Balasubramanian and Irfan Ahmed, Global J Technol Optim 2015, 6:2
DOI: 10.4172/2229-8711.1000178

Research Article Open Access

Software Dependency Estimation in the Code Repositories for the Requirement Evolution

Karthikeyan Balasubramanian* and Irfan Ahmed MS

[1]*Bharathiar University, Tamilnadu, India*

Abstract

Dependency is the only means to ensure that the source code of a system is consistent with its requirements. During software maintenance and evolution, requirement dependency links become obsolete because dependency model is been not trained properly to updating them. Yet, recovering these dependency links later is a daunting and costly task for building the model for unsupervised enhancements. Consequently, the literature has proposed methods, techniques, and tools to recover these dependency links semi-automatically or automatically. Among the proposed techniques, the literature showed that information retrieval (IR) techniques can automatically recover traceability links between free-text requirements and source code through classification techniques to the Software repositories. However, IR techniques lack accuracy (precision and recall) in terms of Text and concept based mining also leads to code sense disambiguation. In this paper, we show that Semantic mining of software repositories and combining mined results with IR can improve the accuracy (precision and recall) of IR techniques. We apply Dependency Estimation on to compare the accuracy of its dependency links with those recovered using state-of-the-art IR techniques from Vector Space model and Concept based mining. We thus show that mining software repositories and combining the mined data with existing results from IR techniques improves the precision and recall of requirement dependency links.

Keywords: Dependency; Source code; Repository; Code mining; Requirement traceability

Introduction

The amount of information that is accessible to an fresh engineers or even to experienced staff seems to be today is mind-boggling. While a few centuries ago people were struggling to access information, today many are struggling to eliminate the irrelevant information that reaches them through various channels like news feeds and Database servers. Since concepts are abstract entities, representing them is another problem. In part of this research, we establish a Data Warehouse or Code Repository as knowledge base. It contains lot of codes related to different concepts in different databases used to many domains (set of codes representing the same concept) and their relationships with other codes collection. In this work, we present two alternate ways for requirement dependency, one is based on identifying relation to the requirement through Principle component analysis process and the other one is based on representing concepts through neighboring words using domain specific corpus.. The organization of the paper is as follows: Section 2 gives detailed related work about concept-based indexing such as concept representation methods, several Semantic based techniques and building knowledge repositories. Section 3 presents our approach to achieve Semantic -based access using evidence combination of several WSD techniques and query expansion with related concepts. Section 4 describes the evaluation process of our approach. Section 5 discusses the results of WSD and concept-based indexing experiments [1].

Related Work

A literature survey shows that ontologies have been employed to achieve better precision and recall in text retrieval systems. Query expansion has improved the effectiveness of ranked retrieval by automatically adding additional terms to a query. Guarino et al. [2] has attempted to perform query expansion through the use of semantically related terms and the use of conceptual similarity measures to find document similarity.

Concept representation approaches

I mention two approaches here due to space constraints:

Principle component analysis of codebase: The concepts are represented by concept nodes in the graph. A concept node is a component structure that has slots that contain information about that concept such as its triggering word, patterns for extracting concept from text as in Riloff and Lehnert [3].

Conceptual graphs: The conceptual graph is a technique that is developed by Sowa [4,5], to represent knowledge. A conceptual graph g is a bipartite graph that has two kinds of nodes called, concepts and conceptual relations [6-8]. This graphical notation of conceptual relations is for human readability. This notation can be transformed into Knowledge Interchange Format (KIF) or predicate calculus notation for processing them in computers [9-11].

Latent semantic analysis

Latent Semantic Analysis (LSA) is proposed in order to overcome the polysemy and synonym problems of traditional keyword based retrieval [12]. The main goal of this technique is to reveal the underlying semantic structure of the documents by representing them in high dimensional space. LSA uses singular value decomposition in order to reduce the number of dimensions in the term-by-documents matrix and tested the performance of latent semantic indexing in two test corpus: MED and CISI. The results show that LSA improves average

***Corresponding author:** Karthikeyan Balasubramanian, Bharathiar University, Tamilnadu, India, Tel: 020-30213250; E-mail: bkarthikeya@gmail.com

Received May 07, 2014; **Accepted** February 27, 2015; **Published** March 07, 2015

Citation: Balasubramanian K, Irfan Ahmed MS (2015) Software Dependency Estimation in the Code Repositories for the Requirement Evolution. Global J Technol Optim 6: 178. doi:10.4172/2229-8711.1000178

precision of traditional term matching by 13% in MED collection. However, they couldn't achieve any improvement in CISI experiments over classical approach [13].

Proposed System

The new framework based Software Dependency estimation which represents a Semantic mining for extracting the code from the code repository. Semantic mining is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. By proposing a semantic network representation as in directed labelled graph [14]. It is a simplified conceptual graph. I represent conceptual relationships as edges between nodes in the graph rather than representing them as nodes like in conceptual graphs (Figure 1).

WSD based on different domains

Concept Identification: Codebase is been used to associate with data warehouse or Repository to sense distinctions as predefined set of code [15]. Our technique is based on evidence combination of supervised and unsupervised methods using semantic mechanism [16]. Secondly, technique has to be presented to identify the concepts in a domain with its related words.

The relatedness of two concepts is identified using similarity of their content neighbouring concept with description using Latent Semantic Analysis technique [17].

Contextual weights of semantics related abstract query: This is an unsupervised Semantic approach that is similar to the method presented in [14]. It uses semantic relationships among codes (concepts) in Codebase for different domain. The method has the following steps:

Evidence combination of WSD techniques for extraction: We apply different techniques to combine result of Semantic mining methods such as using uncertainty values, Count based measures based with uncertainty and using sense rankings. Each will be explained below:

Evidence combination based on dependency with uncertainty: In this case, we use the simple voting principle to do the evidence combination in the first phase. That means each WSD method gives its choice for the sense of the Model or Code. Then the sense that has maximum vote is chosen as the sense of the code. If there is a tie in the first phase, then we compare the uncertainty values of WSD sources and choose the sense with minimum uncertainty value (Figure 2).

Evidence combination based on sense rankings: This evidence combination technique considers the sense rankings given by each Semantic Mining method. Instead of using only first senses given by each method like in the Dependency case, this technique takes other probable senses into account.

Concept-based Search with Knowledge Repository

Concept-based IR as-based indexing

We used Semantic based mining model for establishing the software dependency to process dependability to the drift of software evolution. Since system doesn't do disambiguation of all content codes, system been modelled using word based indexing for the terms through convergence of the context to predict the dependency link to derive the result (Figure 3).

For Example Simple Mobile Phonebook is a small application running on mobile devices. A user can edit the details of his contacts. A contact has a name, a phone number and an address. For every contact there can be a voice file recorded as an alternative mean to identify the

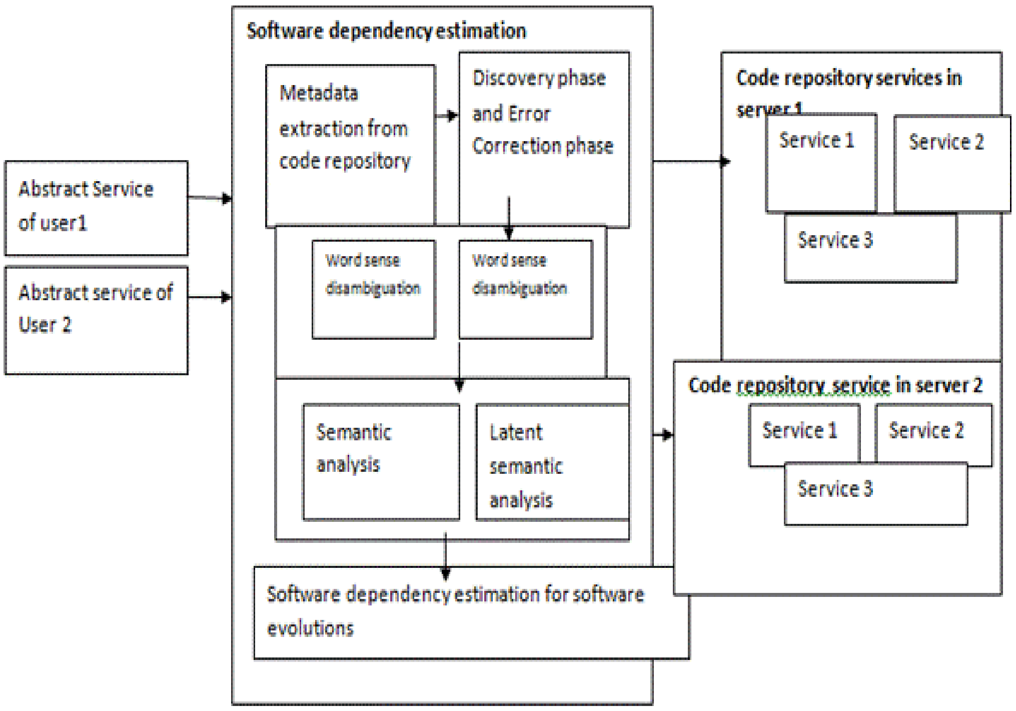
Software dependency estimation
Metadata extraction from code repository
Discovery phase and Error Correction phase
Word sense disambiguation
Word sense disambiguation
Semantic analysis
Latent semantic analysis
Software dependency estimation for software evolutions
Abstract Service of user1
Abstract service of User 2
Code repository services in server 1
Service 1
Service 2
Service 3
Code repository service in server 2
Service 1
Service 2
Service 3

Figure 1: Representation of Software Dependency modelling to the Evolution.

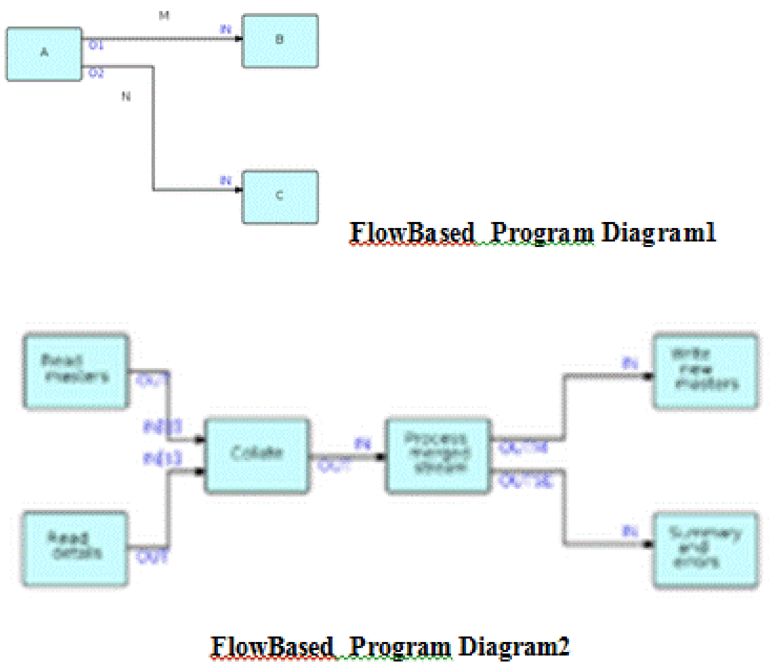

Figure 2: FlowBased Programs 1 and 2.

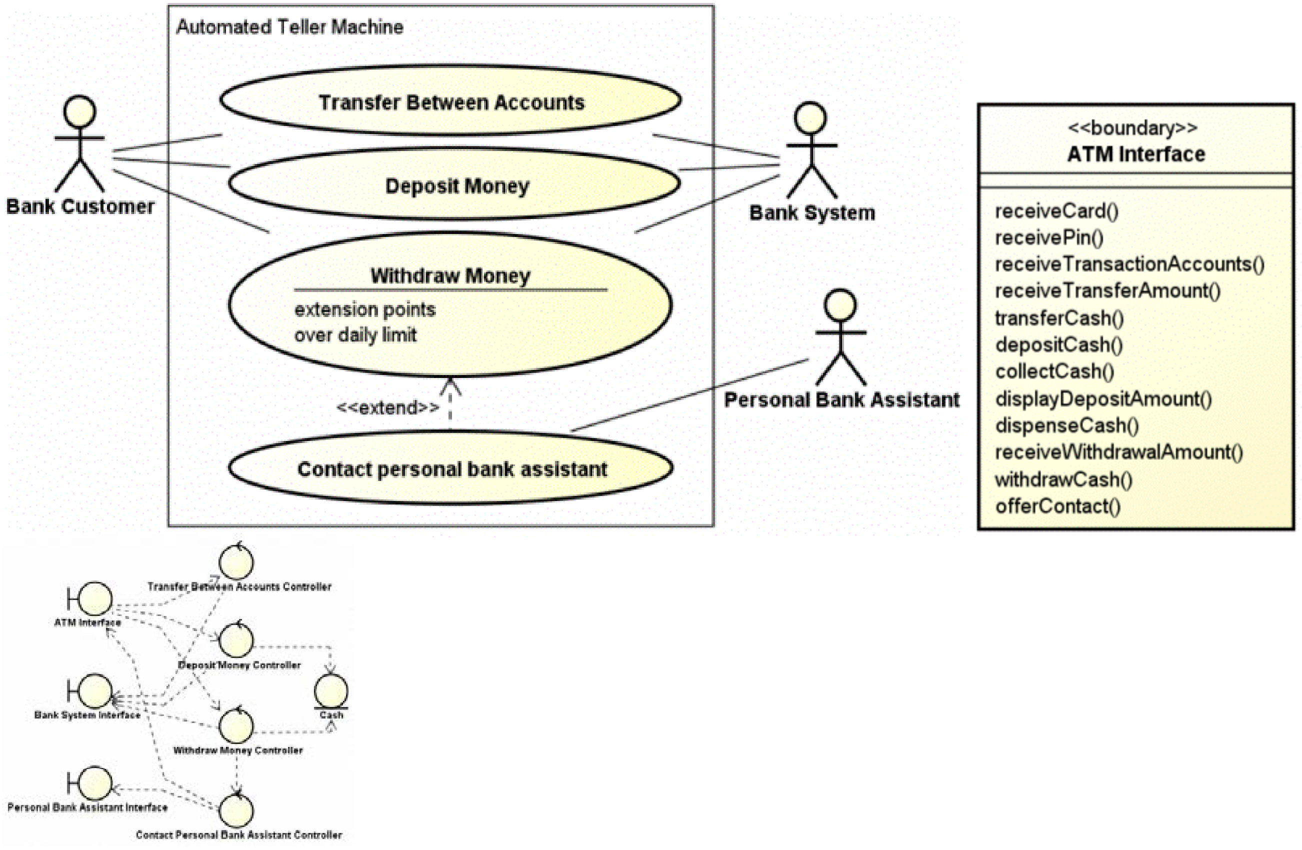

Figure 3: Code Dependency for ATM.

contact (Figure 4). A user can make a call by entering the name of a contact or by voice lookup. To call a contact identified by voice file, he presses a special button on the screen; the device records his voice and compares it to the voice files for the contacts. If none of the voice files matches with a high enough score a list with the best matches will be presented from which the user can select the contact. After successful selection of a contact the call is initiated.

Concept-based IR using latent semantic analysis and query expansion

Considering the difficulty of WSD techniques and state-of art results as around 70% precision and system decided to utilize concept based mining code retrieval with less accuracy. To assume that only a single sense of a word is relevant to a domain. There are some senses of words that are domain independent in the sense that it can be used in many domains but since they don't contribute much to the general meaning of the document as domain specific words do, we don't consider them at this point. A corpus input and produces word-word similarity measures. It constructs a model that is called CODE-SPACE" that represents each word using the content bearing co-occurring words. Each content-bearing word represents a dimension in workspace or Data warehouse. A variant of Latent Semantic Analysis is used to reduce the number of dimensions in data vectors. Then it computes the similarity of two terms based on cosine similarity of these co-occurrence vectors. At the end, it gives related terms of a word with normalized similarity measures. So we can use these related concepts for query expansion in order to retrieve documents. Addition of related concepts to the query eliminates the documents that use a different sense of the concept and retrieves codes that do not contain the main concept but related concepts in it.

Automated code review software checks source code for compliance with a predefined set of rules or best practices. The use of analytical methods to inspect and review source code to detect bugs has been a standard development practice. This process can be accomplished both manually and in an automated fashion. With automation, software tools provide assistance with the code review and inspection process. The review program or tool typically displays a list of warnings (violations of programming standards). A review program can also provide an automated or a programmer-assisted way to correct the issues found.

Experimental Results

Dataset

Dataset used in this section contains all metadata about all software

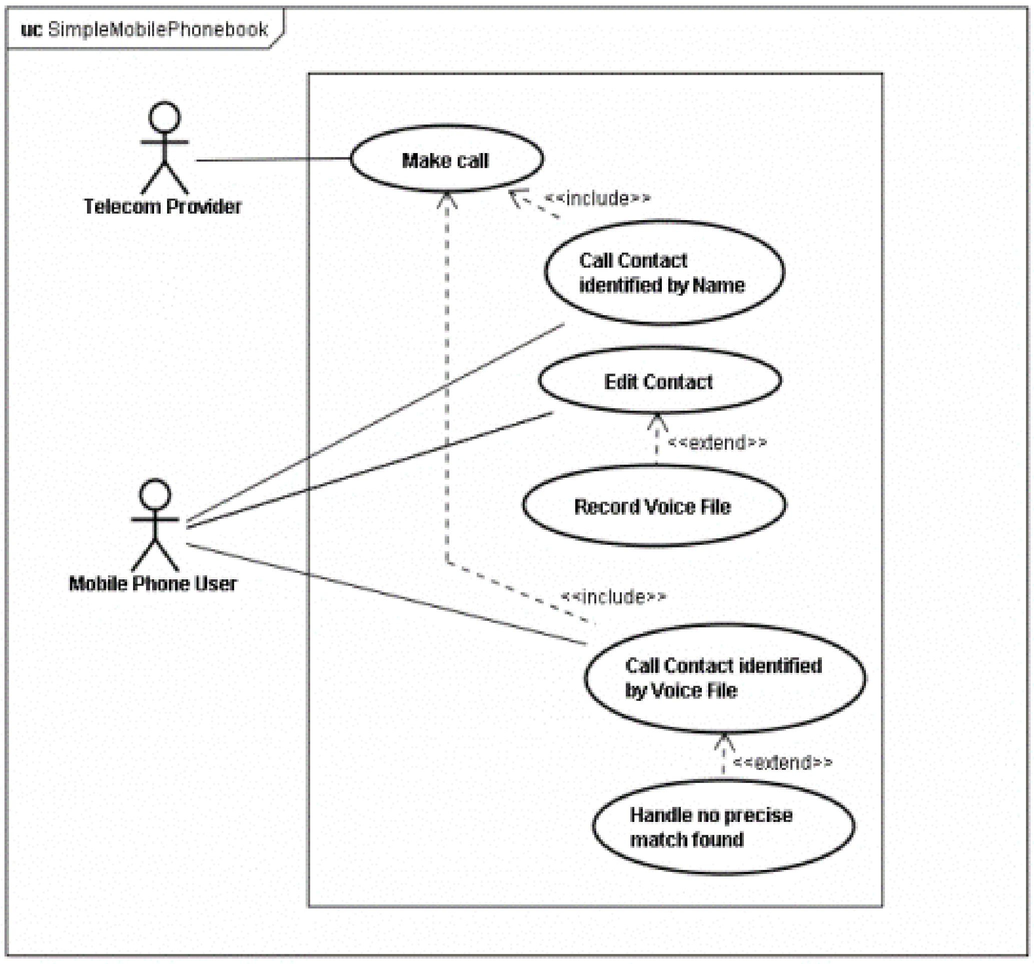

Figure 4: Use Case Diagram for the Simple Mobile Phonebook.

codes. Codebase is been used to associate with data warehouse or Repository to sense distinctions as predefined set of code with huge concept and process evolution.

Software failure proneness through concept based mining in code retrieval

Software failure proneness in concept based mining to the code base updating is challenging due to the software dependency links mining and link management during the software evolution. Considering the difficulty of Concept based techniques and state-of art results as around 70% precision, in this work, by identify concepts in a several codes. We assume that only a single sense of a word is relevant to a domain. There are some senses of words that are domain independent in the sense that it can be used in many domains but since they don't contribute much to the general meaning of the document as domain specific words don't consider them at this point been utilized by training, validation and testing data for classification of results using historical prediction models identify the results set estimation efficiently and effectively in large codebase. The performances of the classification are experimented and presented in terms of relative speed, computational time as properties measure of performance using the large codebase (Table 1).

Query frequency estimation and temporal probability estimation

The temporal prediction states observed from the large codebase are as follows: supervised code, unsupervised code and semi-supervised code (Table 2).

Feature extraction through user query modeling: Feature Extraction is employed in large dataset with code drifting and information retrieval with estimating various factors in the query analysis to the large codebase.

Feature extraction: The data in the big data is evolved with several feature classification with novel features estimation in each sample such as, y1, y2, y3, y4 and y5, are extracted by the equation as follows:

$$\text{yk} = \frac{c^k}{\max\limits_{i=1}^{5}\left(c^i\right)} \qquad (9)$$

Where k=1, 2,. . ., 5,

ck – Absolute feature data per one sample.

(1) The absolute information is calculated for different samples given by,

$$\text{Y6=log10}\left(\max\limits_{m=1}^{5} c^m\right) \qquad (26)$$

Result Analysis

The proposed framework is implemented and tested using different types of codebase using domain specific modeling and multi correlation estimation using Word sense disambiguation. An extensive experimental study was conducted to evaluate the efficiency and effectiveness of the proposed methodology on various parameters of benchmark instances and the prediction states of the dependency link.

The following parameters are utilized to estimate the performance of the big data classification and prediction of data for user queries (Figure 5).

Minimize average diameter of clusters results

This factor estimates the performance of proposed framework in the classifying the data with concept drift. Proposed framework by latent analysis and sematic miming proves the accuracy results set with precision and recall in the cluster achieved for software dependency link establishment for evolution and updating in the software

Maximum likelihood of Software process evolution monitoring through latent Semantic analysis

It is process to determine the latent semantic analysis of maximum likelihood of the software updating and software under enhancement. The software dependency is estimating the parameters of a statistical model with semantic mining. When applied to a code base and given a statistical model, maximum-likelihood estimation provides estimates for the model's parameters. We have proved the performance of system in WSN through semantic mining of abstract query based on the several factors included in the in the domain and knowledge base to determine the performance factors with better results.

Easy Adaptability in Programming

If application software is designed in such a way that its programmers are able to easily adapt the interface layer that deals with the OS, window manager or desktop environment to new or changing standards, then the programmers would only have to monitor notifications from the environment creators or component library designers and quickly adjust their software with updates for their users, all with minimal effort and a lack of costly and time-consuming redesign. This method would encourage programmers to pressure those upon whom they depend to maintain a reasonable notification process that is not onerous to anyone involved.

Software appliances

Parameters	Notations used	Values
Learning rate	Λ	0.01
Scaling factor	Σ	1

Table 1: Parameters of classification and Prediction of code analysis.

Parameters	Notations used	Values
Number of iteration	I	15000
Order of the polynomial	Order	3
Scaling factor	Σ	1

Table 2: Performance parameters of estimating for code Extraction and its analysis.

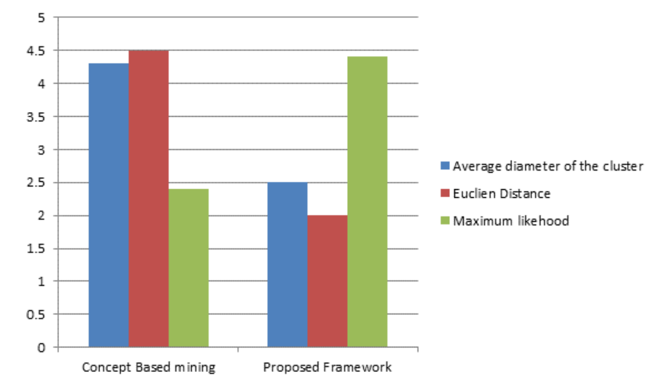

Figure 5: Estimation of the proposed framework against concept based mining.

Another approach to avoiding dependency issues is to deploy applications as a software appliance. A software appliance encapsulates dependencies in a pre-integrated self-contained unit such that users no longer have to worry about resolving software dependencies. Instead the burden is shifted to developers of the software appliance.

Portable applications

An application (or version of an existing conventional application) that is completely self-contained requires nothing to be altered. It is coded to have all necessary components included, or is designed to keep all necessary files within its own directory, and will not create a dependency problem. These are often able to run independently of the system to which they are connected. Applications in RISC OS and the ROX Desktop for Linux use application directories, which work in much the same way: programs and their dependencies are self-contained in their own directories (folders) [8]. This method of distribution has also proven useful when porting applications designed for Unix-like platforms to Windows, the most noticeable drawback being multiple installations of the same shared library. For example, Windows installers for gedit, GIMP, and XChat all include identical copies of the GTK+ toolkit, which these programs use to render widgets.8.5, Optimizing with the Maven Dependency Plugin.

On larger projects, additional dependencies often tend to creep into a POM as the numbers of dependencies grow. As dependencies change, you are often left with dependencies that are not being used, and just as often, you may forget to declare explicit dependencies for libraries you require. Because Maven 2.x includes transitive dependencies in the compile scope, your project may compile properly but fail to run in production. Consider a case where a project uses classes from a widely used project such as Jakarta Commons BeanUtils. Instead of declaring an explicit dependency on BeanUtils, your project simply relies on a project like Hibernate that references BeanUtils as a transitive dependency. Your project may compile successfully and run just fine, but if you upgrade to a new version of Hibernate that doesn't depend on BeanUtils, you'll start to get compile and runtime errors, and it won't be immediately obvious why your project stopped compiling. Also, because you haven't explicitly listed a dependency version, Maven cannot resolve any version conflicts that may arise. A good rule of thumb in Maven is to always declare explicit dependencies for classes referenced in your code. If you are going to be importing Commons BeanUtils classes, you should also be declaring a direct dependency on Commons BeanUtils. Fortunately, via byte code analysis, the Maven Dependency plugin is able to assist you in uncovering direct references to dependencies. Using the updated POMs we previously optimized, let's look to see if any errors pop up:

```
$ mvndependency:analyze
[INFO] Scanning for projects...
[INFO] Reactor builds order:
[INFO]   Chapter 8 Simple Parent Project
[INFO]   Chapter 8 Simple Object Model
[INFO]   Chapter 8 Simple Weather API
[INFO]   Chapter 8 Simple Persistence API
[INFO]   Chapter 8 Simple Command Line Tool
[INFO]   Chapter 8 Simple Web Application
[INFO]   Chapter 8 Parent Project
[INFO] Searching repository for plugin with prefix: 'dependency'.
[INFO] ------------------------------------------------------------
[INFO] Building Chapter 8 Simple Object Model
[INFO]task-segment: [dependency: analyze]
[INFO] ------------------------------------------------------------
[INFO] Preparing dependency: analyze
[INFO] [resources: resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler: compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources: testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [Compiler: testCompile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [dependency: analyze]
[WARNING] Used undeclared dependencies found:
[WARNING] javax.persistence: persistence-api: jar:1.0:compile
[WARNING] Unused declared dependencies found:
[WARNING] org. hibernate: hibernate-annotations :jar :3.3.0.ga : compile
[WARNING]org.hibernate:hibernate:jar:3.2.5.ga:compile
[WARNING]junit:junit:jar:3.8.1:test
[INFO] ------------------------------------------------------------
[INFO] Building Chapter 8 Simple Web Application
[INFO]task-segment: [dependency:analyze]
[INFO] ------------------------------------------------------------
[INFO] Preparing dependency:analyze
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] No sources to compile
[INFO] [dependency:analyze]
[WARNING] Used undeclared dependencies found:
[WARNING]org.sonatype.mavenbook.optimize:simple-model :
jar: 1.0:compile
[WARNING] Unused declared dependencies found:
[WARNING]org.apache.velocity:velocity:jar:1.5:compile
```

```
[WARNING]javax.servlet:jstl:jar:1.1.2:compile
[WARNING]taglibs:standard:jar:1.1.2:compile
[WARNING]junit:junit:jar:3.8.1:test
```

In the truncated output just shown, you can see the output of the dependency:analyze goal. This goal analyzes the project to see whether there are any indirect dependencies, or dependencies that are being referenced but are not directly declared. In the simple-model project, the Dependency plugin indicates a "used undeclared dependency" on javax.persistence:persistence-api. To investigate further, go to thesimple-model directory and run the dependency:tree goal, which will list all of the project's direct and transitive dependencies:

```
$ mvndependency:tree
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'dependency'.
[INFO] ------------------------------------------------------------------
[INFO] Building Chapter 8 Simple Object Model
[INFO]task-segment: [dependency:tree]
[INFO] ------------------------------------------------------------------
[INFO] [dependency:tree]
[INFO] org.sonatype.mavenbook.optimize:simple-model:jar:1.0
[INFO]+- org.hibernate:hibernate-annotations:jar:3.3.0.ga:compile
[INFO] | \- javax.persistence:persistence-api:jar:1.0:compile
[INFO] +- org.hibernate:hibernate:jar:3.2.5.ga:compile
[INFO] | +- net.sf.ehcache:ehcache:jar:1.2.3:compile
[INFO] | +- commons-logging:commons-logging:jar:1.0.4:compile
[INFO] | +- asm:asm-attrs:jar:1.5.3:compile
[INFO] | +- dom4j:dom4j:jar:1.6.1:compile
[INFO] | +- antlr:antlr:jar:2.7.6:compile
[INFO] | +- cglib:cglib:jar:2.1_3:compile
[INFO] | +- asm:asm:jar:1.5.3:compile
[INFO] | \- commons-collections:commons-collections:jar:2.1.1:compile
[INFO] \- junit:junit:jar:3.8.1:test
[INFO] ------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------
```

From this output, we can see that the persistence-api dependency is coming from hibernate. A cursory scan of the source in this module will reveal many javax. Persistence import statements confirming that we are, indeed, directly referencing this dependency. The simple fix is to add a direct reference to the dependency. In this example, we put the dependency version in simple-parent's dependency Management section because the dependency is linked to Hibernate, and the Hibernate version is declared here. Eventually you are going to want to upgrade your project's version of Hibernate. Listing the persistence-api dependency version near the Hibernate dependency version will make it more obvious later when your team modifies the parent POM to upgrade the Hibernate version.

Caching Implicit Dependencies

Scanning each file for #includes lines does take some extra processing time. When we are doing a full build of a large system, the scanning time is usually a very small percentage of the overall time spent on the build. We are most likely to notice the scanning time, however, when build all or part of a large system were: SCons will likely take some extra time to "think about" what must be built before it issues the first build command (or decides that everything is up to date and nothing must be rebuilt).

In practice, having SCons scan files saves time relative to the amount of potential time lost to tracking down subtle problems introduced by incorrect dependencies. Nevertheless, the "waiting time" while SCons scans files can annoy individual developers waiting for their builds to finish. Consequently, SCons lets us to cache the implicit dependencies that its scanners find, for use by later builds. We can do this by specifying the --implicit-cache option on the command line:

```
% scons -Q --implicit-cache hello
cc -o hello.o -c hello.c
cc -o hello hello.o
% scons -Q hello
scons: `hello' is up to date.
```

If we don't want to specify --implicit-cache on the command line each time, you can make it the default behavior for your build by setting the implicit_cache option in anSConscript file:

```
SetOption ('implicit_cache', 1)
```

SCons does not cache implicit dependencies like this by default because the --implicit-cache causes SCons to simply use the implicit dependencies stored during the last run, without any checking for whether or not those dependencies are still correct. Specifically, this means --implicit-cache instructs SCons to not rebuild "correctly" in the following cases:

- When --implicit-cache is used, SCons will ignore any changes that may have been made to search paths (like $CPPPATH or $LIBPATH,). This can lead to SCons not rebuilding a file if a change to $CPPPATH would normally cause a different, same-named file from a different directory to be used.
- When --implicit-cache is used, SCons will not detect if a same-named file has been added to a directory that is earlier in the search path than the directory in which the file was found last time.

The --implicit-deps-changed option

When using cached implicit dependencies, sometimes we want to "start fresh" and have SCons re-scan the files for which it previously cached the dependencies. For example, if we have recently installed a new version of external code that you use for compilation, the external header files will have changed and the previously-cached implicit dependencies will be out of date. You can update them by running SCons with the --implicit-deps-changed option:

```
% scons -Q --implicit-deps-changed hello
```

```
        cc -o hello.o -c hello.c
        cc -o hello hello.o
        % scons -Q hello
    scons: `hello' is up to date.
```

In this case, SCons will re-scan all of the implicit dependencies and cache updated copies of the information.

The --implicit-deps-unchanged option

By default when caching dependencies, SCons notices when a file has been modified and re-scans the file for any updated implicit dependency information. Sometimes, however, you may want to force SCons to use the cached implicit dependencies, even if the source files changed. This can speed up a build for example, when we changeour source files but haven't changed any #include lines. In this case, we can use the --implicit-deps-unchanged option:

```
        % scons -Q --implicit-deps-unchanged hello
        cc -o hello.o -c hello.c
        cc -o hello hello.o
        % scons -Q hello
    scons: `hello' is up to date.
```

In this case, SCons will assume that the cached implicit dependencies are correct and will not bother to re-scan changed files. For typical builds after small, incremental changes to source files, the savings may not be very big, but sometimes every bit of improved performance counts.

Conclusion

We implemented the Semantic mining of software repositories and combining mined results with IR can improve the accuracy (precision and recall) of IR techniques. We apply Dependency Estimation on to compare the accuracy of its dependency links with those recovered using state-of-the-art IR techniques from Vector Space model and Concept based mining. We thus show that mining software repositories and combining the mined data with existing results from IR techniques improves the precision and recall of requirement dependency links. Also result outperforms in the convergence ratio in terms of software evolution and task evolution in dependency mapping.

References

1. Clark P, Thompson J, Holmback H, Duncan L (2000) Exploiting a Thesaurus-Based Semantic Net for Knowledge-Based Search. In Proceeding 12th Conference on Innovative Applications of AI (AAAI/IAAI'2000) 988-995.
2. Baldwin CY, Clark KB (2000) Design Rules: The Power of Modularity. MIT Press.
3. Basili VR, Perricone BT (1984) Software Errors and Complexity: An Empirical Investigation. Comm. of the ACM 12: 42-52.
4. Briand LC, Wust J, Daly JW, Porter DV (2000) Exploring the Relationships between Design Measures and Software Quality in Object- Oriented Systems. The Journal of Systems and Software 51: 245-273.
5. Burt RS (1992) Structural Holes: The Social Structure of Competition. Harvard University Press.
6. Cataldo M, Wagstrom P, Herbsleb JD, Carley KM (2006) Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. In Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'06) 353-362.
7. Cataldo M (2007) Dependencies in Geographically Distributed Software Development: Overcoming the Limits of Modularity. Ph.D. dissertation, Institute for Software Research, School of Computer Sciences, Carnegie Mellon University.
8. Cataldo M, Bass M, Herbsleb JD, Bass L (2007) On Coordination Mechanism in Global Software Development. In Proceedings of the International Conference on Global Software Engineering (ICGSE '07) 71-80.
9. Montoyo A, Suarez A, Palomar M (2002) Combining supervised-unsupervised methods for Word Sense Disambiguation. Presented at International conference on Intelligent Text Processing and Computational Linguistics -CICLing-2002. Lecture Notes in Computer Science 2276: 156-164.
10. Roberto N, Stefano F, Aitor S, Oier deL, Eneko A (2011) Two birds with one stone: Learning semantic models for text categorization and Word Sense Disambiguation. In Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, UK 2317–2320.
11. Curtis B, Kransner H, Iscoe NA (1988) field study of software design process for large systems. Comm. of ACM 31: 1268-1287.
12. de Souza CRB (2005) On the Relationship between Software Dependencies and Coordination: Field Studies and Tool Support. Ph.D. dissertation, Donald Bren School of Information and Computer Sciences, University of California, Irvine.
13. de Souza CRB, Redmiles D, Cheng L, Millen D, Patterson J (2004) How a Good Software Practice Thwarts Collaboration – The multiple roles of APIs in Software Development. In Proceedings of the Conference on Foundations of Software Engineering (FSE '04) 221-230.
14. Eaddy M, Zimmermannn T, Sherwood KD, Garg V, Murphy GC, Nagappan N, Aho AV (2008) Do Crosscutting Concerns Cause Defects? IEEE Trans. on Soft. Eng. 34: 497-515.
15. Eick SG, Graves TL, Karr AF, Mockus A, Schuster P (2002) Visualizing Software Changes. IEEE Trans. on Soft. Eng. 28: 396-412.
16. Eppinger SD, Whitney DE, Smith RP, Gebala DA (1994) A Model-Based Method for Organizing Tasks in Product Development. Research in Eng. Design 6: 1-13.
17. Johnston M, Hanna JRP, Millar RJ (2004) Advances in dataflow programming languages 36: 1-34.