Test Smells for Flaky Test Prediction

# 1 Pranay Reddy Juturu

2 [pjuturu@stevens.edu](mailto:pjuturu@stevens.edu)

3 Stevens Institute of Technology

4 Hoboken, New Jersey, USA

5

# Ashay Pable

[apable@stevens.edu](mailto:apable@stevens.edu) Stevens Institute of Technology

### Hoboken, New Jersey, USA

Diya Sanghvi 61

[dsanghv1@stevens.edu](mailto:dsanghv1@stevens.edu) 62

Stevens Institute of Technology 63

Hoboken, New Jersey, USA 64

65

## 6 ABSTRACT

7 xxx

8

## 9 CCS CONCEPTS

10

#### 11 • Software Engineering

12

13 **KEYWORDS**

→ **AI for SE**;

team members can view it and can get to know the progress. 66

In large software development, there could be several different 67

kinds of changes that are committed, and sometimes viewing 68

all the changes by all the members can be time-consuming. 69

Hence, we decided to make a feature that tells us what kind 70

of changes are committed. 71

Suppose during the development of a mobile app one devel- 72

oper solves some bugs in the software and commits it. Then 73

14 software quality, mining software repositories

15

16

## 17 1 INTRODUCTION

18 Regression testing is a crucial step in software development, as

19 it helps to ensure that software is delivered continuously with

20 quality and minimal failures after changes to the production

21 code. During this phase, developers rely on the test results to

22 determine whether a program has a bug resulting from recent

23 changes. However, the presence of flaky tests can make this

24 evaluation unreliable. Flaky tests are a type of test with an

25 intermittent behavior that alternates between passing and fail-

26 ing when executed in the same codebase, without any changes.

27 This non-deterministic behavior frustrates developers, as it

28 makes it challenging to identify and fix the root cause of the

29 problem. Additionally, flaky tests are difficult to debug and

30 can cause delays in the release cycles, halting the development

31 process. Flaky tests can be a significant challenge in software

32 development and identifying them is essential for ensuring

33 the reliability and accuracy of test results. Dynamic and static

34 approaches can be used to identify flaky tests, with each ap-

35 proach having its advantages and disadvantages. Dynamic

36 approaches involve re-executing test cases a fixed number of

37 times, which can be expensive and error-prone. It can also

38 be difficult to determine how many executions are enough to

39 identify flakiness accurately. Static approaches, on the other

40 hand, do not require code re-execution and rely on machine

41 learning methods to predict flakiness likelihood based on vari-

|  |  |  |
| --- | --- | --- |
| 42 ous features obtained from the code. Recently, an alternative  43 approach for predicting flaky tests has been proposed based | **2 STUDY DESIGN** | 102  103 |
| 44 on identifying test smells. Test smells are associated with po- | *[Instructor: I expect to see more details explaining the study* | 104 |
| 45 tential design problems in the test code, and their presence | *design as indicated in the phase 2 project description, I miss* | 105 |
| 46 may impact software quality and lead to test flakiness. The | *details about model tuning, and cross-validation, if any. No* | 106 |
| 47 alternative approach uses a set of predictors composed only | *details about data analysis (explore, analyze, synthesize)]* The | 107 |
| 48 of metrics collected statically, such as the size of the test case, | objective of our study is to determine if test smells can be | 108 |
| 49 the number of smells in the test code, and binary features | used to predict the existence of flaky tests. After studying the | 109 |
| 50 related to the presence or absence of 19 test smells. The study | data we decided to drop empty columns as well as rows that | 110 |
| 51 found that this approach had better performance than the | contain some empty entries for categorical columns because | 111 |
| 52 vocabulary-based model for cross-project prediction, achiev- | these empty entries do not provide any information to the | 112 |
| 53 ing an F-measure of 0.83 with Random Forest. | model. To increase the dataset’s quality and guarantee that | 113 |
| 54 GitHub is widely used for software development and version | we are only using pertinent data to train our models, empty | 114 |
| 55 control. During development as the changes are made to the | columns and rows should be removed. The columns ’Author | 115 |
| 56 software, they are committed to the repository so that the | Email’, ’Author Name’, ’Committer Email’, ’Committer Name’, | 116 |
| 57 | 1 | 117 |
| 58 |  | 118 |
| 59 |  | 119 |

the commit should show a label with the bug written and the same goes for refactoring and adding features. In this way, without going in-depth, everyone will come to know about all the changes made.

The commits will be classified on the basis of how the code changes are reported using appropriate solutions.

* Static approaches do not need the code to be executed again. Models built using static features have many advantages and are less costly.
* Pinto et al. built a set of predictors considering that some patterns within the test code may be employed to identify flaky tests automatically.
* The authors came to the conclusion that the vocabulary- based strategy performs poorly when used across projects because it is context-sensitive and prone to overfitting.
* Considering this result, an alternative approach for flaky test prediction based on test smells is used. Test smells are associated with potential design problems in the test code.
* Test smells are a deviation from how tests should be created, arranged, and interacted with one another. That deviation can indicate issues with test design and negatively impact test performance.
* An open-source test smell detection tool, tsDetect is used. For each test case, this tool requires the identifi- cation of the corresponding production code to detect the test smells.

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

’MCSoRm20m22i,tMMayes2s3–a2g4e, ’2,0’2C2,oPmiNsmbuitrgSh,HPAA,’,U’SFAilepath’, and ’Line’ are removed. These columns either have no bearing on our goal or include categorical items with blank. Once this was done we decided to drop columns with zero relevance to the target column Klass which is Binary making it a binary classification. The columns that have zero relevance to the target columns are ’App’, ’Build time in minutes’, ’TimeStamp’, and ’Version’. We have used random forest, decision trees, Naive Bayes, KNN, LDA, and Logistic Regression models. Grid search with cross- validation is used to adjust the hyperparameters for each model and the model with the best validation set performance is chosen. The random forest provides the best results out of the tested models. We use k-fold cross-validation, where we divide the data into k subsets and use each subset as a validation set in turn, to guarantee that our results are not overfitting the training data. This procedure is repeated k times, and the average performance over all iterations is reported. There is scope for improvement with the probable introduction of an ANN. The evaluation metrics we used are Accuracy, Precision, Recall, F1 score, Matthews correlation coefficient, and Area Under the Curve. The performance of each of the models is expressed and compared in the further sections of this paper. From studying the data we also concluded that the dataset only identifies if a test is flaky with a binary representation without the inclusion of any magnitude to provide the extent of flakiness, which might produce some limitations and could be scope for future experimentation. For embedding of text inputs we are using CountVectorizer and Tfidf Vectorizer from the sklearn.feature\_extraction.text class.

Our dataset consists of data coming from a variety of fa- mous GitHub projects and extracting information, like commit messages, analyzing tests, LOC, etc.

Our workflow will consist of information extraction from Github repositories, specifically the commit messages, and analyzing tests to identify test smells. This information is then used to feed data into our classifier once the text from the commit messages is tokenized and embedded. This is then fed into the trained model to classify the data point into flaky or not flaky.

## EXPERIMENTAL RESULTS

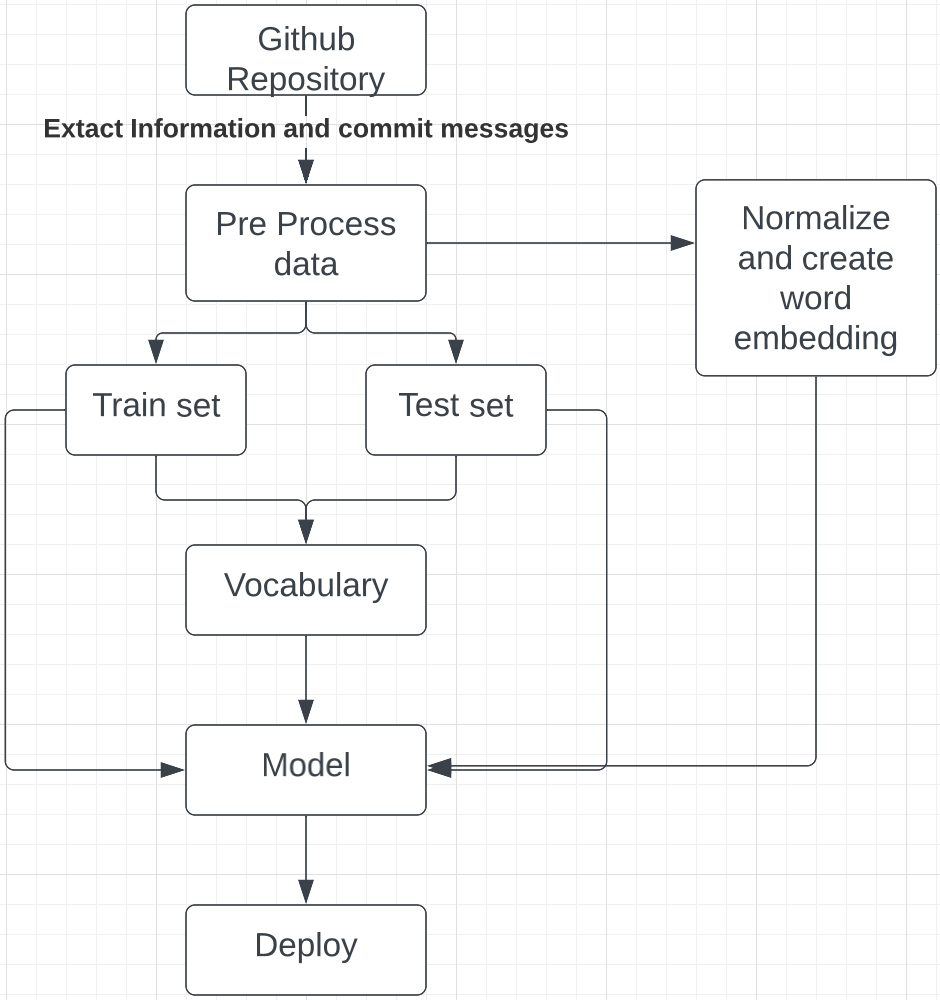
#### RQ1 – How accurately can we predict test flakiness based on test smells in the test cases?

By first training and then evaluating the classifiers, the predic- tion model was developed. Every classifier had a reasonable performance archive except the Naive Bayes classifier with an accuracy of 65%. Random Forest classifier had higher accu- racy and precision of 83%. The collected results demonstrate that test smell-based models, with precision values ranging from 75% to 83%, perform reasonably well in predicting test flakiness. The trained classifiers were tested using the flaky tests included in the idFlakies dataset to confirm the model’s performance in the cross-project environment.

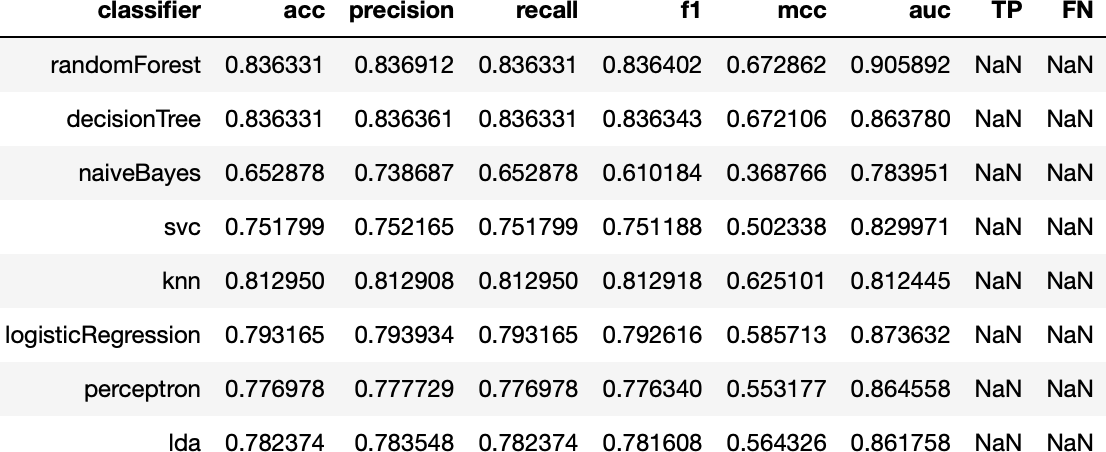
In the intra-project scenario, the performance of all the classifiers is dropped. Logistic regression attained the highest score. It accurately identified 26 out of 9 flaky tests. In an inter-project scenario, the classifier’s performance declined

2

Diya Sanghvi

**Figure 1: Flowchart** *[Instructor: Add details methodology and descriptive caption. What did you do with data preprocessing? I miss this discussion in the text.]*

Pranay Reddy Juturu, Ashay Pable, and



#### Figure 2: Training result parameters

more sharply. With recall values ranging from 48% to 55%, the classifiers do not differ significantly from one another, with Naive-Bayes reaching a value of 14% by accurately identifying 17 out of 103 flaky tests. The results collected demonstrate that the smells can be utilized to predict flakiness. But, in the inter-project scenario, performance suffers significantly. The findings demonstrate that the performance of the smell-based models is equivalent to, and occasionally even superior to vocabulary model. The classifier’s performance ranges from 11% to 55%. This led to the conclusion that smells are reliable indicators of flakiness.

#### RQ2 – Which attributes are the most strongly associ- ated with test flakiness prediction?

To identify associations between attributes and flakiness, we used sklearn.feature\_selection.mutual\_info\_classify method of scikit-learn that allows us to select features and in the future experiment by eliminating the least relevant features to opti- mize the model. The function calculates Mutual Information, which is the measure of the mutual dependence between two random variables. We use MI over Correlation between each attribute as it is more versatile and can capture non-linear re- lationships, while correlation is limited to linear relationships. **The equation for MI between a feature X and a target variable y is:**

*MI* (*X* , *y*) = . *p*(*x*, *y*) ∗ *log*(*p*(*x*, *y*)/(*p*(*x*) ∗ *p*(*y*)))

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

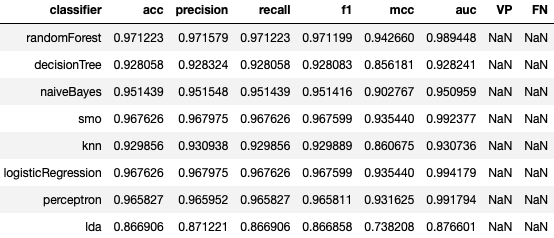
238

239

Test

Smells for Flaky Test Prediction

MSR 2022, May 23–24, 2022, PiNsburgh, PA,



USA

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

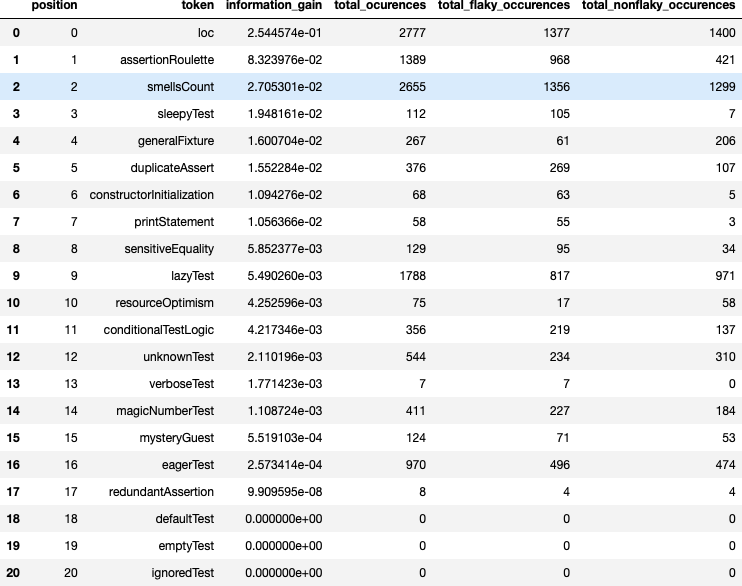
296

297

298

299

#### Figure 3: Vocabulary-based approach



**Figure 4: Features**

While PCA Is a great Dimensionality tool, its usage as a fea- ture selection is controversial due to the information loss it causes, as well as the reduced flexibility to manually select attributes to eliminate. We identified that the most relevant feature in the dataset was loc (lines of Code) followed by as- sertion roulette (test smell), while the least relevant feature turned out to be redundant assertion (which is a test smell). Whereas, IgnoredTest, emptyTest, and defaultTest have no relation with an absolute zero score, thus indicating the need to drop the three columns. The dataset only identifies if a test is flaky with a binary representation without the inclusion of any magnitude to provide the extent of flakiness, which might produce some limitations and could be scope for future experimentation.

3

#### Figure 5: Vocabulary-based approach

**RQ3 – How does the test smell-based approach com- pare with the existing vocabulary-based approach?**

In the vocabulary-based approach, the values of VP (True Pos- itive) and FN (False Negative) for the classification metrics are NaN (Not a Number). This is due to the dataset used for evalu- ation in the vocabulary-based approach only containing flaky tests and excluding non-flaky tests. As a result, this dataset lacks True Negatives (TN) and False Negatives (FN), making it impossible to calculate VP and FN.

Precision, recall, and F1-score for the vocabulary-based technique cannot be determined in the absence of TN and FN data. The accuracy and AUC numbers for this technique are the only ones that the authors have reported. We trained the classifiers with the training and testing dataset using the vocabulary-based approach. The vocabulary-based strategy performs better than the smell-based approach: the best F1 score for vocabulary-based models is 97% (Random Forest), while the score for the smell-based approach is 83%. (Random Forest). The disparity is greater when MCC is analyzed. The best outcome for the smell-based technique was 0.66, and the best result for the vocabulary-based approach was 0.94. This score takes into consideration true and false positives, as well as negatives. The cross-project validation results, however, demonstrate that the test smell-based strategy yields superior outcomes. The test smell-based strategy yielded 74% of recall (LR) in the intra-project context, while the vocabulary-based approach only managed to reach 57%. (KNN).

Using the training and testing datasets, the performance of the vocabulary-based models is superior to that of the test smell-based models. Yet, the smell-based approach achieves noticeably higher outcomes in the intra-project and inter- project contexts in the cross-project validation scenario.

## THREATS TO VALIDITY

**Construct Validity:** The degree to which a concept is oper- ationalized (i.e., how it is measured) accurately reflects the intended construct is known as its construct validity. The re- lation between test smells and flaky test is the construct that interest this study’s researchers. To operationalize this con- cept, a variety of test smells, including the assertion roulette and conditional test logic were considered. There is a potential threat to the identification of the flaky tests.

The most widely used metrics in the machine learning (ML) community were adopted to evaluate the classifiers in order to reduce this threat, which can aid in improving the study’s generalizability and reliability. However, the authors utilized a

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

pMrSoRg2r0a2m2, Mcaalyle23d–t2s4D, 2e02te2,cPtiNtosbfiurngdh, tPeAs,tUsSmA ells in the code during the pre-processing stage of the test code. The production class, a vital component of the codebase that the tests are testing, has occasionally been missed by tsDetect. As a result, there were instances where test smells could not be extracted from the code, which could jeopardize the study’s findings.

**Internal Validity:** The degree to which a research study ac- curately ascertains the link between the independent variable and the dependent variable is referred to as internal validity. The existence of confounding variables, which are variables that can impact the relationship between the independent and dependent variables, is one factor that could endanger inter- nal validity. Confounding variables in this study may include things like the size and complexity of the codebase, the level of experience of the developers, and the particular programming language employed. The authors employed statistical tech- niques like logistic regression and decision trees to account for the effects of these confounding variables and identify the association between test smells and flaky tests in order to ad- dress this possible danger.

**External Validity:** External validity is the extent to which a research study’s findings can be extrapolated to populations, environments, and situations unrelated to the study’s partic- ular context. Four open-source projects were utilized by the authors to gather data for their study, although other software projects in different domains or with various characteristics might not be comparable to these projects. The results may not apply to projects created in other programming languages, for instance, since all of the projects utilized in the study were written in Java. The study’s projects were small to medium- sized, and the authors also pointed out that they might not accurately represent the features of larger software projects because of their size. The authors emphasized the limits of their study and suggested that future research should inves- tigate whether or not their findings apply to other software projects and domains in order to address this possible threat.

**Conclusion Validity:** The degree to which the inferences made from the data accurately reflect the underlying relation- ships between the variables under investigation is referred to as conclusion validity. The authors of the research thoroughly analyzed their data and assessed how well their classifier mod- els performed using the relevant statistical methods. They also talked about some of the study’s possible drawbacks, namely the use of a single dataset and the scant number of projects examined. They did not, however, point out any particular problems or worries regarding the inferences made from the data.

## CONCLUSION

According to the study, a number of test smells, such as asser- tion roulette and conditional test logic, were strongly linked to flaky tests. Based on these smells, the classifiers created using machine learning approaches were highly accurate in predicting flaky tests. The accuracy of the test smell detection

4

tool and the smallPsraizneayaRnedddsycJoutpueru,oAfshthayePdaabltea, saentd DuityializSaendghivni the study are two major risks to the validity of the study, the authors point out. The authors contend that additional study is required to both explore additional variables that can affect test flakiness in software testing and to confirm the efficacy of test smells as predictors of flaky tests. Overall, the study offers insightful information about the use of test smells as a viable method for enhancing the accuracy and effectiveness of software testing.

## FUTURE WORK

A promising solution to the problem of finding and anticipat- ing flaky tests in software testing is presented by the authors. More study in this area is still needed. The current study fo- cused on a particular set of test smells, but there might be additional clues that might be used to spot and foretell prob- lematic testing. To increase the accuracy of shaky test predic- tion models, future studies could include other test smells or other elements, such as code complexity or ambient elements. Despite the study’s encouraging predictions for flaky tests, it is crucial to comprehend how employing test smells impacts the entire software testing process. Future studies could look into how using test smells affects testing effectiveness, efficiency, and overall software product quality. The tsDetect tool was used in the investigation to find test smells in the codebase. Nonetheless, this tool’s accuracy and dependability could yet be enhanced. Future studies could concentrate on creating more advanced tools or enhancing currently available tech- nologies more correctly and effectively detect test scents. The study does not address the fundamental reasons for flakiness, even if it offers a means to recognize and anticipate flaky tests. Future studies could look at the underlying factors that con- tribute to flaky testing and devise methods to stop them from happening in the first place.

## REFERENCES

[1] B. H. P. Camara, M. A. G. Silva, A. T. Endo, and S. R. Vergilio. 2021. On the use of test smells for prediction of flaky tests. In *Brazilian Symposium on Systematic and Automated Software Testing*. 46–54. [https://doi.org/10.1145/](https://doi.org/10.1145/3482909.3482916)

[3482909.3482916](https://doi.org/10.1145/3482909.3482916) arXiv:2108.11781 [cs].

[[1]](#_bookmark0)

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479