Programming assignments evaluation using technical debt

Alessio Mason

6 dicembre 2023

Politecnico di Torino

L’obiettivo della tesi

- Applicazione del concetto di debito tecnico ad esercizi di programmazione svolti da studenti
- Studenti ai primi approcci con la programmazione o, in questo caso, con la programmazione orientata agli oggetti

Contesto

Debito tecnico e analisi statica del codice

- Concetto di debito tecnico solitamente usato nelle grandi aziende
- Metafora introdotta da Ward Cunningham nel 1992
- Le imperfezioni nel codice rappresentano un “debito da saldare”
- “Ogni minuto speso su codice non corretto conta come interesse verso quel debito”
- Imperfezioni possono essere bug veri e propri o code smell: si possono individuare con un’analisi statica del codice

Due prospettive di applicazione

- Punto di vista dello studente
  - Utile per migliorare la propria preparazione e per autovalutarsi
- Punto di vista del docente
  - Utile durante il corso per comprendere gli argomenti meno chiari agli studenti, per poi rivederli
  - Utile in sede di valutazione

L’analisi massiva di progetti di studenti

- Sviluppato uno script che organizza le cartelle, compila il codice ed esegue l’analisi statica con SonarQube
- Analizzato un caso reale: il primo appello del corso di “Programmazione ad oggetti” dello scorso giugno
- Evidenziate le problematiche più diffuse e la loro correlazione e impatto sul voto finale

Problematiche comuni evidenziate

- Gestione dei tipi
- Mancato controllo di casi “pericolosi”
- Confronto di stringhe e tipi Boxed per riferimento e non per valore
- Convenzioni di formattazione del codice non rispettate
- Metodi, attributi o frammenti di codice inutilizzati o commentati

Problematiche comuni evidenziate

- Gestione dei tipi
- Mancato controllo di casi
- Confronto di stringhe e tip
- Convenzioni di formattazi
- Metodi, attributi o framme

Confronto di tipi differenti

```
public class TrainManager {
    private List<String> stopsList;
    private Map<Integer, Stop> stopsMap;
    private Stop lastStop;

    // ...

    public int setLastStop(String stop) {
        this.lastStop = this.stopsMap.get(stop);

        for (String s : this.stopsList) {
            if (s.equals(lastStop)) {
                // ...
            }

            // ...
        }
    }
}
```

Problematiche comuni evidenziate

- Gestione dei tipi
- Mancato controllo di casi “pericolosi”
- Confronto di stringhe e tipi Boxed per rif
- Convenzioni di formattazione del codice
- Metodi, attributi o frammenti di codice ir

Possibile divisione per zero

```
public double getCompleteness() {
    double totSlot = 0;
    double sa = 0;

    for (Schedule s : schedules.values()) {
        totSlot += s.getSlots().size();
        sa += s.getAppointments().size();
    }

    return sa/totSlot;
}
```

Problematiche comuni evidenziate

- Gestione dei tipi
- Mancato controllo di casi “pericolosi”
- Confronto di stringhe e tipi Boxed per riferimento e non per valore
- Convenzioni di formatta
- Metodi, attributi o framm

```
public void addDoctor(String id, String name) {
    for (int i=0; i<doctors.size(); i++) {
        if (doctors.get(i).id == id) {
            // ...
        }
    }

    // ...
}
```

Problematiche comuni evidenziate

- Gestione dei tipi
- Mancato controllo di casi “pericolosi”
- Confronto di stringhe e tipi Boxed per riferimento e non per valore
- Convenzioni di formattazione del codice non rispettate
- Metodi, attributi o frammenti di codice inutilizzati o commentati

Problematiche comuni evidenziate

- Gestione dei tipi
- Mancato controllo di casi “pericolosi”
- Confronto di stringhe e tipi Boxed per riferimento e non per valore
- Convenzioni di formattazione del codice non rispettate
- Metodi, attributi o frammenti di codice inutilizzati o commentati

Qualità del codice e voto finale

- Analisi generale
  - Si è analizzata la correlazione tra numero totale di bug e code smell e voto finale o numero di righe di codice del progetto dello studente
- Analisi per ogni bug o code smell
  - Analizzata la correlazione tra numero di occorrenze e voto finale o numero di linee di codice
  - Analisi con t-test per verificare se c'è una differenza significativa tra la media di chi ha riscontrato tale bug o code smell e quella di chi no

Conclusioni

- L’applicazione del concetto di debito tecnico all’analisi di esercizi di programmazione di studenti è un’idea possibile e utile
- Con un’analisi massiva è possibile verificare quali sono le problematiche più diffuse tra gli studenti
- Possibili sviluppi futuri
  - Introdurre anche gli studenti a questa applicazione del debito tecnico
  - Testare l’idea su corsi differenti e con target differenti
  - Inserire la valutazione della qualità del codice all’interno della valutazione d’esame