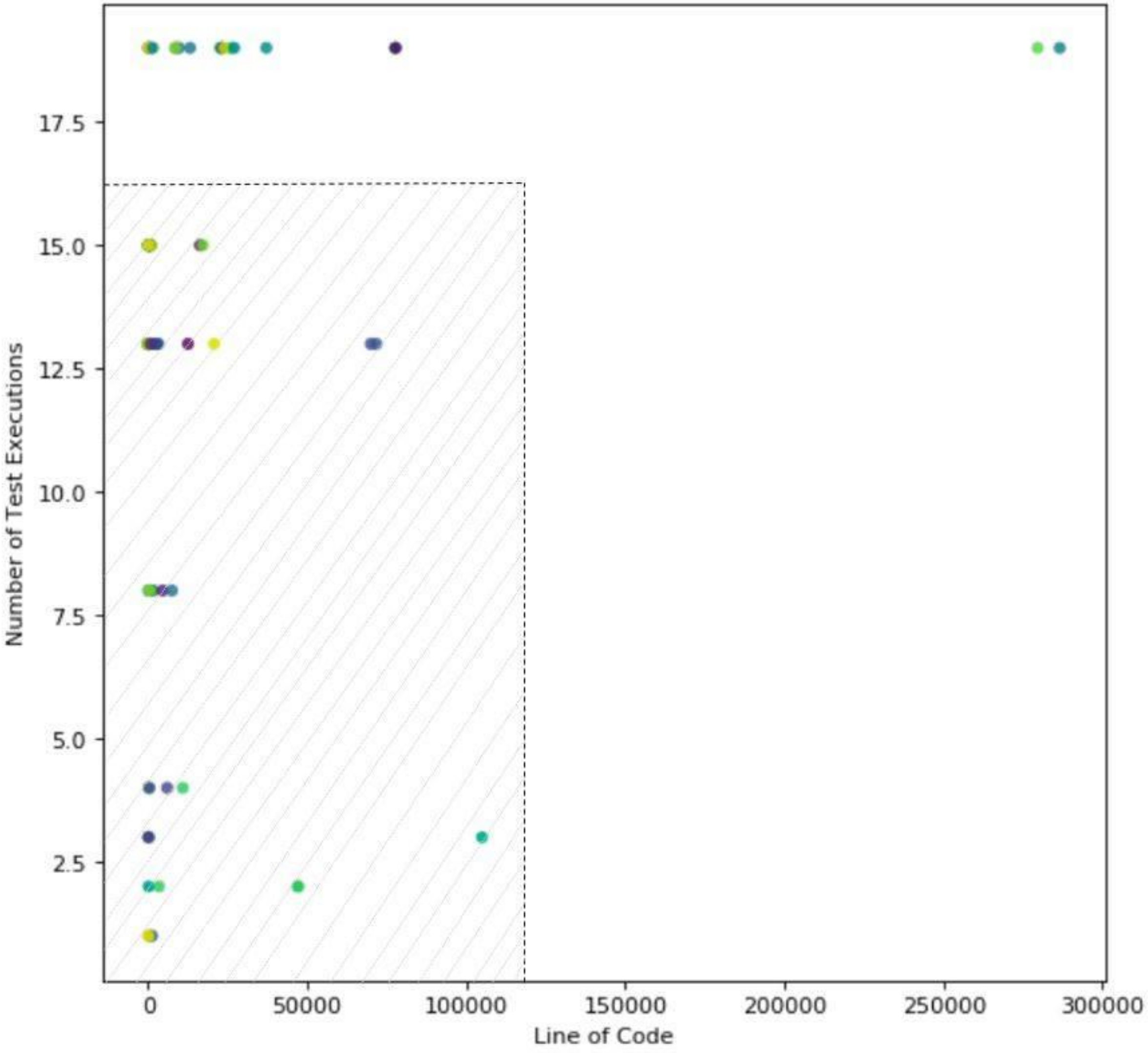
Step 3: (Classification)



Evaluation – Case and Dataset

- Company: Software Telecommunication in Sweden
- Dataset: 12 test cases, 82 executions

- Original Dataset:
  - Mix of small and large churns
  - 1.4m lines of code, 500 features
- Curated Dataset:
  - <120k lines of code per churn
  - 290k lines of code, 500 features



Evaluation - Metrics

- Precision: how many test cases identified as passing will pass?

$$precision = \frac{|TruePositive|}{|TruePositive| + |FalsePositive|}$$

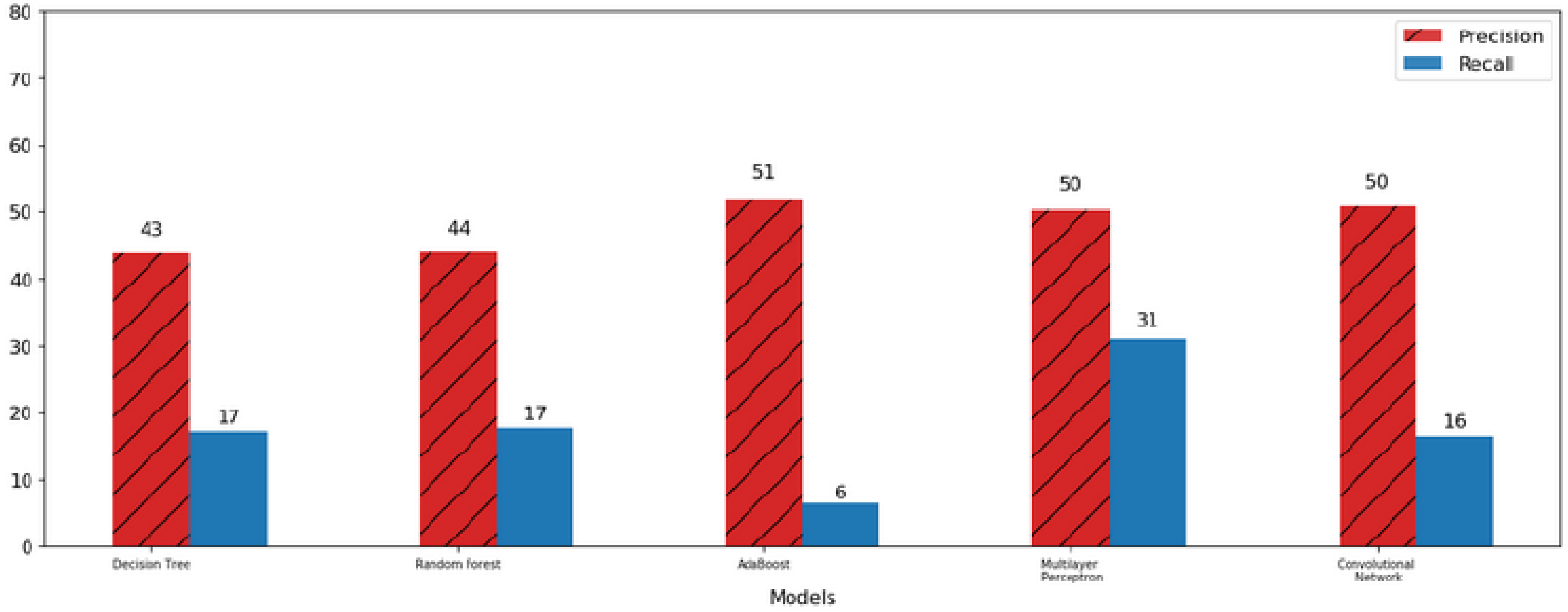
- Recall: How many test cases passing, will be identified as such?

$$recall = \frac{|TruePositive|}{|TruePositive| + |FalseNegative|}$$

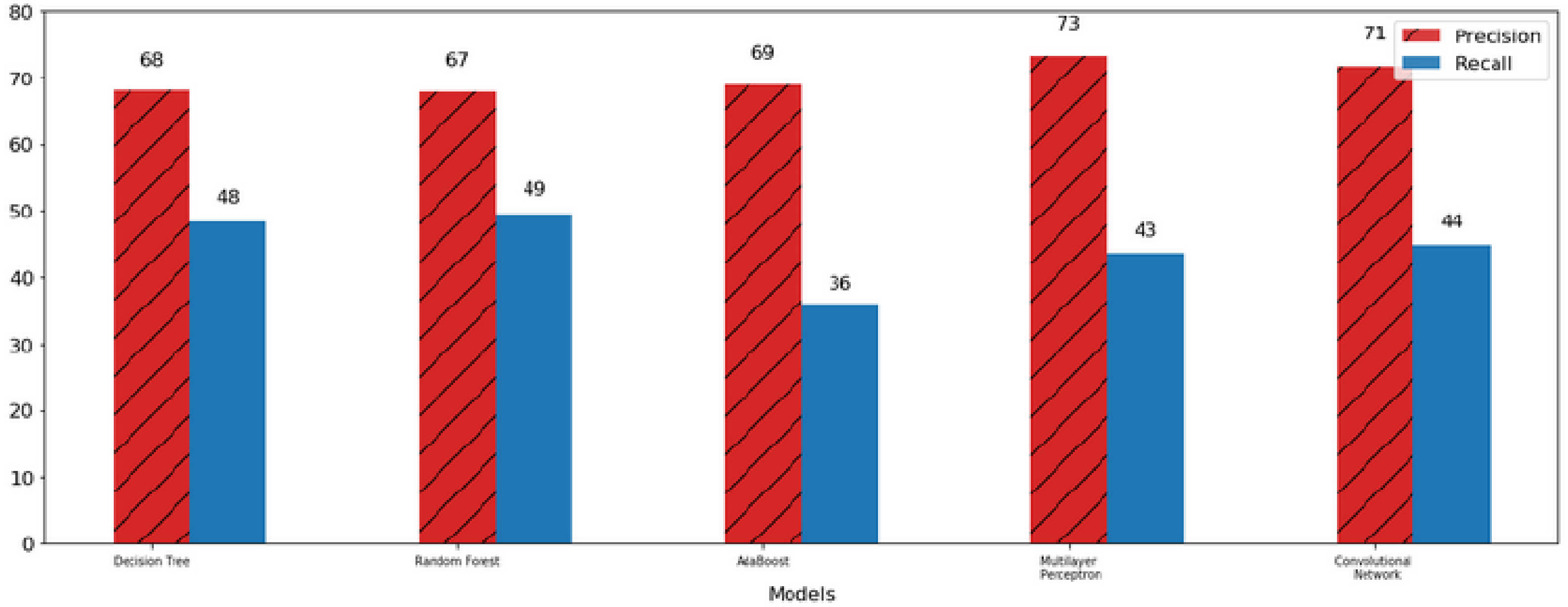
- Goal:
  - High recall to identify many test cases that need no execution
  - High precision to be sure about them

Evaluation - Results

- Before data curation



- After data curation



Result:

- Medium recall: we already identify many test cases that need no execution
- High precision: we are sure about them in >7 of 10 cases

Threats to validity and mitigation

- Small sample size of test executions (7 test cases).
- Test failures may be caused by an environment upgrade or defect in the test scripts.
- Non-deterministic behavior of test cases (flaky tests)
- Different architecture and configuration of the networks’ hyperparameters may result in higher prediction performance.

Conclusion and future work

- More data to evaluate the effectiveness of MeBoTS in practice.
- The prediction performance showed a precision rate of 73% and a medium recall.
- Using the method with small code churns showed an overall improvement in precision and recall.
- Evaluate other textual analysis techniques for better prediction.
- Evaluate the method on different software systems and contexts.
- Evaluate the trained model on code changes from outside the extracted sample.
- Measure the required time to retrain the model for better accuracy.