HBSniff: A Static Analysis Tool for Java Hibernate Object-Relational Mapping Code Smell Detection

Zijie Huang, Zhiqing Shao*, Guisheng Fan*, Huiqun Yu, Kang Yang, Ziyi Zhou

Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China

Abstract

Code smells are symptoms of sub-optimal software design and implementation choices. General code smell (e.g., coupling and cohesion issues) detection tools were actively developed, but they cannot capture domain-specific problems. In this work, we fill the gap in data persistence and query code quality by proposing HBSniff, i.e., a static analysis tool for detecting 14 code smells and 4 mapping metrics in Java Hibernate Object-Relational Mapping (ORM) codes. HBSniff is tested, documented, and manually validated. It also generates readable and customizable reports for every project. Moreover, it is beneficial to Mining Software Repository (MSR) research requiring large-scale analysis since project compilation is not needed for detection.

Keywords: Code Smell, Object-Relational Mapping, Hibernate, Static Analysis, Object-Oriented Programming

1. Introduction

Code smells (*i.e.,* symptoms of sub-optimal design and implementation choices [1]) are related to determining factors of software quality such as change- and error-proneness [2]. Recently, researchers found that general smells and their detection tools cannot capture more specific problems related

*Corresponding Authors.

Email addresses: `hzj@mail.ecust.edu.cn` (Zijie Huang), `zshao@ecust.edu.cn` (Zhiqing Shao), `gsfan@ecust.edu.cn` (Guisheng Fan), `yhq@ecust.edu.cn` (Huiqun Yu), `15921709583@163.com` (Kang Yang), `zhouziyi@mail.ecust.edu.cn` (Ziyi Zhou)

to domain-specific code [3]. Since then, the quality of these codes (*e.g.*, for data persistence [4]) were attracting more attention from researchers.

To facilitate Object-Oriented Programming (OOP), practitioners use Object-Relational Mapping (ORM) frameworks which bridges database and application by filling the gap of data mapping and persistence [4, 5]. Despite its flexibility and capability, there exist various challenges in practice [5] including the metamorphic class and table inheritance [6, 7], the inconsistency in data structure [8], and the uncontrollable propagation of relational data retrieval [9]. Consequently, ORM usage is regarded as a double-edged sword [4, 5] or even an anti-pattern [10]. However, recent works suggested ORM need not affect performance if used properly [4], and practitioners need more static analysis tool support to help them with development [3]. In response, related works [10, 11] outlined several smells and refactoring strategies to cope with them. In most cases, they either did not provide tools, or the tools were early prototypes and requires project compiling, which is not ideal [12, 13] for large-scale analysis over real-world systems. Thus, we fill the gap by proposing a static analysis tool called HBSniff (HiBernate Sniffer) for ORM code smell detection. Similar to related works [3, 9, 10, 11, 14, 15, 16], we use the trending Java Hibernate[1] framework as the context of our implementation.

The highlights of HBSniff and our work includes: (1) capable for detecting 14 code smells and evaluating 4 mapping metrics, and no project compilation is required, (2) manually validated on 5 open-source projects and 1 commercial project, with test cases and documentations included for every smell detector and metric, (3) generates a customizable and readable report for every project, and (4) fully open-sourced on GitHub.

2. Problems and Background

Java ORM frameworks implement the Java Persistence API (JPA)[2] in order to map the tables, columns, and relationships of trending RDBMS (Relational DataBase Management Systems, *e.g.,* MySQL[3]) to the OOP-driven classes, attributes, and inheritance [5]. For example [10], classes annotated with `@Entity` indicates that it is an entity in the ORM context,

[1]https://hibernate.org/

[2]JSR 338: Java(TM) Persistence 2.2. https://www.jcp.org/en/jsr/detail?id=338

[3]https://www.mysql.com/

while the `@Table` annotation specifies its corresponding data table. Moreover, `@Id` could be used to specify the unique identifier (in most cases, the corresponding field of the Primary Key of RDBMS data table), and relational annotations like `@ManyToOne`, `@OneToMany` could be used to describe relationships between entities with `FetchType` (*e.g.*, `EAGER, LAZY`) specified to determine whether data should be fetched in advance or on demand.

HQL is a SQL-alike query language designed for ORM [10, 17]. HQL could either be generated by ORM or specified by developers. Then, ORM will translate HQL to executable SQLs for RDBMS. Later, the results of the queries would be processed by ORM and presented with the form of OOP in the context of JVM. We focus on the human-written HQLs in this work.

We implement the smells and metrics of the following works. Holder *et al.* [7] proposed a metric suite to measure ORM mapping code complexity. Silva *et al.* [11] proposed a set of rules to check if HIBERNATE entity codes follow JPA[4] specifications. Loli *et al.* [10] summarized ORM code smells proposed in prior works [3, 9, 14, 15, 16] as well as in grey literatures, and they investigated the agreement of developers towards the definition of smells. Result shows most developers agree with the definitions and severity.

3. Software Framework

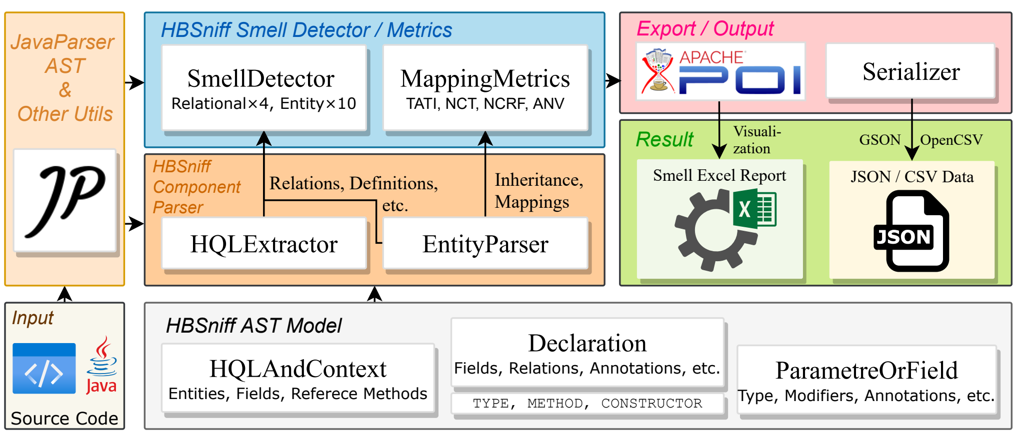


Figure 1: The general architecture of HBSNIFF.

[4]JSR 338: Java(TM) Persistence 2.2. https://www.jcp.org/en/jsr/detail?id=338

3.1. Software Architecture

HBSniff is a Java project using Maven[5] as dependency management and building tool which consists of 5 major modules (sub-packages), *i.e.*, `model`, `parser`, `detector`, `metric`, and `util`. The general architecture is depicted in Figure 1. First, users specify the path of projects and the directory of output. Then, the `parsers` of HQL and Hibernate entities construct the HBSniff AST `models` using JavaParser[6]. Afterwards, the `models` will be used to populate the context of code smell `detectors` and `metrics`, and the detection and evaluation will be performed. Finally, the results will be transferred to Excel reports as well as `csv` and `JSON` data.

3.2. Software Functionalities

In this section, we outline the ORM smells and metrics implemented in `SmellDetectors` and `MappingMetrics`.

3.2.1. Inter-Entity Relational Smells

Relational smells summarized in [10] are inter-entity smells caused by inappropriate usage of data retrieval strategies in entity relationships. The smells are listed as follows. **(1) Eager Fetch [9, 14, 15]:** `EAGER FetchType` preloads all data in advance. However, they could be retrieved on demand to improve performance. **(2) Lacking Join Fetch [9, 15]:** Fields annotated with `EAGER FetchType` should be joined by `join fetch` in HQL to avoid N+1 problems. **(3) One-By-One [9, 14]:** A collection annotated with `@OneToMany` or `@ManyToMany` using `LAZY FetchType` will be fetched one-by-one in every loop iteration. `@BatchSize` should be involved to load data on demand and in batch. **(4) Missing ManyToOne [16]:** Using `@OneToMany` annotation in a field without `@ManyToOne` presented on the other side of the relationship may lead to performance issues such as N+1.

3.2.2. Intra-Entity and Application Smells

Entity smells mostly summarized in [11] are caused by inappropriate definition or application of entity fields and methods. The smells are listed as follows. **(5) Collection Field:** Collection fields should use `Set` instead of `List` due to performance concern. **(6) Final Entity:** Using `final` classes

---

[5] https://maven.apache.org/

[6] https://javaparser.org/

as entities would disable the lazy loading functionality. **(7) Missing Identifier:** Identifier field should be specified to uniquely determine an entity. **(8) Missing No Argument Constructor:** A no argument constructor should be implemented for HIBERNATE to generate an entity object using reflection. **(9) Missing Equals Method:** The default `equals` method compares the reference of objects, which is not ideal for comparing entities especially for collection-related operations. **(10) Missing HashCode Method:** `HashCode` is vital for collections such as `HashSets` to determine equivalent entities. **(11) Using Identifier in Equals or HashCode Methods:** The identifier should not be used in `equals` and `hashCode` since all transient objects may be equal because their identifiers could be null. **(12) Not Serializable:** Detachable Entities (*i.e.,* used for data export) should implement the `Serializable` interface. **(13) Missing Accessor Methods:** JPA specification recommends implementing visible methods to access and to update private fields. **(14) Local Pagination [10]:** Built-in pagination of ORM should be used instead of fetching all data and page locally.

3.2.3. Mapping Metrics

The 4 Mapping metrics [7] are designed to evaluate data redundancy and performance of entities related to inheritance. Thresholds to identify a smell should be investigated further. **Table Accesses for Type Identification (TATI):** The number of tables needed to identify the requested type of entity. **Number of Corresponding Tables (NCT):** The number of tables that contain data of an entity, which measures object retrieval performance. **Number of Corresponding Relational Fields (NCRF):** The number of relational fields in all tables that correspond to each non-inherited non-key field of an entity, which measures change propagation. **Additional Null Values (ANV):** The number of null values in the row of union superclasses, which measures the data redundancy.

4. Implementation and Empirical Results

HBSNIFF could be executed as a command-line program under JDK version $\geq$ 1.8 with a line of command, *e.g.*, `java -jar HBSniff-1.6.7.jar -i <projectRootPath> -o <outputPath>`. The tool is shipped together with unit tests for all smells and metrics implemented and documentations for both developers and users. However, since it is also a tool used in practice, we also test it on real-world projects.

Project	Purpose	Entities	I%	R%
WeixinMultiPlatform [11]	Content management system.	26	100.00	38.46
Jpa-issuetracker [11]	Development issue tracker.	6	100.00	16.67
Broadleaf Commerce [3]	E-commerce framework.	162	100.00	71.60
Devproof Portal [7]	Blogging platform.	22	100.00	72.73
2ndInvesta [8]	Invest management.	16	100.00	62.50
CP (Commercial)	Order processing.	27	100.00	77.78

Table 1: Projects analyzed. I% refers to entities affected by intra-entity smells. R% refers to entities affected by relational smells.

4.1. Manual Validation

We perform smell detection over 5 open-source projects and 1 commercial project (`CP`). The brief introduction of the projects are listed in Table 1. We pick 3 non-toy projects from [3, 11] having actual purpose and functionality. We also randomly pick the 4th and 5th project by locating `createQuery` method calls using GitHub search to find projects performing potential HQL execution. The 6th project is used to confirm if our tool can be used in a more realistic scenario. The smells are manually validated by the 1st and the 5th author independently. The detection is all accurate and there is no missing cases. Hibernate-based projects are heavily affected by ORM smells. Thus, our implementation could foster large-scale empirical analysis to reveal the actual impact of such smells to software quality.

However, we did not find an appropriate project for assessing the 4 metrics. Nevertheless, we implement the hibernate-based examples of the original paper [7], which is also available with database generation code (ddl) in the example folder in our source code. Note that since recent versions of Hibernate does not support mixed inheritance strategy[9], our implementation is slightly different from the original paper. Finally, the 4 metrics are verified by unit tests and manual evaluation of the sample project.

4.2. Comparison with Related Tools

Difference with Respect to [11]. This work proposed a design rule checker which is capable of detecting 9 out of 10 intra-entity smells (except

[7]https://github.com/devproof/portal
[8]https://github.com/2ndStack/2ndInvesta
[9]Mixing inheritance is not allowed. https://hibernate.atlassian.net/browse/HHH-7181

for Local Pagination). However, the checker requires the analyzed project to compile, and its upstream parser (DesignWizard [10]) provide full support only for class files compiled with JDK (Java Development Kit) version $\leq$ 1.7. JDK 1.7 is no longer supported by manufacturer since April 2015 [11]. Moreover, we compiled 2 projects in the datasets in Table 1 to verify the results, and we fixed some issues in its implementation, *e.g.,* we can detect the cases of using identifier annotations in accessor methods, calls of parent methods by `super` in `equals` and `hashCode`, and we do not treat missing `equals` and `hashCode` as an occurrence of **smell (11)** since default methods compare object references instead of attributes, and so on.

The 3 Unimplemented Data Usage Smells Mentioned in Section 4.5 of [10]. The original source of the 3 smells [15] used static analysis to locate method calls of queries, and analyzed the accessed data using dynamic analysis. We did not implement them since static analysis is not able to profile execution. However, we will extend our work to find trials of redundant usage, *e.g.,* locating `findAll` and `update` operations of fetching a whole entity or table. To achieve this goal, a large-scale empirical analysis should be conducted to capture different forms of entity update. Moreover, we may propose new data usage smells, which is not within the scope of this work.

4.3. Remarks on Implementation

Exclusion of Controversial Smells. We allow users to exclude every smell in command-line parametre `-e` in case they do not perceive them as real problems, *e.g.*, the Collection Field smell may be controversial [16].

Using Third-Party Libraries to Construct Entity Classes. We cover the usage of libraries such as Lombok[12] and Apache Commons[13] to generate `equals`, `hashCode`, and accessor (`getter, setter`) methods. We may not be able to detect similar usage if practitioners use other libraries.

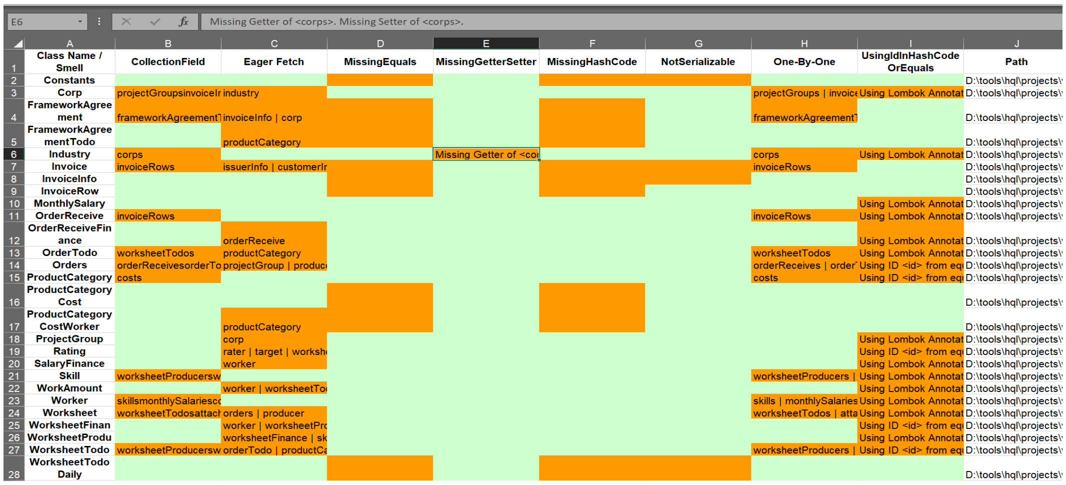
Detecting Pagination Smell. We locate the `setMaxResults` or `setFirstResult` method calls in parent methods with HQL appearance, and we analyze if the code component that called the method defines any `Integer` or `Long` object whose name contains "page" or "limit". This complies with the sample code of [10], which may be impractical, and may be improved in future.

[10]https://github.com/joaoarthurbm/designwizard

[11]End of Public Updates Notice. https://java.com/en/download/help/java_7.html

[12]https://projectlombok.org/

[13]https://commons.apache.org/


E6 | Missing Getter of <corps>. Missing Setter of <corps>.

	A	B	C	D	E	F	G	H	I	J
1	Class Name / Smell	CollectionField	Eager Fetch	MissingEquals	MissingGetterSetter	MissingHashCode	NotSerializable	One-By-One	UsingIdInHashCode OrEquals	Path
2	Constants									D:\tools\hql\projects\
3	Corp	projectGroupsinvoiceIr	industry					projectGroups | invoice	Using Lombok Annotat	D:\tools\hql\projects\
4	FrameworkAgreement	frameworkAgreementT	invoiceInfo | corp					frameworkAgreementT		D:\tools\hql\projects\
5	FrameworkAgreementTodo		productCategory							D:\tools\hql\projects\
6	Industry	corps			Missing Getter of <co			corps	Using Lombok Annotat	D:\tools\hql\projects\
7	Invoice	invoiceRows	issuerInfo | customerIr					invoiceRows		D:\tools\hql\projects\
8	InvoiceInfo									D:\tools\hql\projects\
9	InvoiceRow									D:\tools\hql\projects\
10	MonthlySalary								Using Lombok Annotat	D:\tools\hql\projects\
11	OrderReceive	invoiceRows						invoiceRows	Using Lombok Annotat	D:\tools\hql\projects\
12	OrderReceiveFinance		orderReceive						Using Lombok Annotat	D:\tools\hql\projects\
13	OrderTodo	worksheetTodos	productCategory					worksheetTodos	Using Lombok Annotat	D:\tools\hql\projects\
14	Orders	orderReceivesorderTo	projectGroup | produc					orderReceives | order	Using ID <id> from eq	D:\tools\hql\projects\
15	ProductCategory	costs						costs	Using ID <id> from eq	D:\tools\hql\projects\
16	ProductCategoryCost									D:\tools\hql\projects\
17	ProductCategoryCostWorker		productCategory							D:\tools\hql\projects\
18	ProjectGroup		corp						Using Lombok Annotat	D:\tools\hql\projects\
19	Rating		rater | target | worksh						Using ID <id> from eq	D:\tools\hql\projects\
20	SalaryFinance		worker						Using Lombok Annotat	D:\tools\hql\projects\
21	Skill	worksheetProducersw						worksheetProducers |	Using Lombok Annotat	D:\tools\hql\projects\
22	WorkAmount		worker | worksheetTo						Using Lombok Annotat	D:\tools\hql\projects\
23	Worker	skillsmonthlySalariesc						skills | monthlySalaries	Using Lombok Annotat	D:\tools\hql\projects\
24	Worksheet	worksheetTodosattach	orders | producer					worksheetTodos | atta	Using Lombok Annotat	D:\tools\hql\projects\
25	WorksheetFinan		worker | worksheetPr						Using ID <id> from eq	D:\tools\hql\projects\
26	WorksheetProdu		worksheetFinance | sk						Using Lombok Annotat	D:\tools\hql\projects\
27	WorksheetTodo	worksheetProducersw	orderTodo | productC					worksheetProducers |	Using ID <id> from eq	D:\tools\hql\projects\
28	WorksheetTodoDaily									D:\tools\hql\projects\

Figure 2: A snapshot of the EXCEL report for the CP project generated by HBSNIFF.

5. Illustrative Example

Figure 2 illustrates the exported xls report of the analyzed commercial project. Undetected smells are not presented. Fields in orange represents smelly, and texts in these fields are corresponding comments (*e.g.,* affected entity attributes). Light green fields refer to clean entities.

6. Conclusions and Future Work

We present a static analysis-based Java HIBERNATE ORM code smell detector called HBSNIFF which is capable for evaluating 14 smells and 4 mapping metrics in uncompiled Java project source codes. Moreover, we conduct unit tests and manual verification for the detectors and metrics to ensure the reliability of our implementation.

Future work includes: (1) Proposing and implementing data usage smells, (2) Extending our scope to Python ORM smells, and (3) Assess empirically the impact of ORM smells to architecture degradation.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China under Grant No. 61772200, and the Natural Science Foundation of Shanghai under Grant No. 21ZR1416300.

References

[1] F. Palomba, A. Panichella, A. Zaidman, R. Oliveto, A. De Lucia, The scent of a smell: An extensive comparison between textual and structural smells, IEEE Transactions on Software Engineering 44 (10) (2018) 977–1000.

[2] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, A. De Lucia, On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation, Empirical Software Engineering 23 (3) (2018) 1188–1221.

[3] T.-H. Chen, W. Shang, J. Yang, A. E. Hassan, M. W. Godfrey, M. Nasser, P. Flora, An empirical study on the practice of maintaining object-relational mapping code in Java systems, in: Proc. 13th International Conference on Mining Software Repositories (MSR), 2016, p. 165–176.

[4] G. Vial, Lessons in persisting object data using object-relational mapping, IEEE Software 36 (6) (2019) 43–52.

[5] A. Torres, R. Galante, M. S. Pimenta, A. J. B. Martins, Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design, Information Software Technology 82 (2017) 1–18.

[6] M. Lorenz, J.-P. Rudolph, G. Hesse, M. Uflacker, H. Plattner, Object-relational mapping revisited: A quantitative study on the impact of database technology on O/R mapping strategies, in: Proc. 50th Hawaii International Conference on System Sciences (HICSS), 2017, pp. 4877–4886.

[7] S. Holder, J. Buchan, S. G. MacDonell, Towards a metrics suite for object-relational mappings, in: Proc. 1st International Workshop on Model-Based Software and Data Integration (MBSDI), 2008, pp. 43–54.

[8] L. Meurice, C. Nagy, A. Cleve, Detecting and preventing program inconsistencies under database schema evolution, in: Proc. IEEE 16th International Conference on Software Quality, Reliability and Security (QRS), 2016, pp. 262–273.

[9] T.-H. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. N. Nasser, P. Flora, Detecting performance anti-patterns for applications developed using object-relational mapping, in: Proc. 36th International Conference on Software Engineering (ICSE), 2014, pp. 1001–1012.

[10] S. Loli, L. Teixeira, B. Cartaxo, A catalog of object-relational mapping code smells for Java, in: Proc. 34th Brazilian Symposium on Software Engineering (SBES), 2020, pp. 82–91.

[11] T. M. Silva, D. Serey, J. C. A. de Figueiredo, J. Brunet, Automated design tests to check hibernate design recommendations, in: Proc. 33th Brazilian Symposium on Software Engineering (SBES), 2019, pp. 94–103.

[12] V. Lenarduzzi, V. Nikkola, N. Saarimäki, D. Taibi, Does code quality affect pull request acceptance? an empirical study, Journal of Systems and Software 171 (2021) 110806.

[13] M. Tufano, F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, D. Poshyvanyk, There and back again: Can you compile that snapshot?, Journal of Software: Evolution and Process 29 (4) (2017) e1838.

[14] T.-H. Chen, Improving the quality of large-scale database-centric software systems by analyzing database access code, in: Proc. 31st IEEE International Conference on Data Engineering Workshops (ICDEW), 2015, pp. 245–249.

[15] T.-H. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, P. Flora, Finding and evaluating the performance impact of redundant data access for applications that are developed using object-relational mapping frameworks, IEEE Transactions on Software Engineering 42 (12) (2016) 1148–1161.

[16] P. Węgrzynowicz, Performance antipatterns of one to many association in hibernate, in: Proc. 2013 Federated Conference on Computer Science and Information Systems (FedCSIS), 2013, pp. 1475–1481.

[17] L. Meurice, C. Nagy, A. Cleve, Static analysis of dynamic database usage in java systems, in: Proc. 28th International Conference on Advanced Information Systems Engineering (CAiSE), pp. 491–506.

Required Metadata

Current executable software version

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	v1.6.7
S2	Permanent link to executables of this version	*https://github.com/HBSniff/HBSniff/releases/tag/v*1.6.7
S3	Legal Software License	GPL
S4	Computing platform/Operating System	Linux, OS X, Microsoft Windows.
S5	Installation requirements & dependencies	JDK 8.0
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	https://hbsniff.github.io/
S7	Support email for questions	hzj@mail.ecust.edu.cn

Table 2: Software metadata (optional)

Current code version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.6.7
C2	Permanent link to code/repository used of this code version	$https : //github.com/HBSniff/HBSniff$
C3	Legal Code License	GPL
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Java
C6	Compilation requirements, operating environments & dependencies	JDK 8.0, Maven 5
C7	If available Link to developer documentation/manual	https://hbsniff.github.io/
C8	Support email for questions	hzj@mail.ecust.edu.cn

Table 3: Code metadata (mandatory)