Vol. 1, No. 2, pp 18 - 41, 2020
DOI: 10.55969/paradigmplus.v1n2a2

PARADIGM *Plus*

Automated Fine Grained Traceability Links Recovery between High Level Requirements and Source Code Implementations

Alejandro Velasco [1],✉ and Jairo Aponte [1]

[1] *Universidad Nacional de Colombia, Bogotá, Colombia*
`{savelascod, jhapontem}@unal.edu.co`

Abstract

Software Traceability has been a matter of discussion in the Software Engineering community for a long time. The process of keeping and recovering traces among software artifacts in any system represents a fundamental aspect to properly perform software maintenance tasks and requirements compliance verification. Furthermore, there exist application contexts where this becomes a mandatory process, for instance, banking and healthcare. Software traceability researchers have been proposing alternatives to recover lost traceability links in coarse-grained and middle-grained levels of detail; however, proposed techniques are not enough to meet the desired levels of granularity in specific critical contexts. In this work we propose a fine-grained traceability algorithm designed to recover traces between high level requirements written in natural language and source code statements where they are implemented. We tested our approach in four open-source healthcare systems to trace constraints requirements specified by the HIPAA law, and evaluated the results as presented in this paper.

Keywords: Software Traceability · Information Retrieval · Static Code Analysis · Program Slicing · Software Maintenance · Natural Language Processing · Healthcare

Received: 18 July 2020 · Accepted: 13 August 2020 · Published: 19 August 2020.

1 Introduction

Traceability in software engineering is a research field whose main purpose is to recover lost information of traces and links between high-level artifacts (e.g., documentation, use case diagrams, requirements specifications) and source code artifacts (e.g., classes and methods) [1, 2]. Throughout the evolution of any software system, the information of existing relations between requirements, source code artifacts, and underlying infrastructures may be lost, leading to a progressive increase in the complexity of software maintenance tasks. Recovering those links among software artifacts constitutes a crucial requirement to ease code comprehension and help to perform a wide variety of tasks such as bug tracking, feature location, code restructuring, and change impact analysis. Moreover, maintaining traceability links makes easier the validation of stakeholder's needs and the verification of the correct implementation of requirements in any software system [3].

Keeping the links among software artifacts is a task that software teams should do it from the very beginning of any software project [4]. However, in practice, in most cases, developers are focused on programming tasks and forget their responsibility of creating and maintaining documentation links between the various software artifacts. Hence, when time permits, the system documentation, and some percentage of links recovery, is left for the final stage of the project [4]. These bad practices lead to significant drawbacks when assessing the quality of the constructed software products as well as the verification of their compliance with the mandatory constraints and stakeholders' needs.

In the case of critical systems, it is essential to comply with explicit privacy and security rules, as is the case of HIPAA for healthcare information systems. Software firms and regulatory entities, such as the HHS (U.S. Department of Health and Human Services) Office for Civil Rights, are in charge of verifying and ensuring the regulatory compliance of software systems [5]. Usually, regulators have no alternative but to perform manual checks of the source code to verify compliance with existing regulations [6, 7, 8]. Such a way of working is a time-consuming and error-prone traceability task, and clearly, its main cause is the absence of traceability links. To aid people in mandatory constraints verification, researchers have proposed automated approaches to traceability links recovery. Existing techniques focus mainly on linking classes and methods with high-level constraints and requirements. Nevertheless, in some cases, the verification of compliance with mandatory rules requires analysis at a finer level of granularity to confirm and assure that source code implements them.

Fine-grained traceability link recovery aims at supporting the verification process of requirements compliance of software, by connecting mandatory constraints to concrete programming statements, including conditionals, assignments, loops, and basic code snippets. The main contribution of this paper is to present an extension of our previous work [9], proposing a fine-grained traceability technique that combines heuristics, information retrieval (IR) techniques, and static software code analysis, to identify traces between software mandatory constraints and source code structures.

To address all the work presented in this paper, the following research questions were defined:

1° What are the code structures commonly used to implement different types of mandatory constraints?

2° How natural language processing, information retrieval, and static source code analysis techniques can support fine-grained traceability for mandatory constraints compliance?

3° How can we assess the accuracy of a technique for fine-grained traceability link recovery?

This paper is organized as follows: Section 2 - Related Work, briefly describes the studies relevant to our problem. Section 3 - Dataset Description, provides information about the artifacts used to conduct this study. Section 4 - Heuristics, explains the observations and remarks we take as basis for the proposed technique. Section 5 - Fine-grained traceability algorithm, presents the traceability technique designed and developed in this study. Section 6 - Results, reports the evaluation data of the proposed technique. Section 7 - Discussion, presents relevant remarks regarding the main strengths and weaknesses of the proposed approach. Lastly, the paper ends with conclusions and future work, in sections 8 and 9, respectively.

2 Related work

Requirements mapping is a time-consuming and error-prone task that is indispensable for ensuring that a software system complies with restrictions imposed by official regulations and constraints expected by stakeholders [7]. In the case of HIPAA [10], several classifications by typology have been made to facilitate the validation process and make it more complete and comprehensive. HIPAA law has three main sets of regulatory constraints; for healthcare software systems, HIPAA establishes standards and rules aimed at ensuring security and confidentiality, and enforcing correct management of all patient data. [5]. In that sense, Breaux. et al. [6, 11], propose a semantic model by identifying language patterns to extract rights and obligations from HIPAA statutes; they considered the privacy rules on HIPAA regulations for performing the requirements classification. The Breaux model was the basis for the construction of other frameworks [12, 13, 14]. Because the legal regulations are written in a complex and dense format, Alshugran et al. proposed a process to extract the privacy requirements from HIPAA; specifically, the defined process has a set of methods to analyze, extract and model privacy rules [15]. Some studies have proposed mechanisms to encourage developers to verify legal compliance with HIPAA requirements during the construction of healthcare software systems. For instance, Maxwell et al. propose a production rule model to encourage requirements engineers to keep trace between law and high level artifacts across every development process [16]; they validated their approach using iTrust, a HIPAA compliant healthcare system.

2.1 Traceability links recovery

Coarse-grained and middle-grained levels of granularity have been the most common in proposed approaches for traceability link recovery between high-level software artifacts and source code snippets [17, 18, 19, 20, 21, 22, 23, 24, 25]. Fasano [18] built ADAMS, a tool for automatic traceability management based on Latent Semantic Indexing (LSI), that improves Vector Space Model techniques [26, 27], under the assumption that most software systems have high-level software artifacts with a well defined hierarchical structure. ADAMS is able to recover traceability links between software high-level documentation and source code classes.

Marcus and Maletic [19] use Latent Semantic Indexing (LSI) for recovering traceability links between methods and documentation. Based on the comments and identifier names present in the source code, they manage to extract semantic information useful for recovering links.

Paloma et al. [23] created CRYSTAL (Crowdsourcing RevIews to SupporT App evoLution), an IR-based tool for recovering traceability links among commits, issues, and user reviews. The tool can remove useless words, calculate the textual similarity of the artifacts, and select the candidate links according to predefined criteria and threshold values.

De Lucia et al. [24] reduce the effects of noise in software artifacts by using smoothing filters. They empirically evaluated their approach and report a significant improvement of the IR techniques Vector Space Model and Latent Semantic Indexing.

Diaz et al. [25] propose TYRION, a tool for traceabilitY link recovery using information retrieval and code ownership. They use code ownership to improve the candidate links generation. Their main assumption is that if a developer authored code that is linked to a particular high-level artifact, then another code snippet created by the same author is likely to be associated with the same artifact.

2.2 Fine-grained traceability link recovery techniques

Some techniques have been proposed that perform the recovery of traceability links at a fine-grained granularity level. This means that such techniques recover links between high-level artifacts and specific lines of code [28]. The execution traces of specified features of a software system, heuristics, and source code static analysis techniques have been used for this purpose [29, 30, 31, 8, 32]. Wong et al. [32] come up with a technique based on execution slices, that receives two sets of test cases, one that exercises the feature of interest and another that does not. By using dynamic information obtained from the instrumented software system (i.e., list of statements executed), the technique can discriminate between code that is unique to a feature, and code that is common to several features.

Dagenais and Robillard [30] propose a model to recover traceability links between the software learning resources and the Application Programming Interface (API) documentation. They use a meta-model representation of the involved artifacts (i.e., documentation, source code, and support channels) to understand the context in which a code-like term is used.

Sharif and Maletic [31] address the problem of evolving traceability links. For this purpose, they update and evolve traceability links based on the differentiation among versions of the involved artifacts in a traceability link.

2.3 Limitations of adopted strategies

As already mentioned, current IR-based techniques for recovering traceability links are designed to relate software requirements to files and classes. They have focused mostly on coarse-grained granularity, so that they usually trace high-level software artifacts to files, classes, and, in best cases, to methods. The main problem with fine-granularity source code structures, such as conditionals or exception handling statements, is that they usually do not contain enough textual information that matches the vocabulary of high-level artifacts [27]. This circumstance translates into a low accuracy of the IR-based techniques. Therefore, new techniques or improvements of the existing ones should be able to generate traceability links between high-level documents, such as software mandatory constraints written in formal regulations, and low-level code structures such as conditionals, assignment statements, and method calls.

3 Dataset description

HIPAA statutes are written in a complex language. Thus, to trace source code artifacts back to high-level constraints, we defined a taxonomy of HIPAA statutes according to their likelihood of being related to software implementation of standards and regulations. Moreover, we found that not all sections of the HIPAA rules are related to healthcare management systems. Thus, we decided to focus on the HIPAA statutes most related to those systems and user data security regulations (as stated in the privacy rule). Therefore, we extracted a set of law statutes that apply for our particular problem. Table 1 summarizes the HIPAA administrative simplification. In this study we analyze the Security and Privacy rule of HIPAA since it includes most of the regulations related to the implementation of healthcare management systems.

We consider those statutes derived from HIPAA security and privacy rules, defined in the administrative simplification; then, we identify those that are suitable to associate with source code implementations and configurations in healthcare systems. As a result, we classified the law constraints using a taxonomy with three categories. The first category (A) has the rules that could be easily traced into code. These are regulations that mainly refer to functional requirements that must be implemented in every HIPAA compliant healthcare system. The second category (B) includes statutes that define constraints related to software implementation and non-functional requirements, still suitable to be traced to code. The last category (C) lumps together statutes and standards that describe how organizations and health care providers implement practices and methods to audit interactions with healthcare systems so that the private information of patients is protected; those are unsuitable to be traced to code.

Table 1: HIPAA Statutes Taxonomy

ID	Category	Subpart	Statutes Samples
A	Potentially suitable to trace into code.	Security Standards: General Rules. Administrative safeguards. Technical safeguards.	164.306(a)(1), 164.306(a)(2), 164.306(a)(3), 164.308(a)(1)(ii)(D), 164.308(a)(5)(ii)(C), 164.308(a)(5)(ii)(D).
B	Moderately suitable to trace into code.	Administrative safeguards.	164.308(a)(1)(i), 164.308(a)(1)(ii), 164.308(a)(4)(ii)(A), 164.308(a)(4)(ii)(B).
C	Indirectly related to health care software systems	Security Standards: General Rules. Administrative safeguards. Physical safeguards.	164.306(a)(1), 164.306(a)(2), 164.306(a)(3), 164.306(c), 164.306(d), 164.306(d)(1),164.306(d)(2), 164.306(d)(3).

3.1 Healthcare systems

As a first step of the design of a fine-grained traceability algorithm, we look for Java open-source healthcare software systems with available documentation (e.g. use cases and user stories). We found four candidate systems: iTrust [33, 34], OpenMRS [35], OSCAR [36] and TAPAS [37]. The available documentation has suitable descriptions of the features offered by each system. Moreover, these four health care systems are explicitly defined as HIPAA compliant in their official pages. Table 2 shows the repository and system version analysed in this study.

Table 2: Summary of open source health care systems chosen to analyze in this study.

Name	Repository	Version
TAPAS	https://sourceforge.net/projects/ap-apps/	v. 0.1
iTrust	https://sourceforge.net/projects/itrust/	v. 21
OSCAR	https://sourceforge.net/projects/oscarmcmaster/	v. 14.0.0
OpenMRS	https://github.com/openmrs	v. 1.12.x

Each health care system offers features that implement standards defined in HIPAA law. The features that we identified in all systems, by observing the deployed applications, include user authentication, patient historical data management, appointment definitions, disease reports, audit controls, and data encryption.

For each system, we identified the file extensions associated with source code artifacts, as showed in Table 3. For source code analysis we consider .java .js and .jsp extensions. Such files have Java and JavaScript statements, and also Html and custom tags; The .xml and .properties extensions correspond to files with relevant information about constants and variables used within the source code, and therefore, are also pertinent for our purpose. Lastly, files with .sql extension has SQL sentences with significant information about data and entities of the application domain that are referenced in the source code.

Table 3: Summary of source code files containing source code artifacts that were taken as the corpus of the traceability algorithm.

	iTrust	OpenMRS	OSCAR	TAPAS
JAVA	936	1545	3810	221
JSP	266	447	1655	0
JS	3636	416	875	0
XML	14	820	193	14
SQL	192	2	711	0
PROPS	7	39	39	2
TOTAL SC ARTIFACTS	5051	3269	7283	237

3.2 Extracting requirements from high level artifacts

One of the data items that we require for this study is explicit high-level requirements. To this end, we had to extract specific requirements from the available documentation on each health care system.

We set apart several text fragments from requirements explicitly declared in the software high-level artifacts (query of our algorithm). Then, we extracted software requirements embedded in sentences from the documentation of all four systems, according to the following template 1:

$$[Article] + [Subject] + [ObligationVerb] + [Complement] \tag{1}$$

Table 4: Some specific requirements extracted from the available documentation of selected healthcare systems

ID	Requirement	System	Documentation
ITRUST-1	"An HCP is able to create a patient [S1] or disable a selected patient [S2]. The create/disable patients and HCP transaction is logged (UC5)."	iTrust	http://agile.csc.ncsu.edu/iTrust/wiki/doku.php?id=requirement
OSCAR-1	"Your password is stored in an encrypted format such that even the system administrator cannot find out what it is. If you have forgotten your user name and/or password, your administrator can reset the password for you but he/she cannot tell you what the original password was"	OSCAR	http://oscarmanual.org/oscar_emr_12/General%20Operation/access-preferences-and-security/accessing-oscar
TAPAS-1	"The system must have the ability to manage users in the system. Like clinical data, users should not be able to be deleted from the system as they will be tied to activities in a record."	TAPAS	http://tap-apps.sourceforge.net/docs/use-cases.html#UC-SYSADM-01

Thus, the extracted requirements were textually taken from documentation without any modification. Furthermore, the extraction process was conducted only from available Use Cases and User Stories. Those artifacts convey information about requirements and user needs that is essential for subsequent implementation tasks. A User Story describes functionality that will be valuable to any actor, ranging from software developers to final users [38]; therefore, it is likely probable to find implementations in source code based on textual information from User Stories. By the same token, Use Cases typically describe interactions between the system and its users that allow them to achieve their goals. Thus, they also represent a mechanism to define software requirements that somehow will be represented in source code.

In some cases, following the precise definition of the format leads to the extraction of phrases with no sense or context. For that reason, in such particular cases, we included in the extraction additional information from the surrounding text of such phrases to complete them, i.e., terms that were not considered when the filter was applied. Table 4 shows some examples of specific requirements extracted from artifacts of each system.

Finally, we selected all the extracted requirements that were more suitable to trace into code according to the proposed taxonomy of HIPAA Security and Privacy regulations. In other words, we associated each extracted requirement with a HIPAA statute and focused on those in the categories (A) and (B) of the classification in Table 1.

3.3 Source Code extraction

Before proposing a traceability algorithm to link up requirements and source code structures, we study in detail how different software healthcare systems are built. The intent of this process was to identify patterns in the source code and come up with a solution based on empirical observations. In this section we describe the examination process conducted through the source code of the selected healthcare systems presented in Table 2.

For each of the requirements selected from all the systems (those that were related to any HIPAA statute), we identified files and source code lines that played a dominant role in their implementation. This process required a joint effort that involved several people with knowledge of the programming

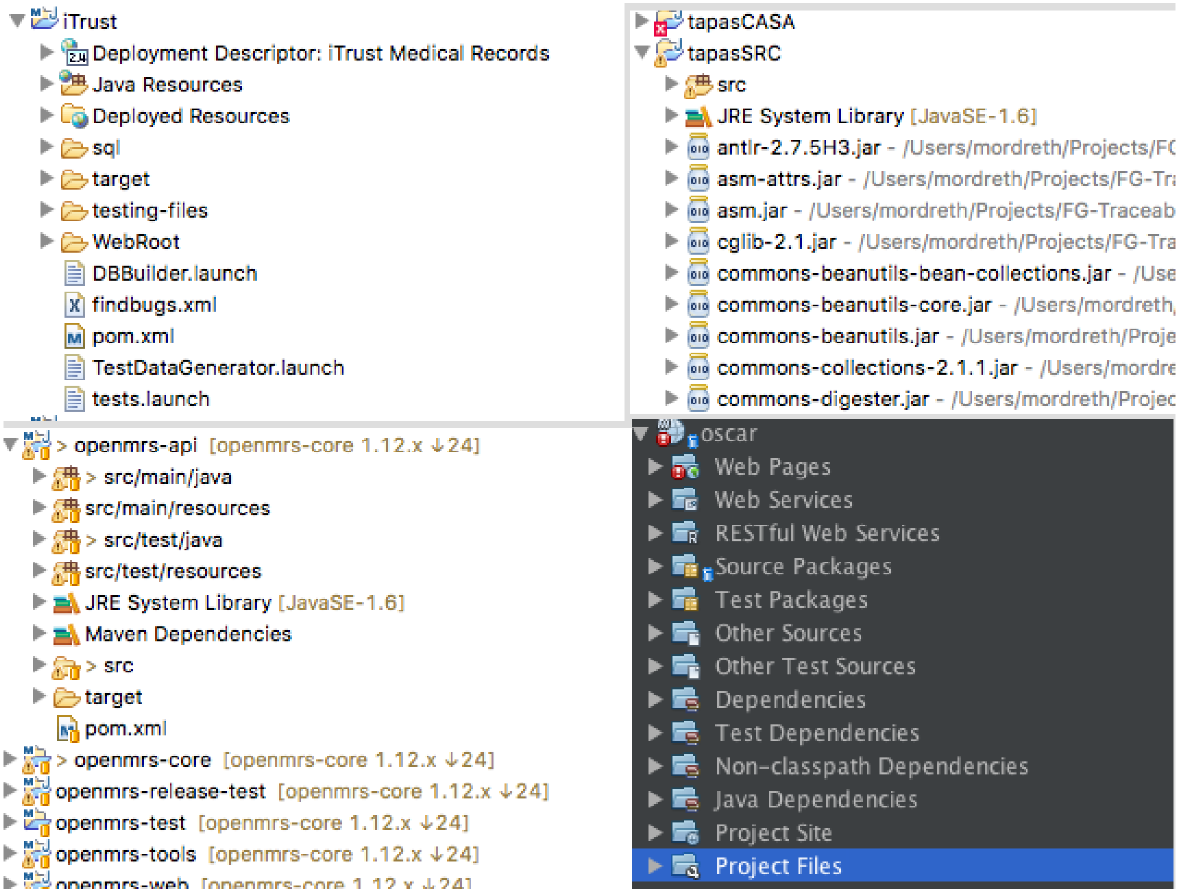

iTrust
Deployment Descriptor: iTrust Medical Records
Java Resources
Deployed Resources
sql
target
testing-files
WebRoot
DBBuilder.launch
findbugs.xml
pom.xml
TestDataGenerator.launch
tests.launch
tapasCASA
tapasSRC
src
JRE System Library [JavaSE-1.6]
antlr-2.7.5H3.jar - /Users/mordreth/Projects/FC
asm-attrs.jar - /Users/mordreth/Projects/FG-Tr
asm.jar - /Users/mordreth/Projects/FG-Traceab
cglib-2.1.jar - /Users/mordreth/Projects/FG-Tra
commons-beanutils-bean-collections.jar - /Use
commons-beanutils-core.jar - /Users/mordreth,
commons-beanutils.jar - /Users/mordreth/Proje
commons-collections-2.1.1.jar - /Users/mordre
commons-digester.jar - /Users/mordreth/Projec
> openmrs-api [openmrs-core 1.12.x ↓24]
> src/main/java
src/main/resources
> src/test/java
src/test/resources
JRE System Library [JavaSE-1.6]
Maven Dependencies
> src
target
pom.xml
> openmrs-core [openmrs-core 1.12.x ↓24]
openmrs-release-test [openmrs-core 1.12.x ↓24]
openmrs-test [openmrs-core 1.12.x ↓24]
openmrs-tools [openmrs-core 1.12.x ↓24]
openmrs-web [openmrs-core 1.12.x ↓24]
oscar
Web Pages
Web Services
RESTful Web Services
Source Packages
Test Packages
Other Sources
Other Test Sources
Dependencies
Test Dependencies
Non-classpath Dependencies
Java Dependencies
Project Site
Project Files

Figure 1: Overview of source package organization for each of the healthcare systems selected in the this study. On left top, iTrust source packages, right-top TAPAS source packages, bottom-left OPENMRS source packages, bottom-right OSCAR source packages.

languages used in the implementation of the healthcare systems.

Once deployed and configured the development environments for the four healthcare systems according to the specifications given in user manuals and deployment instructions, we performed the analysis of the source code. In general terms, given a requirement, we conducted a process of exploration starting from the layer of views, followed by the layer of services until the layer of entities at the database level. Figure 1 shows an overview of the source code organization from the used IDEs for each of the projects.

For all the selected requirements extracted from documentation, we collected detailed information about the source code files used in the implementation, the methods that were defined in those files and also the strictly related code lines within each method. For each trace of source code, we register the information to be processed later, when evaluating the effectiveness of the proposed traceability algorithm.

4 Heuristics

Over time, the solutions to complex problems in software engineering have often been based on premises that greatly simplify the search for an answer. These premises can be derived from the direct observation of a problem in such a way that through an empirical process, the limits and starting points of a possible solution can be defined. Heuristics are often used in multiple contexts to propose possible solutions to problems of diverse characteristics, such as design patterns [39], computational problems[40], optimization algorithms [41] and fault location in software [42].

In this context, we denote the term heuristic as a problem-solving approach used to improve the accuracy of our solution. Due to the nature of the problem that we tackle, we propose some heuristics to reach an approximate solution to find traceability links between artifacts whose domains are quite dissimilar, i.e., source code and natural language documents.

Based on observations derived from the manual process of traceability links recovering between the source code and the high level requirements described in natural language, we defined a set of heuristics that determined the basis for the design of an automated fine-grained traceability algorithm.

4.1 (H1) Heuristic 1: Developers tend to use terms present in high-level requirements when writing source code

Before giving a solution to a specific domain problem, any software development process goes through a phase of abstraction in the first place. Abstracting a problem requires an understanding of all the business rules, entities and interactions between the different components. The abstraction process requires identifying the entities that interact within a problem domain. To avoid losing the context of the problem when creating a solution, software architects and developers often naturally use descriptive terms to generate the software artifacts needed for the construction of a system. For instance, some programming paradigms such as Object Oriented Programming [43] and Aspects Oriented Programmings [44] are based on abstraction processes to identify actors and key entities in a determined context; once such key entities have been identified, as well as the business rules present in the domain, it is feasible to implement a suitable solution that models the real situation.

4.2 (H2) Heuristic 2: Comments in source code are a valuable source of information that can be exploited to increase the accuracy of any traceability algorithm

In addition to the information encoded in the source code identifiers, developers working on open source projects often write comments to explain functionalities that are implemented at the source code level [45, 46]. Making comments in the code is a practice that could be reduced as long as good coding techniques are followed. In high-level programming languages, the source code itself should be sufficiently explanatory so that no additional comments or explanations are required. [47]. In spite of that, in the health care management systems selected for conducting this study, the frequency of comments and explanatory texts within the code is very high; comments range from explanations about the implemented code to rudeness or bad words.

While it is true that the presence of comments in the source code can improve the accuracy of a traceability algorithm based on textual similarity, it must be kept in mind that not all the comments in the source code provide a significant input. In some cases, stated information is not related at all with the requirements that are being implemented.

4.3 (H3) Heuristic 3: Subjects in texts that describe a high-level software requirement regularly represent entities in the source code

Given a software requirement written in natural language, there is often a subject, a verb, and an object of determined action. For that reason and as a consequence of H1, when developers are implementing a requirement at the source code level, entities, methods, and variables names usually include terms directly associated with the subjects and objects described in a high-level requirement.

As our problem domain is health care, there is a great variety of terminology that if not adopted when a requirement is implemented, it would be hard and tedious to understand the code. In the particular context of healthcare management systems, terms like “patient”, "appointment", "disease" and "health care provider" are usually present in a large number of artifacts.

4.4 (H4) Heuristic 4: Conventions used by developers to name source code entities can be used to define patterns to improve accuracy in IR algorithms

Conventions in the source code are a mechanism accepted by the vast majority of developers to conduct an specific development process [48, 49]. Nowadays, there are many frameworks that prioritize convention over configuration allowing developers to focus on the comprehension of the problem domain and business rules that must be implemented, instead of tedious and repetitive technical details. For instance, frameworks such as Ruby on Rails and Groovy on Grails provide a big set of code artefacts tested by the community; there is a well-known collection of tools that allow developers to assume conventions and reuse tested artefacts in the development process leaving aside technical details, In other words, developers trust the functional code tested by the community and focus their efforts on the problem domain.

Although the health care systems that we analyzed in this study were not built over the set of tools offered by a framework such as Ruby on Rails, the adopted conventions that we managed to observe after a detailed analysis of the source code include ways to name the different code artifacts (e.g. camelCase, UpperCamelCase), design patterns, folder structures, and code assets organization.

Well defined conventions are present in projects where a large number of people work together (e.g. Health care systems selected); in that way it is more addressable to articulate teamwork and facilitate readability of the source code. In open source projects, the community usually defines its own conventions.

4.5 (H5) Heuristic 5: Delegation is ubiquitous in object-oriented systems and can be used to determine the artifacts that are involved in the implementation of a high-level requirement

Delegation is a concept that can be extrapolated in any context since every single part of any system can assign responsibilities to another one perform a particular task. Delegation in source code is a design pattern where different objects that make up a system, expose behaviors in such a way that the responsibility to perform a certain operation does not fall on a single member but is assigned to different instances and in this way, the work is distributed [47]. The software delegation facilitates the structuring of the source code and facilitates software maintenance tasks. Delegation in general is always present when the architecture of a system is defined.

This principle is important and should not be overlooked to propose a solution to the traceability problem. Since different members in the source code delegate responsibilities, it would be wrong to assume that the implementation of a high level requirement will be condensed in a single member or component within the system.

As a consequence of this premise, in the context of our problem, it is necessary to support any Information Retrieval (IR) process based on textual similarity with techniques of Static Code Analysis to determine delegation interactions between members that may be involved in the implementation of a high-level requirement.

5 Fine-grained traceability algorithm

A great variety of techniques have been formulated to perform traceability recovery processes in software artefacts. In order to verify compliance of regulatory constraints in a fine granular level, we proposed an algorithm that take advantage of information retrieval (IR) techniques, static code analysis and heuristics derived from observations after a process of manual analysis over source code implementations. Figure 1, shows a summary of our technique.

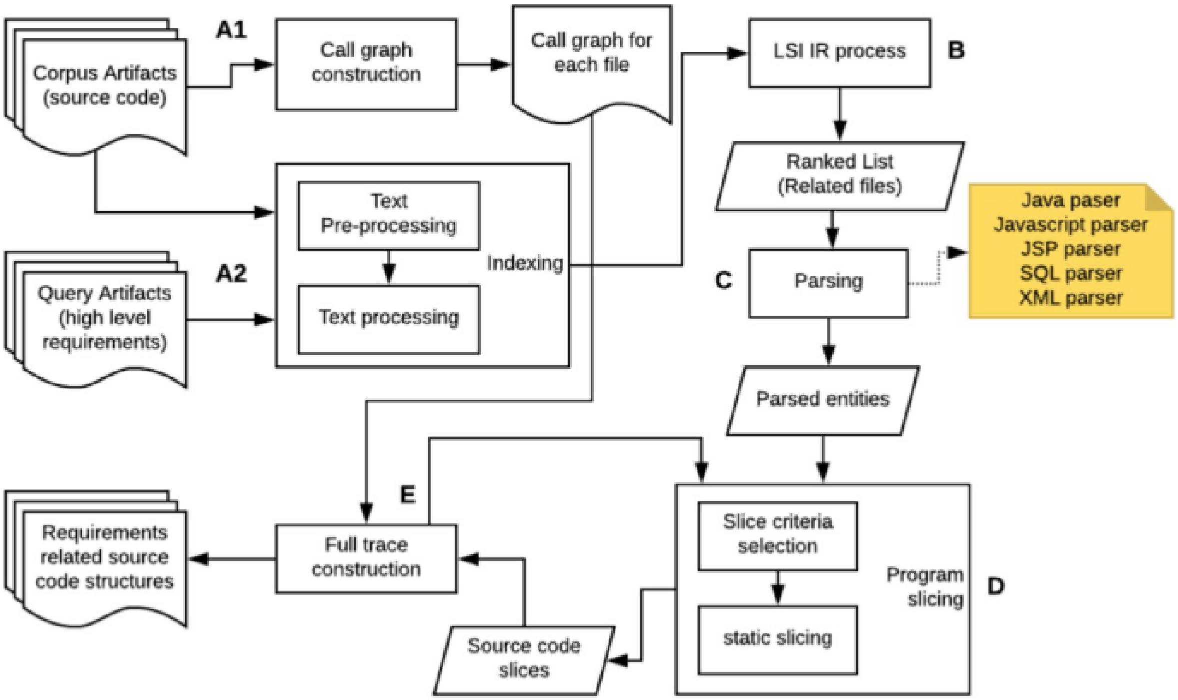


Figure 2: Flow diagram of proposed Fine-Grained traceability algorithm.

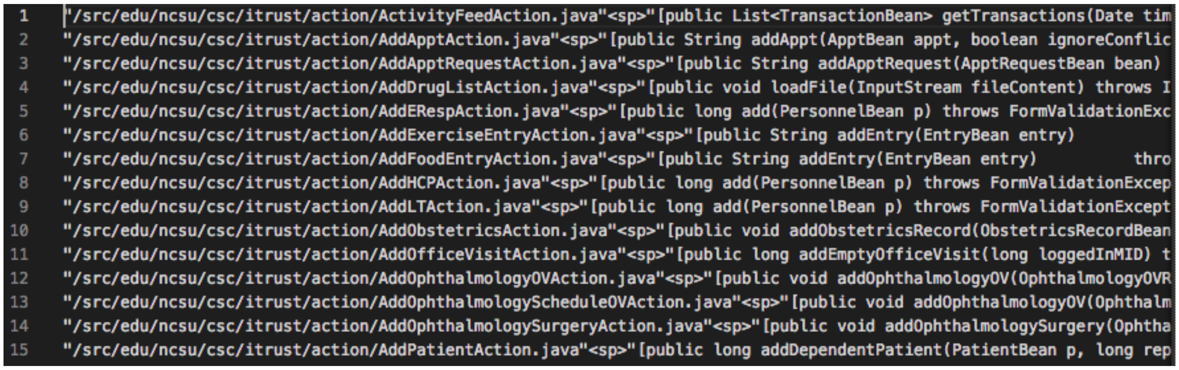
5.1 A1: Call graph construction from source code files

The source code files and artifacts and relevant structures that are part of the implementation of a system are all of a quite different nature. Java classes are constituted by attributes and methods; JavaScript code files are batch processing files that are stored in plain text; JSP files are a combination of Java code, JavaScript code, Tomahawk and Html tags. The process of A1 as depicted in Figure 2, can be summarized as the generation of a plain text file with the information of methods headers (i.e. access modifier [optional], return type, name, parameter list, exception throws [optional]) and their paths in the different source code artifacts. Figure 3, shows more detailed view of this file.

5.2 A2: Indexing phase

Before going through the algorithm of information retrieval, several processing tasks are performed on the text for both the corpus and the query. The query is a list of requirements writing in natural language to be traced into the corpus, which is the source code of a selected system.

When dealing with two entries of a different nature, the treatment of the terms that make up the corpus and the query is essentially different in both cases. Concerning the treatment performed on the



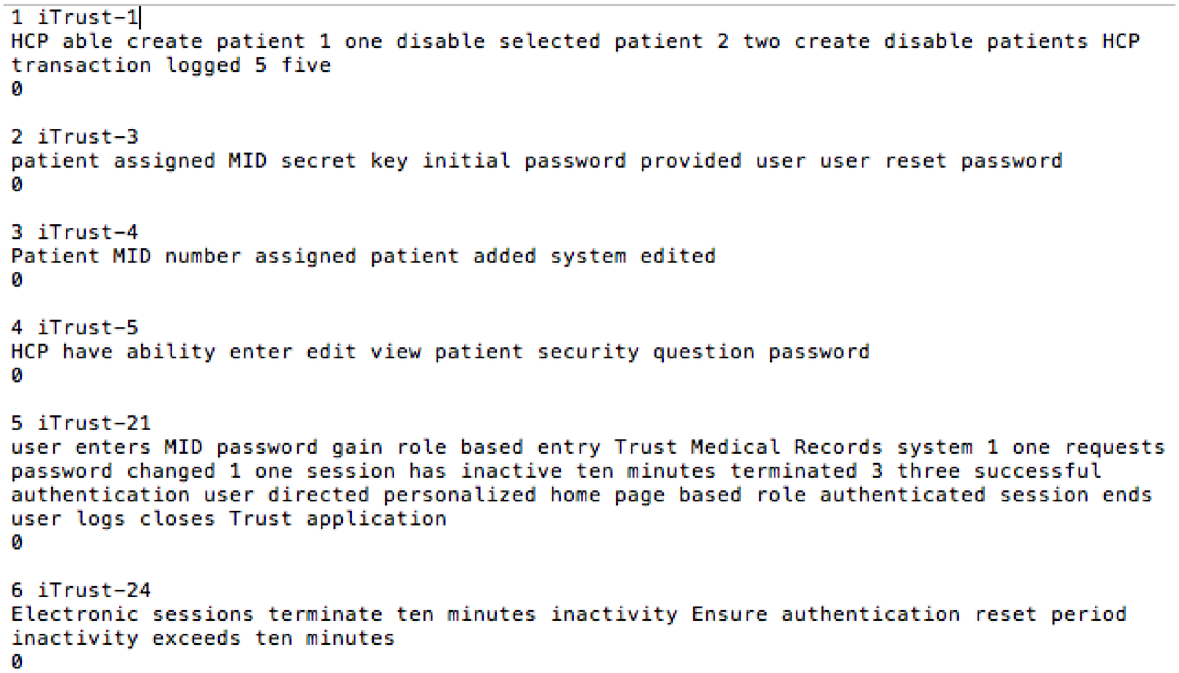
```
"/src/edu/ncsu/csc/itrust/action/ActivityFeedAction.java"<sp>"[public List<TransactionBean> getTransactions(Date tim
"/src/edu/ncsu/csc/itrust/action/AddApptAction.java"<sp>"[public String addAppt(ApptBean appt, boolean ignoreConflic
"/src/edu/ncsu/csc/itrust/action/AddApptRequestAction.java"<sp>"[public String addApptRequest(ApptRequestBean bean)
"/src/edu/ncsu/csc/itrust/action/AddDrugListAction.java"<sp>"[public void loadFile(InputStream fileContent) throws I
"/src/edu/ncsu/csc/itrust/action/AddERespAction.java"<sp>"[public long add(PersonnelBean p) throws FormValidationExc
"/src/edu/ncsu/csc/itrust/action/AddExerciseEntryAction.java"<sp>"[public String addEntry(EntryBean entry)
"/src/edu/ncsu/csc/itrust/action/AddFoodEntryAction.java"<sp>"[public String addEntry(EntryBean entry)        thro
"/src/edu/ncsu/csc/itrust/action/AddHCPAction.java"<sp>"[public long add(PersonnelBean p) throws FormValidationExcep
"/src/edu/ncsu/csc/itrust/action/AddLTAction.java"<sp>"[public long add(PersonnelBean p) throws FormValidationExcept
"/src/edu/ncsu/csc/itrust/action/AddObstetricsAction.java"<sp>"[public void addObstetricsRecord(ObstetricsRecordBean
"/src/edu/ncsu/csc/itrust/action/AddOfficeVisitAction.java"<sp>"[public long addEmptyOfficeVisit(long loggedInMID) t
"/src/edu/ncsu/csc/itrust/action/AddOphthalmologyOVAction.java"<sp>"[public void addOphthalmologyOV(OphthalmologyOVR
"/src/edu/ncsu/csc/itrust/action/AddOphthalmologyScheduleOVAction.java"<sp>"[public void addOphthalmologyOV(Ophthalm
"/src/edu/ncsu/csc/itrust/action/AddOphthalmologySurgeryAction.java"<sp>"[public void addOphthalmologySurgery(Ophtha
"/src/edu/ncsu/csc/itrust/action/AddPatientAction.java"<sp>"[public long addDependentPatient(PatientBean p, long rep
```

Figure 3: First 15 lines of the method mapping file for iTrust

query, tokenization, stemming, stopword removal, part of speech (a.k.a. POS) tagging, punctuation removal, number spell out and word embedding [50] tasks are all performed. On the other hand, the tasks carried out with the terms that make up the corpus include stop word removal, splitting by camel case, punctuation removal and tokenization. Figures 4 and 5 show examples of corpus and query files after the indexing phase.

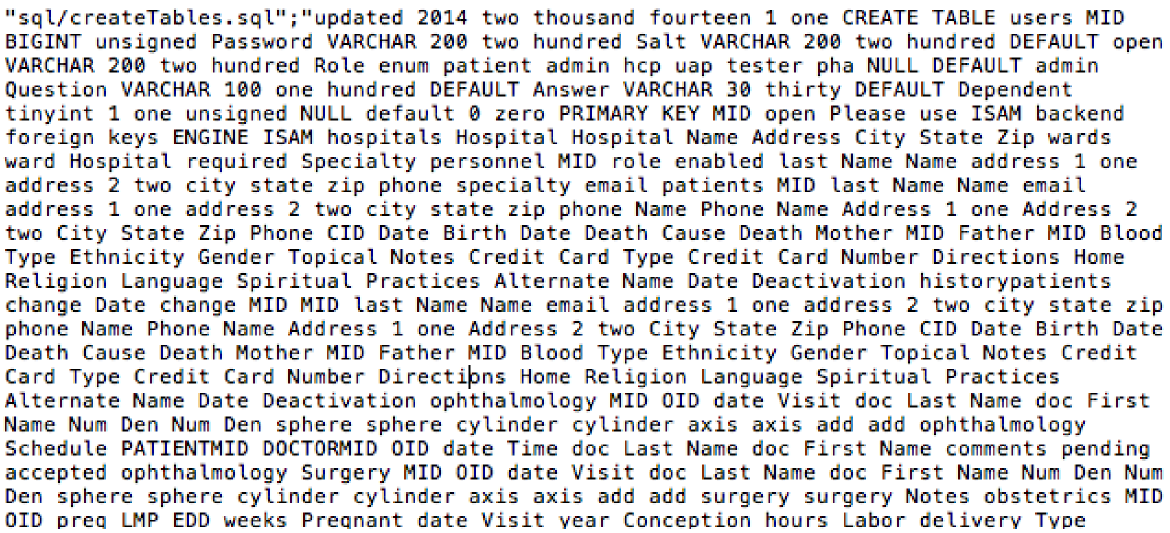
5.3 B: Information retrieval process

Once the corpus and the query are indexed, they are treated as entries for the information retrieval algorithm to obtain a ranked list of the possible corpus files that are most likely related to the text of the query requirements.



```
1 iTrust-1
HCP able create patient 1 one disable selected patient 2 two create disable patients HCP
transaction logged 5 five
0

2 iTrust-3
patient assigned MID secret key initial password provided user user reset password
0

3 iTrust-4
Patient MID number assigned patient added system edited
0

4 iTrust-5
HCP have ability enter edit view patient security question password
0

5 iTrust-21
user enters MID password gain role based entry Trust Medical Records system 1 one requests
password changed 1 one session has inactive ten minutes terminated 3 three successful
authentication user directed personalized home page based role authenticated session ends
user logs closes Trust application
0

6 iTrust-24
Electronic sessions terminate ten minutes inactivity Ensure authentication reset period
inactivity exceeds ten minutes
0
```

Figure 4: Fragment of query file for iTrust

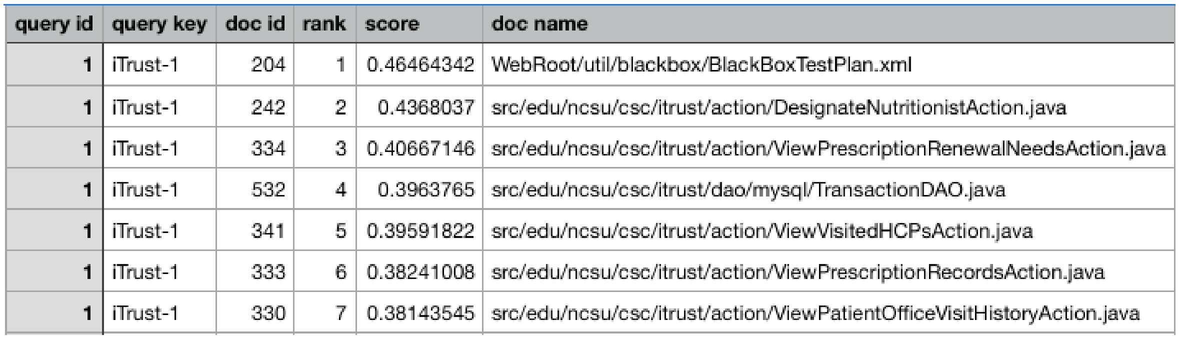


```
"sql/createTables.sql";"updated 2014 two thousand fourteen 1 one CREATE TABLE users MID
BIGINT unsigned Password VARCHAR 200 two hundred Salt VARCHAR 200 two hundred DEFAULT open
VARCHAR 200 two hundred Role enum patient admin hcp uap tester pha NULL DEFAULT admin
Question VARCHAR 100 one hundred DEFAULT Answer VARCHAR 30 thirty DEFAULT Dependent
tinyint 1 one unsigned NULL default 0 zero PRIMARY KEY MID open Please use ISAM backend
foreign keys ENGINE ISAM hospitals Hospital Hospital Name Address City State Zip wards
ward Hospital required Specialty personnel MID role enabled last Name Name address 1 one
address 2 two city state zip phone specialty email patients MID last Name Name email
address 1 one address 2 two city state zip phone Name Phone Name Address 1 one Address 2
two City State Zip Phone CID Date Birth Date Death Cause Death Mother MID Father MID Blood
Type Ethnicity Gender Topical Notes Credit Card Type Credit Card Number Directions Home
Religion Language Spiritual Practices Alternate Name Date Deactivation historypatients
change Date change MID MID last Name Name email address 1 one address 2 two city state zip
phone Name Phone Name Address 1 one Address 2 two City State Zip Phone CID Date Birth Date
Death Cause Death Mother MID Father MID Blood Type Ethnicity Gender Topical Notes Credit
Card Type Credit Card Number Directions Home Religion Language Spiritual Practices
Alternate Name Date Deactivation ophthalmology MID OID date Visit doc Last Name doc First
Name Num Den Num Den sphere sphere cylinder cylinder axis axis add add ophthalmology
Schedule PATIENTMID DOCTORMID OID date Time doc Last Name doc First Name comments pending
accepted ophthalmology Surgery MID OID date Visit doc Last Name doc First Name Num Den Num
Den sphere sphere cylinder cylinder axis axis add add surgery surgery Notes obstetrics MID
OID preg LMP EDD weeks Pregnant date Visit year Conception hours Labor delivery Type
```

Figure 5: Fragment of corpus file for iTrust

After executing the information retrieval algorithm, **k** files are obtained in order of relevance for each requirement; that is, if the query file contains a set of **n** requirements in total, the list of ranked links will contain a total of **kn** files most related to each entry in the query file.

The number of links taken into account (**k**) for each requirement was determined empirically. In other words, after 20 executions of FGTHunter over the selected systems (i.e. 5 evaluations for each selected system) we determined that the optimal value for **k** fluctuates between 25 and 30 files per requirement. This number is enough to obtain the most strictly relevant files, ignoring those that are less related to requirements. An example of the ranked list of links given by the traceability algorithm are shown in Figure 6.



query id	query key	doc id	rank	score	doc name
1	iTrust-1	204	1	0.46464342	WebRoot/util/blackbox/BlackBoxTestPlan.xml
1	iTrust-1	242	2	0.4368037	src/edu/ncsu/csc/itrust/action/DesignateNutritionistAction.java
1	iTrust-1	334	3	0.40667146	src/edu/ncsu/csc/itrust/action/ViewPrescriptionRenewalNeedsAction.java
1	iTrust-1	532	4	0.3963765	src/edu/ncsu/csc/itrust/dao/mysql/TransactionDAO.java
1	iTrust-1	341	5	0.39591822	src/edu/ncsu/csc/itrust/action/ViewVisitedHCPsAction.java
1	iTrust-1	333	6	0.38241008	src/edu/ncsu/csc/itrust/action/ViewPrescriptionRecordsAction.java
1	iTrust-1	330	7	0.38143545	src/edu/ncsu/csc/itrust/action/ViewPatientOfficeVisitHistoryAction.java

Figure 6: First 7 results in the ranked list of artifacts generated for iTrust after applying LSI algorithm.

5.4 C: Parsing phase

Once the list of files, most related to a given requirement based on textual similarity is obtained, the next phase of the algorithm consists of analyzing source code implementations and perform abstractions through program slicing, to find structures at source code level that are related to the implementation of a particular requirement. To correctly extract different structures within the code, we use parsers for each type of file included in the corpus file. The parsers statically analyze code elements present in .java, .js, .jsp, .xml, .sql, .properties and plain text files. Parsers receive as input a source code fragment and give us an Abstract Syntax Tree (AST) representation of it in a way that makes it easier to process.

5.5 D: Program slicing process

Some mechanisms at source code level have been proposed to support traditional optimization techniques. One of those mechanisms is **program slicing**; this abstraction technique has multiple applications in software engineering such as testing [51], refactoring of the source code, reverse engineering, software comprehension and software maintenance [52].

Formally, we can define program slicing as an abstraction process [53] that takes place at the code level, with the purpose of looking for fragments (slices) of code that directly or indirectly affect the state of a variable (slicing criteria); in this way it is possible to simplify the source code of the programs focusing on previously selected semantic or syntactic aspects. The slicing process eliminates all those parts of the code that are not related or that have no effect on the criterion of interest [54].

After executing the parsers for each type of source code file, it is possible to extract code fragments of interest since we are able to perform an exhaustive static analysis of source code. For each of the files that were identified by the information retrieval algorithm (LSI), slicing criteria were defined considering the subjects present in the description of each requirement. Within the source code, there are terms (e.g. subjects, adjectives, verbs) of the problem context and therefore it is possible to define slicing criteria from the names of variables, objects, classes, and methods. To illustrate this point, let's assume that a requirement **r1** contains 4 subject names; so it is possible to define 4 slicing criteria to conduct a program slicing process in each file given by the retrieval information algorithm for **r1** after executing the process defined in B (Figure 2). For each source code file, a slicing process is performed to get the code lines within each file that affect or are affected by a particular criterion. The result of this operation is a set of slices for each file ranked in B.

5.6 E: Full trace construction phase

The information given by the process of abstraction of the source code through program slicing is not enough to build entirely the trace of a requirement. To put it another way, consider a slice of source code given by the process defined in D; it is highly likely that such a code snippet contains calls to other methods within the system that do not necessarily contain terms in common with a high-level requirement but that fulfill a very important role in the implementation. This problem can also be seen as a consequence of delegation.

In this phase of the algorithm, the code snippets obtained in the slices are analyzed and, for each call to an external method that is not defined in the parent file, a search process is carried out according to the definition header of the method that is invoked with the support of the mapping file obtained in A1. Each of the referenced methods is also included in the trace of the requirement, and recursively, a slicing process D is performed.

As a final result of the execution of the traceability algorithm for a particular project, a list of files and source code structures that are most related to each of the requirements specified in the query file are obtained. Figures 7 and 8 show an example of the expected output after the full traceability algorithm execution.

To evaluate and test our approach, we developed a Java program that implements the proposed traceability technique. It was named **F**ine **G**rained **T**raceability **Hunter** (**FGTHunter**). We also implemented tools designed for static analysis of source code, as well as information retrieval algorithms.

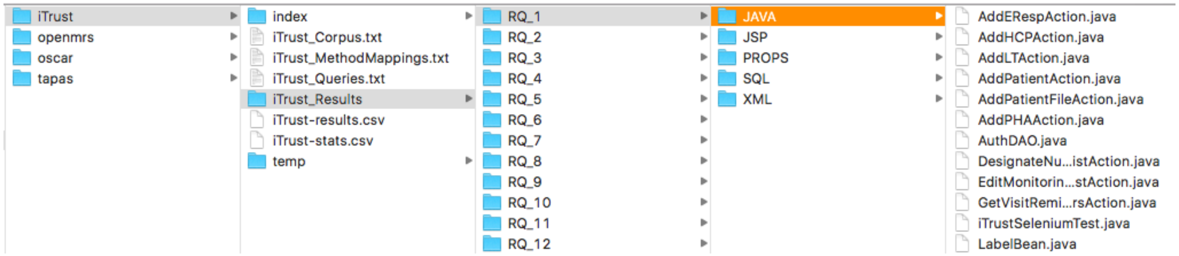
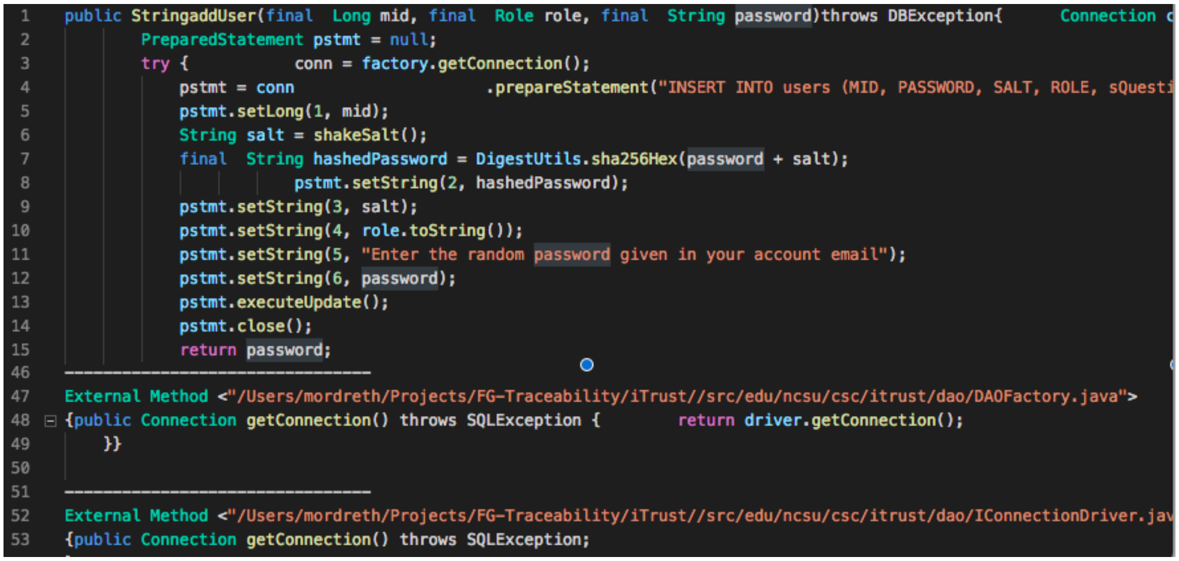


Figure 7: Folder structure of results obtained after the execution of the traceability algorithm for iTrust. Parent folder contains stats about the algorithm execution, corpus and query files, method mapping file and result summary file. Results folder contains subfolders for each requirement specified in the query file, containing the most related artefacts and source code lines for a given requirement.



```
public StringaddUser(final  Long mid, final  Role role, final  String password)throws DBException{      Connection c
        PreparedStatement pstmt = null;
        try {           conn = factory.getConnection();
            pstmt = conn                                .prepareStatement("INSERT INTO users (MID, PASSWORD, SALT, ROLE, sQuesti
            pstmt.setLong(1, mid);
            String salt = shakeSalt();
            final  String hashedPassword = DigestUtils.sha256Hex(password + salt);
                        pstmt.setString(2, hashedPassword);
            pstmt.setString(3, salt);
            pstmt.setString(4, role.toString());
            pstmt.setString(5, "Enter the random password given in your account email");
            pstmt.setString(6, password);
            pstmt.executeUpdate();
            pstmt.close();
            return password;
--------------------------------
External Method <"/Users/mordreth/Projects/FG-Traceability/iTrust//src/edu/ncsu/csc/itrust/dao/DAOFactory.java">
{public Connection getConnection() throws SQLException {        return driver.getConnection();
    }}

--------------------------------
External Method <"/Users/mordreth/Projects/FG-Traceability/iTrust//src/edu/ncsu/csc/itrust/dao/IConnectionDriver.jav
{public Connection getConnection() throws SQLException;
```

Figure 8: Example of the content for a particular result file. The source code slices and method calls are all included in the trace.

6 Results

We executed the traceability algorithm implemented in the FGTHunter tool for each of the four open source systems. We collected the results and evaluated the precision of our technique to find traceability links between high level artifacts and source code lines. The algorithm was able to successfully find a large number of the high-level requirements filtered for this study, along with new traceability links that were not originally found by the manual construction of traces.

The results were grouped for each of the analyzed systems. Table 5 summarises the findings for Trust system; Table 6 shows the collected results for OPENMRS, Table 7 presents the results for TAPAS and Table 8 collects the results for OSCAR.

To evaluate the performance of our technique, we calculated the F1 score for each extracted requirement that was associated with a HIPAA statute; then, we calculated the average of F1 scores obtained in each statute. Results were grouped for each of the analyzed systems. Table 9 summarizes our findings. Provided that, it is important to denote that for a given HIPAA statute, not always every

Table 5: FGTHunter results for iTrust requirements.

R-ID	Total Links	True Positives	True Positives	Not Found	GOLD SET	Precision	Recall	F1
1	706	265	441	0	4	0,38	1	0,55
3	341	149	192	0	24	0,44	1	0,61
4	411	50	361	17	22	0,12	0,75	0,21
5	382	18	364	1	1	0,05	0,95	0,09
21	306	62	244	6	48	0,2	0,63	0,31
24	25	12	13	4	8	0,48	0,71	0,57
25	97	26	71	9	33	0,27	0,76	0,4
37	121	35	86	0	1	0,29	1	0,45
38	120	31	89	0	1	0,26	1	0,41
40	297	116	181	0	6	0,39	0,98	0,56
45	514	161	353	0	28	0,31	0,98	0,47
47	411	165	246	1	12	0,4	1,01	0,57
110	165	79	86	3	9	0,48	0,99	0,64
111	164	79	85	3	8	0,48	1	0,65
143	906	14	892	22	22	0,02	0,39	0,03
144	498	170	328	3	30	0,34	0,87	0,49
145	166	43	123	0	7	0,26	1	0,41
163	222	41	181	0	9	0,18	1	0,31
164	456	12	444	9	10	0,03	0,57	0,05
187	120	97	23	0	1	0,81	1	0,89
189	130	67	63	0	5	0,52	0,93	0,66
191	229	3	226	0	1	0,01	0,75	0,03
180	74	36	38	0	1	0,49	1,13	0,68

selected system had at least one related high level requirement; that explain why the value of column "Average F1 score" was not calculated in all the four system in each HIPAA statute.

The audit management is a fundamental part of any software system that aims to keep track of all the operations performed by every actor that use the system. Table 9 shows the average of the harmonic mean (average of F1 score) for the requirements in each healthcare system that are related to HIPAA statute 164.312 (b). The precision of FGTHunter for iTrust and OSCAR systems had an approximate value of 0.4 whereas in TAPAS the precision was much lower. When performing the manual code inspection, we noticed that very few code lines in TAPAS handled the audit control; this fact may explain the low precision in this case.

The access control to the information within a system ensures the correct manipulation of the data. In the four systems that we analyzed, the access to the information was restricted by the definition of roles and user permissions to view, read, modify, or remove data. Table 9 shows the F1 score means of FGTHunter obtained when using it for finding the lines of code most related to the requirements

Table 6: FGTHunter results for OPENMRS requirements.

R-ID	Links Retrieved	True Positives	True Positives	Not Found	GOLD SET	Precision	Recall	F1
53	1338	1041	297	0	11	0,78	1	0,88
80	428	226	202	10	33	0,53	0,89	0,66

Table 7: FGTHunter results for TAPAS requirements.

R-ID	Links Retrieved	True Positives	True Positives	Not Found	GOLD SET	Precision	Recall	F1
1	939	179	760	11	11	0,19	0,94	0,32
3	248	77	171	12	19	0,31	0,87	0,46
21	872	393	479	8	14	0,45	0,98	0,62
22	479	179	300	10	14	0,37	0,95	0,54
42	16	1	15	10	10	0,06	0,1	0,08
43	647	17	630	10	10	0,03	0,63	0,05
44	139	30	109	4	4	0,22	0,88	0,35
51	20	3	17	10	10	0,15	0,3	0,2
54	4610	2893	1717	0	12	0,63	1	0,77
56	16	1	15	10	10	0,06	0,1	0,08
57	1231	452	779	0	13	0,37	1	0,54
59	16	1	15	10	10	0,06	0,1	0,08
60	3608	1465	2143	0	20	0,41	1	0,58
62	16	1	15	10	10	0,06	0,1	0,08
70	3236	1337	1899	0	10	0,41	1	0,58
72	343	8	335	7	12	0,02	0,53	0,04
73	16	1	15	10	10	0,06	0,1	0,08
75	288	134	154	0	10	0,47	1	0,64
77	336	89	247	8	12	0,26	0,92	0,41
81	336	89	247	8	12	0,26	0,92	0,41

associated with the statute 164.312 (a) (1) of HIPAA in each system. The average harmonic mean has an acceptable level, greater than 0.4 points, in OPENMRS, OSCAR and TAPAS. On the other hand, for iTrust the algorithm had the lowest performance. What happens is that the access control by roles in iTrust was handled by access restrictions defined in the configuration .xml files for the TOMCAT server.

Table 8: FGTHunter results for OSCAR requirements.

R-ID	Links Retrieved	True Positives	True Positives	Not Found	GOLD SET	Precision	Recall	F1
1	8440	605	239	12	34	0,72	0,98	0,83
2	691	531	160	2	12	0,77	1	0,87
3	876	775	101	3	28	0,88	1	0,94
5	380	136	244	12	22	0,36	0,92	0,52
40	1777	853	924	8	102	0,48	0,99	0,65
44	1110	636	474	22	92	0,57	0,97	0,72
45	1216	1065	151	10	203	0,88	0,99	0,93
46	1273	317	956	31	43	0,25	0,91	0,39
47	262	236	26	0	34	0,9	1	0,95
49	6575	6012	563	0	106	0,91	1	0,96
56	3969	949	3020	52	52	0,24	0,95	0,38
63	142	28	114	7	9	0,2	0,8	0,32
70	1668	462	1206	2	2	0,28	1	0,43

Table 9: Summary of results for all healthcare systems analyzed in this study.

HIPAA statute	Description	Average F1 score
164.312(b)	"Implement hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use electronic protected health information."	iTrust:0,39 OSCAR:0,39 TAPAS:0,15
164.312(a)(1)	"Implement technical policies and procedures for electronic information systems that maintain electronic protected health information to allow access only to those persons or software programs that have been granted access rights as specified in §164.308(a)(4)."	iTrust:0,25 OSCAR:0,39 OPENMRS:0,66 TAPAS:0,39
164.312(a)(2)(i)	"Assign a unique name and/or number for identifying and tracking user identity."	iTrust:0,43 OSCAR:0,65 OPENMRS:0,77
164.312(d)	"Implement procedures to verify that a person or entity seeking access to electronic protected health information is the one claimed."	iTrust:0,60 OSCAR:0,87 TAPAS:0,46
164.312(a)(2)(iii)	"Implement electronic procedures that terminate an electronic session after a predetermined time of inactivity."	iTrust:0,30 OSCAR:0,32 TAPAS:0,46
164.308(a)(5)(ii)(C)	"Procedures for monitoring log-in at tempts and reporting discrepancies."	iTrust:0,48 OSCAR:0,73 OPENMRS:0,66
164.308(a)(1)(ii)(D)	"Implement procedures to regularly review records of information system activity, such as audit logs, access reports,and security incident tracking reports."	iTrust:0,40 OSCAR:0,39
164.312(e)(2)(i)	"Implement security measures to ensure that electronically transmitted el ctronic protected health information is not improperly modified without detection until disposed of."	iTrust:0,41
164.308(a)(5)(ii)(D)	"Procedures for creating, changing, and safeguarding passwords."	iTrust:0,53 OSCAR:0,69
164.312(a)(2)(iv)	"Implement a mechanism to encrypt and decrypt electronic protected health information."	OSCAR:0,83
164.308(a)(7)(ii)(A)	"Establish and implement procedures to create and maintain retrievable exact copies of electronic protected health information."	OSCAR:0,96 TAPAS:0,58
164.308(a)(4)(i)	"Establish and implement procedures to create and maintain retrievable exact copies of electronic protected health information."	TAPAS:0,32
164.312(c)(2)	"Implement electronic mechanisms to corroborate that electronic protected health information has not been altered or destroyed in an unauthorized manner."	TAPAS:0,63
164.312(c)(1)	"Implement policies and procedures to protect electronic protected health information from improper alteration or destruction."	TAPAS:0,51

Assigning a unique identifier to each entity of a system ensures the correct manipulation of the data and facilitates the control of the information integrity. According to our observations, the restrictions of non-repetition were always defined at the database level, that is, the traceability of the requirements associated with the Statute 164.312 (a)(2) (i) of HIPAA was carried out analyzing .sql files with thousands of code lines impacting negatively on the precision of our technique. However, according to the results, for OSCAR and OPENMRS systems the algorithm reaches its highest performance, probably because at the application level, the uniqueness of the identifiers for each entity was also validated.

Ensuring the correct implementation of access control mechanisms is a fundamental aspect in any system that manipulates critical information. Such control strategies range from verification of passwords and access codes to role management within the system. Considering the wide spectrum of artifacts in each system that may be related to HIPAA Statute 164.312 (d), it is natural that the accuracy tend to be very high; however in TAPAS that value is low, maybe because the access control in this application is not clearly defined at the application level, i.e., there are no login forms or access restrictions, and externally they must control the access to the information by applying restrictions policies.

Closing the session after a period of inactivity is a measure of additional protection that usually exists to avoid improper manipulation of the data. The session time limit for a given user is often specified in a particular line of code within the application, either through a property file, a database record or a global variable. Considering the quantity of artifacts analyzed and the reduced number of code lines in which the requirements related to statute 164.312 (a)(2)(iii) of HIPAA are implemented, for iTrust and OSCAR the level of precision was very low. According to Table 9, TAPAS is the exception probably because in many parts of the code the mechanisms for closing inactive sessions are repeated, once a particular operation has started.

As a security measure, many information systems keep a record of unsuccessful attempts prior to login for a particular service. Depending on the number of failed attempts, an account is blocked for a period of time or indefinitely until an administrator decides to unblock it. Table 9 summarizes the traceability results for the requirements related to HIPAA Statute 164.308 (a)(5)(ii)(C). The case with less precision was iTrust, probably because there are very few code lines where the control of failed attempts is made.

When a user of the system views or edits the information of a particular patient, the audit information should be enough to know the detail of the modifications made to critical data, as is established by the statute 164.308 (a)(1)(ii)(D) of HIPAA. Table 9 shows the average of the harmonic mean for the systems that implemented control mechanisms of visualization for audited data. In general, the performance of the traceability algorithm was acceptable, above 0.4.

As a measure of securing information in the event of a disaster, data security should be periodically generated from the information contained in the database in order to be effectively restored. The HIPAA statute 164.308 (a)(7)(ii)(A) refers to this topic, as shown in Table 9. OSCAR was the system in which our algorithm reached a higher precision, while TAPAS was the opposite case. This particular behavior can be proved when we compare the source code artifacts from both systems that were involved in the implementation of such a statute. From our findings in the preliminary analysis, we observe that OSCAR has more lines of code associated with data backups and database restoration than TAPAS.

7 Discussion

This paper presents a novel approach to a fine-grained traceability problem that is common in the healthcare systems context. Our findings indicate that the precision of FGTHunter is, in general, more than 0,4 for each HIPAA requirement (Average F1 score).

The regulations related to audit control standards and session expiration in the implementation of healthcare systems were the hardest to trace to source code statements. In those cases, our approach achieved the lowest precision (about 0,35 F1 score mean, for each system). Very few lines and source code structures related to these requirements were successfully mapped by our algorithm for each system. This can be explained by the few places within the source code where these requirements were implemented, increasing in that way the recall of our technique and decreasing the precision. With respect to the other statutes defined in the privacy and security rule of HIPAA, the performance of our algorithm was acceptable (over 0,4 F1 score mean) and in some cases excellent (over 0,8 F1 score mean).

The precision of the proposed technique is highly dependent on good design practices and the adherence to conventions for naming entities according to the problem domain (i.e., quality of source code); this is a key factor that should be considered in future research to increase the accuracy of any text-based technique.

High level requirement extraction from available resources following a linguistic pattern, may compromise the correctness and completeness of query artifacts in our algorithm, since not always documentation written in natural language is good enough to describe what actually is implemented in source code. To reduce the impact of this issue, a meta-language could be proposed to denote high level requirements prior the definition of query artifacts. Although it is not a mandatory task in the execution of our technique, when starting from requirements that are defined from a specific context (i.e. system domain) the precision of the technique would improve.

8 Conclusions

In specific contexts, as is the case of medical health systems, strict rules are required to comply with the secure processing of personal data. For the effective verification of compliance with these rules, it is essential to establish links between the formal text of the regulations and the lines of code in which they are implemented. In this paper we describe an algorithm, based on heuristics, static code analysis and information retrieval techniques, that supports the recovering such traceability links.

One of the biggest difficulties encountered is that the regulations expressed in legal documents are usually written in fairly technical language and detached from a particular context. To overcome this problem, we manually identified and extracted the documentation of requirements that is related to some HIPAA statute, within the taxonomy that we previously defined.

Even though the static code analysis techniques have played a determining role in the creation of automatic techniques for the recovery of traceability links, our approach properly combines those techniques with heuristics and information retrieval methods. We believe that these combined methods can improve the results of the individually used techniques, and could reach an adequate level of precision to automatically retrieve the links between formal regulations and source code.

Although it is quite difficult to reach a suitable degree of precision in a traceability technique based on an information retrieval algorithms, without the assistance of a person that helps to delimit the list of ranked candidate links, our approach would undoubtedly ease the work of those people involved in the process of certifying that specified system complies with rules and standards at a source code level. In any case, further empirical validations are required to assess the effectiveness of our approach in real contexts.

9 Future Work

Our work can be expanded in many directions. First, as the technique is designed for software systems written in Java EE, future work could focus on expanding the current traceability algorithm to cover other languages and systems development environments. For this purpose, one alternative would be to keep the original approach of our technique and build syntactic analyzers for each new programming language in such a way that the source code slicer and code analysis algorithms can be applied as they are.

Second, another evaluation of our technique can be made with other regulations other than HIPAA. To do this it would be essential to analyze the structure of such regulations and identify appropriate software systems whose documentation and code are available to carry out the entire process outlined in this paper.

Lastly and most importantly, empirical evaluations of our approach are more than desirable. They are required to assess the impact of our technique on the work of real users and regulation reviewers as well as its performance, usability, and compliance with real needs.

Authors' Information

– **Alejandro Velasco** is a software engineering practitioner currently working in Scotiabank Colpatria. He received his Engineering and Master's degree in Systems and Computing Engineering from Universidad Nacional de Colombia (Bogotá, Colombia). His research interest includes Software Engineering, Chaos Engineering and Machine Learning.

– **Jairo Aponte** is an associate professor in the Department of Computing Systems and Industrial Engineering at Universidad Nacional de Colombia. He received his Engineering and a Master's degree in Computing Systems Engineering from Universidad de los Andes (Bogotá, Colombia), and a Ph.D. in Engineering from Universidad Nacional de Colombia (Bogotá). His research interests include agile software processes, program comprehension, and software evolution.

Authors' Contributions

– **Alejandro Velasco** Conducted the analysis of HIPAA statutes and proposed the derived taxonomy, built the dataset and goldset from open source heatlh care systems as presented in section 3, proposed the set of heuristics based on observations as described in section 4, designed the fine grained traceability algorithm presented in section 5, implemented the proposed technique by creating FGTHunter, tested the proposed approach and described the results as stated in section 6; and finally, conducted the analysis of results and discussions presented in sections 7, 8, and 9.

– **Jairo Aponte** Validated the constructed dataset presented in section 3, validated and improved the proposed heuristics presented in section 4, participated in the design and improved the performance of the proposed algorithm by suggesting changes based on previous research, improved the analysis of results described in section 6; and finally, participated in the writing of all sections.

Competing Interests

The authors declare that they have no competing interests.

Funding

No funding was received for this project.

Availability of Data and Material

1° Datasets and Goldsets for each of the systems described in this work, as well as HIPAA artifacts can be found in the next repository for public access: `https://bitbucket.org/savelascod/fgt-attachments`

2° Source code files of FGTHunter can be found in the next repository: `https://bitbucket.org/savelascod/fgthunter`

References

[1] O. C. Z. Gotel and C. W. Finkelstein, "An analysis of the requirements traceability problem," in *Proceedings of IEEE International Conference on Requirements Engineering*, pp. 94–101, 1994.

[2] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[3] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 58–93, 2001.

[4] V. Yadav, R. K. Joshi, and S. Ling, "Evolution traceability roadmap for business processes," in *Proceedings of the 12th Innovations on Software Engineering Conference (Formerly Known as India Software Engineering Conference)*, ISEC'19, (New York, NY, USA), Association for Computing Machinery, 2019.

[5] O. f. C. Rights (OCR), "HIPAA Compliance and Enforcement," May 2008. Library Catalog: www.hhs.gov.

[6] T. D. Breaux, M. W. Vail, and A. I. Anton, "Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations," in *14th IEEE International Requirements Engineering Conference (RE'06)*, pp. 49–58, 2006.

[7] S. Avancha, A. Baxi, and D. Kotz, "Privacy in mobile technology for personal healthcare," *ACM Comput. Surv.*, vol. 45, Dec. 2012.

[8] W. Shen, C. Lin, and A. Marcus, "Using traceability links to identifying potentially erroneous artifacts during regulatory reviews," in *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pp. 19–22, 2013.

[9] A. Velasco and J. H. Aponte Melo, "Recovering Fine Grained Traceability Links Between Software Mandatory Constraints and Source Code," in *Applied Informatics* (H. Florez, M. Leon, J. M. Diaz-Nafria, and S. Belli, eds.), (Cham), pp. 517–532, Springer International Publishing, 2019.

[10] "Health Insurance Portability and Accountability Act of 1996 (HIPAA) | CDC," Feb. 2019. Library Catalog: www.cdc.gov.

[11] T. Breaux and A. Antón, "Analyzing regulatory rules for privacy and security requirements," *IEEE Transactions on Software Engineering*, vol. 34, no. 1, pp. 5–20, 2008.

[12] N. Kiyavitskaya, N. Zeni, T. D. Breaux, A. I. Antón, J. R. Cordy, L. Mich, and J. Mylopoulos, "Automating the Extraction of Rights and Obligations for Regulatory Compliance," in *Conceptual Modeling - ER 2008* (Q. Li, S. Spaccapietra, E. Yu, and A. Olivé, eds.), (Berlin, Heidelberg), pp. 154–168, Springer Berlin Heidelberg, 2008.

[13] T. D. Breaux and A. I. Antón, "A systematic method for acquiring regulatory requirements : A frame-based approach," 2007.

[14] N. Zeni, L. Mich, J. Mylopoulos, and J. R. Cordy, "Applying gaiust for extracting requirements from legal documents," in *2013 6th International Workshop on Requirements Engineering and Law (RELAW)*, pp. 65–68, 2013.

[15] T. Alshugran and J. Dichter, "Extracting and modeling the privacy requirements from hipaa for healthcare applications," in *IEEE Long Island Systems, Applications and Technology (LISAT) Conference 2014*, pp. 1–5, 2014.

[16] J. C. Maxwell and A. I. Antón, "Checking existing requirements for compliance with law using a production rule model," in *2009 Second International Workshop on Requirements Engineering and Law*, pp. 1–6, 2009.

[17] J. Huang, O. Gotel, and A. Zisman, eds., *Software and Systems Traceability*. London: Springer-Verlag, 2012.

[18] F. Fasano, "Fine-grained management of software artefacts," in *2007 IEEE International Conference on Software Maintenance*, (Los Alamitos, CA, USA), IEEE Computer Society, oct 2007.

[19] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, (USA), p. 125–135, IEEE Computer Society, 2003.

[20] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Tracing object-oriented code into functional requirements," in *Proceedings IWPC 2000. 8th International Workshop on Program Comprehension*, pp. 79–86, 2000.

[21] G. Antoniol, G. Canfora, A. De Lucia, and E. Merlo, "Recovering code to documentation links in oo systems," in *Sixth Working Conference on Reverse Engineering (Cat. No.PR00303)*, pp. 136–144, 1999.

[22] Antoniol, Canfora, Casazza, and De Lucia, "Information retrieval models for recovering traceability links between code and documentation," in *Proceedings 2000 International Conference on Software Maintenance*, pp. 40–49, 2000.

[23] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "User reviews matter! tracking crowdsourced reviews to support evolution of successful apps," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 291–300, 2015.

[24] A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Improving ir-based traceability recovery using smoothing filters," in *2011 IEEE 19th International Conference on Program Comprehension*, pp. 21–30, 2011.

[25] D. Diaz, G. Bavota, A. Marcus, R. Oliveto, S. Takahashi, and A. De Lucia, "Using code ownership to improve ir-based traceability link recovery," in *2013 21st International Conference on Program Comprehension (ICPC)*, pp. 123–132, 2013.

[26] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, vol. 41, no. 6, pp. 391–407, 1990.

[27] S. T. Dumais, "Improving the retrieval of information from external sources," *Behavior Research Methods, Instruments, & Computers*, vol. 23, pp. 229–236, June 1991.

[28] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *J. Softw. Evol. Process.*, vol. 25, pp. 53–95, 2013.

[29] A. Qusef, G. Bavota, R. Oliveto, A. D. Lucia, and D. Binkley, "Recovering test-to-code traceability using slicing and textual analysis," *Journal of Systems and Software*, vol. 88, pp. 147 – 168, 2014.

[30] B. Dagenais and M. P. Robillard, "Recovering traceability links between an api and its learning resources," in *2012 34th International Conference on Software Engineering (ICSE)*, pp. 47–57, 2012.

[31] B. Sharif and J. I. Maletic, "Using fine-grained differencing to evolve traceability links," in *In GCT'07*, pp. 76–81, ACM, 2007.

[32] W. E. Wong, S. S. Gokhale, J. R. Horgan, and K. S. Trivedi, "Locating program features using execution slices," in *Proceedings 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99 (Cat. No.PR00122)*, pp. 194–203, 1999.

[33] "iTrust | Medical Free/Libre and Open Source Software."

[34] "start [iTrust]."

[35] "Home - Documentation - OpenMRS Wiki."

[36] "OSCAR EMR User's Manual — Site."

[37] "TAPAS Home."

[38] M. Cohn, *User Stories Applied: For Agile Software Development*. Boston: Addison-Wesley Professional, edición: 1 ed., Mar. 2004.

[39] J. Garzas, *Object-Oriented Design Knowledge: Principles, Heuristics and Best Practices*. Hershey, PA: Idea Group Publishing, July 2006.

[40] P. Azimi and P. Daneshvar, "An Efficient Heuristic Algorithm for the Traveling Salesman Problem," in *Advanced Manufacturing and Sustainable Logistics* (W. Dangelmaier, A. Blecken, R. Delius, and S. Klöpfer, eds.), (Berlin, Heidelberg), pp. 384–395, Springer Berlin Heidelberg, 2010.

[41] Z. W. Geem, J. H. Kim, and G. Loganathan, "A New Heuristic Optimization Algorithm: Harmony Search," *SIMULATION*, vol. 76, pp. 60–68, Feb. 2001. Publisher: SAGE Publications Ltd STM.

[42] H. Agrawal, J. R. Horgan, S. London, and W. E. Wong, "Fault localization using execution slices and dataflow tests," in *Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95*, pp. 143–151, 1995.

[43] "Amazon.com: Head First Object-Oriented Analysis and Design (0636920008675): Brett D. McLaughlin, Gary Pollice, Dave West: Books."

[44] S. M. Sutton, "Aspect-Oriented Software Development and Software Process," in *Unifying the Software Process Spectrum* (M. Li, B. Boehm, and L. J. Osterweil, eds.), (Berlin, Heidelberg), pp. 177–191, Springer Berlin Heidelberg, 2006.

[45] B. Fluri, M. Wursch, and H. C. Gall, "Do code and comments co-evolve? on the relation between source code and comment changes," in *14th Working Conference on Reverse Engineering (WCRE 2007)*, pp. 70–79, 2007.

[46] B. Fluri, M. Würsch, E. Giger, and H. C. Gall, "Analyzing the co-evolution of comments and source code," *Software Quality Journal*, vol. 17, pp. 367–394, Dec. 2009.

[47] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*. Upper Saddle River, N.J: Pearson, edición: 1st ed., Oct. 2002.

[48] D. Binkley, M. Davis, D. Lawrie, J. I. Maletic, C. Morrell, and B. Sharif, "The impact of identifier style on effort and comprehension," *Empirical Software Engineering*, vol. 18, pp. 219–276, Apr. 2013.

[49] M. Ohba and K. Gondow, "Toward mining "concept keywords" from identifiers in large software projects," in *Proceedings of the 2005 international workshop on Mining software repositories*, MSR '05, (St. Louis, Missouri), pp. 1–5, Association for Computing Machinery, May 2005.

[50] Y. Goldberg and O. Levy, "word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method," 2014.

[51] S. Arlt, A. Podelski, and M. Wehrle, "Reducing gui test suites via program slicing," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, (New York, NY, USA), p. 270–281, Association for Computing Machinery, 2014.

[52] K. B. Gallagher and J. R. Lyle, "Using program slicing in software maintenance," *IEEE Transactions on Software Engineering*, vol. 17, no. 8, pp. 751–761, 1991.

[53] I. Mastroeni and D. Zanardini, "Abstract program slicing: An abstract interpretation-based approach to program slicing," *ACM Trans. Comput. Logic*, vol. 18, Feb. 2017.

[54] M. Weiser, "Program slicing," in *Proceedings of the 5th International Conference on Software Engineering*, ICSE '81, p. 439–449, IEEE Press, 1981.