Analisi What-If dell'Impatto del Refactoring sulla Propensione ai Bug

Studio empirico su Apache Bookkeeper e Apache Storm

Sofia Tosti
Ingegneria del Software 2
Università degli Studi di Roma Tor Vergata
A.A. 2024/2025

Il Problema

Il refactoring può davvero prevenire i bug?

Manutenibilità del Software

Il codice sorgente subisce modifiche frequenti e la presenza di difetti a livello di metodo compromette la stabilità del sistema

Carenza di Evidenze

Mancano studi empirici sull'impatto della riduzione dei code smells sulla bugginess a livello di singolo metodo

Necessità di Metriche

Serve un approccio data-driven per guidare le decisioni di refactoring e ridurre il debito tecnico

Domande di Ricerca

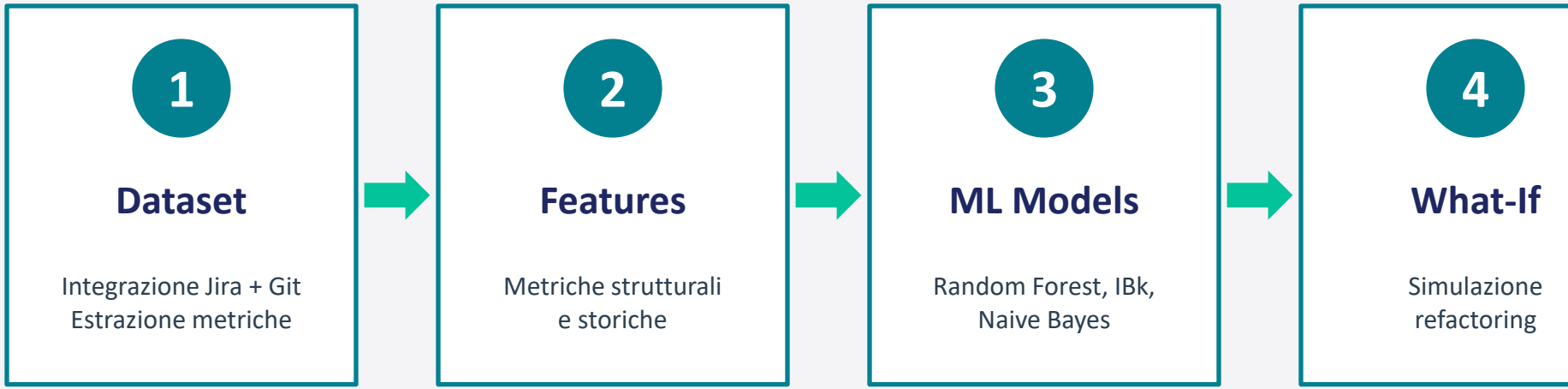
RQ1

Quale classificatore è in grado di prevedere con maggiore accuratezza la presenza di bug a livello di metodo?

RQ2

Qual è l'impatto potenziale di un intervento di refactoring mirato sulla riduzione dei metodi buggy stimati?

Metodologia



Metriche Analizzate

Metriche Strutturali

Cosa possiamo modificare

- Lines of Code (LOC)
- Complessità Ciclomatica
- Complessità Cognitiva
- Profondità Annidamento
- Numero di Parametri
- Code Smells (PMD)

Metriche Storiche

Storia evolutiva del codice

- Numero di Revisioni
- Numero di Autori
- Code Churn
  - Totale
  - Massimo
  - Medio

Etichettatura

Integrazione di Jira+Git

- Bug risolti e chiusi da Jira
- Commit correttivi da Git
- Collegamento via ticket ID

Determiniamo:

- Fixed Version (FV)
- Opening Version (OV)
- Injected Version (IV) **?**

Tecnica Proportion

Per ticket con IV mancante:

P=(FV-IV)/(FV-OV)

- Ordinamento per Resolution Date
- Analisi incrementale
- P stimato come media dei valori già osservati

Impatto delle metriche

Apache Bookkeeper

Sensibile alle metriche strutturali

Top 3 Correlazioni:

1. Revisions (storica)
2. Authors (storica)
3. LOC (azionabile)

→ Metodi lunghi e complessi accumulano difetti

Apache Storm

Dominato dalla storia evolutiva

Top 3 Correlazioni:

1. Authors (storica)
2. Revisions (storica)
6. NestingDepth (azionabile)

→ Bug da interazioni complesse e modifiche incrementali

Validazione e Preparazione dei Dati

10x10 Cross-Validation

- Dataset diviso in 10 fold
- 9 fold per training, 1 per test
- Ripetuto 10 volte

Beneficio: valutazione robusta e riduzione overfitting

Bilanciamento delle Classi

Problema: classi sbilanciate (più metodi non-buggy che buggy)

Bookkeeper → SMOTE (+3,144)
Storm → Down-sampling (20K)

Feature Selection: Information Gain

Misura quanto ogni feature riduce l'incertezza sulla classe

Soglia: IG ≥0,01
Bookkeeper : 11→9 | Storm: 11 → 6

RQ1: Risultati Predittivi

Random Forest è il classificatore più efficace

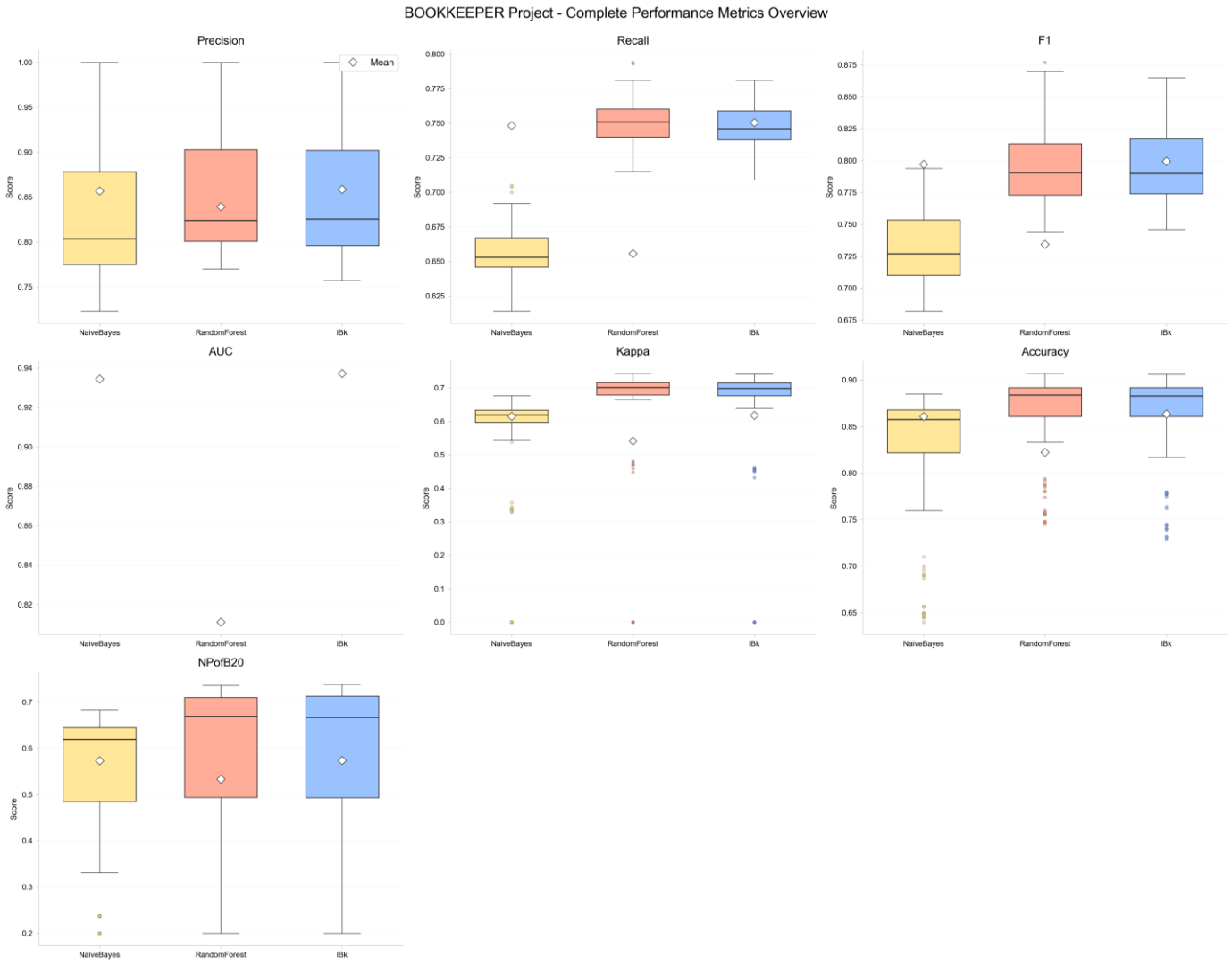
Random Forest

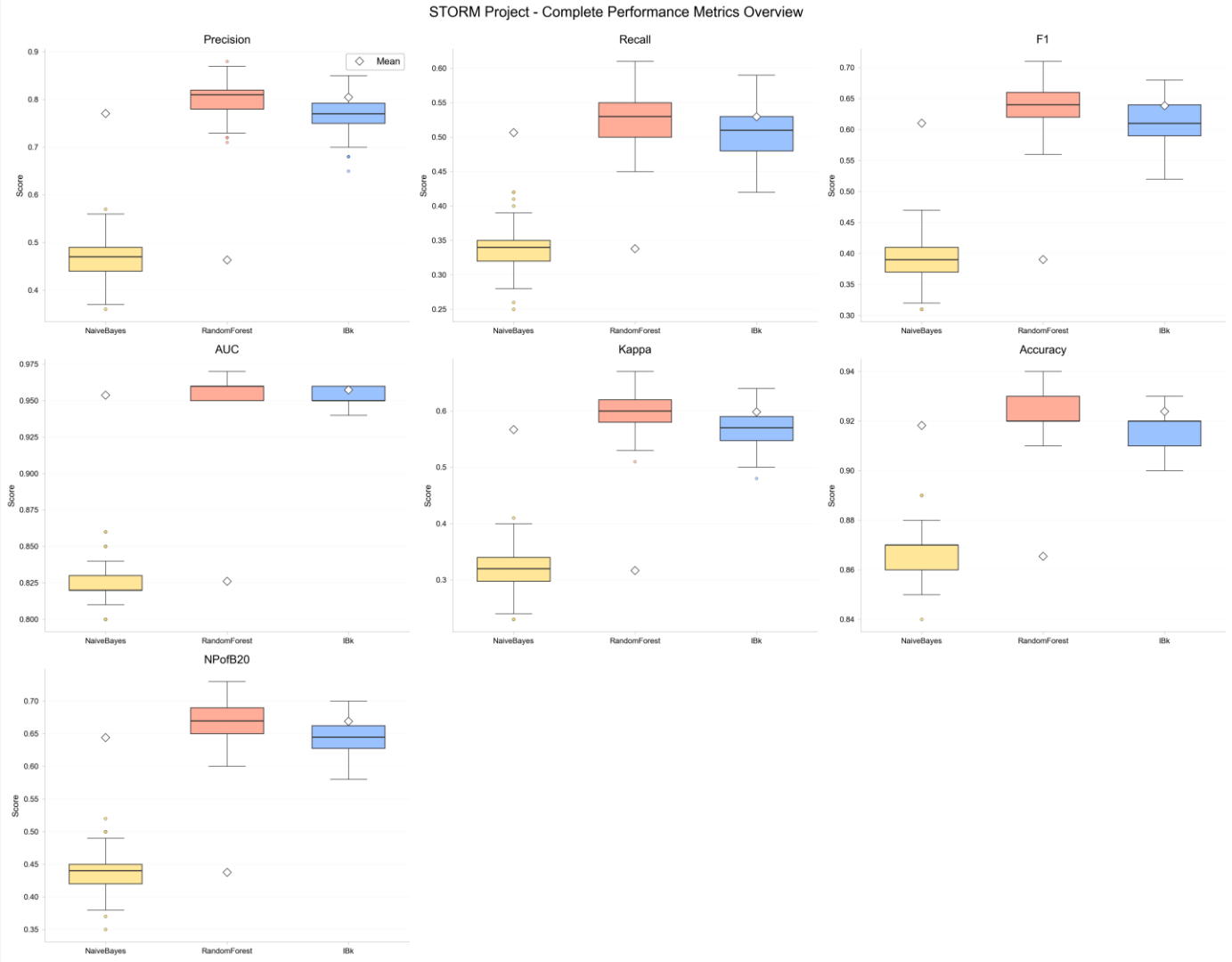
Bookkeeper: Score 0.635
Storm: Score 0.579

Migliore bilanciamento tra precisione e recall

IBk: Performance comparabile, secondo posto

Naive Bayes: Recall significativamente inferiore


BOOKKEEPER Project - Complete Performance Metrics Overview
Precision
Mean
Recall
F1
AUC
Kappa
Accuracy
NPofB20
Score
NaiveBayes
RandomForest
IBk


STORM Project - Complete Performance Metrics Overview
Precision
Mean
Score
NaiveBayes
RandomForest
IBk
Recall
F1
AUC
Kappa
Accuracy
NPofB20

Refactoring: Apache Bookkeeper

Metodo: processPacket(ByteBuffer, Cnxn) in BookieServer.java

✕ Prima

LOC: 174
Complessità Cognitiva: 73
Nesting Depth: 5
Code Smells: 14

Problema: Metodo monolitico che viola il principio di singola responsabilità

✓ Dopo

LOC: 29 (-83%)
Complessità Cognitiva: 4 (-95%)
Nesting Depth: 2 (-60%)
Code Smells: 0 (-100%)

Tecnica: Extract Method
Classificazione: Non Buggy

Refactoring: Apache Storm

Metodo: select(String, List) in JdbcClient.java

✕ Prima

LOC: 54
Nesting Depth: 14
Complessità Cognitiva: 10
Code Smells: 3

Problema: Strutture di controllo eccessivamente annidate

✓ Dopo

LOC: 19 (-65%)
Nesting Depth: 3 (-79%)
Complessità Cognitiva: 6 (-40%)
Code Smells: 0 (-100%)

Tecnica: Extract Method
Intera classe: Non Buggy

RQ2: Analisi What-If

Se avessimo rifattorizzato preventivamente...

Apache Bookkeeper — **188 metodi buggy in meno**

≈ 9.2% di riduzione • Forte impatto delle metriche strutturali

Apache Storm — **61 metodi buggy in meno**

≈ 1.7% di riduzione • Dominanza delle metriche storiche

Conclusioni

1

Il refactoring preventivo funziona

Interventi mirati possono ridurre significativamente i metodi buggy, ma l'efficacia dipende dal tipo di progetto

2

Le metriche storiche dominano

Revisions e Authors sono i predittori più forti, ma le metriche strutturali sono quelle su cui possiamo agire

3

Progetti diversi, strategie diverse

Bookkeeper beneficia di refactoring strutturale, Storm richiede attenzione al processo di sviluppo

4

Random Forest è il vincitore

Migliore bilanciamento tra precisione e recall per la predizione di difetti a livello di metodo

Sviluppi Futuri

- Estendere l'analisi ad altri progetti open source per validare la generalizzabilità dei risultati
- Includere metriche più specifiche come accoppiamento e coesione tra componenti
- Sviluppare strumenti automatici per misurazioni più accurate

4

5

https://github.com/tostfia/ISW2_project.git

https://sonarcloud.io/project/overview?id=tostfia_ISW2_project

Grazie per l'attenzione

Sofia Tosti
sofia.tosti@students.uniroma2.eu
Università di Roma Tor Vergata