CheckDep: A Tool for Tracking Software Dependencies

Dirk Beyer
Simon Fraser University, B.C., Canada
& University of Passau, Germany

Ashgan Fararooy
Simon Fraser University, B.C., Canada

Abstract

Many software developers use a syntactical 'diff' in order to perform a quick review before committing changes to the repository. Others are notified of the change by e-mail (containing diffs or change logs), and they review the received information to determine if their work is affected. We lift this simple process from the code level to the more abstract level of dependencies: a software developer can use CheckDep to inspect introduced and removed dependencies before committing new versions, and other developers receive summaries of the changed dependencies via e-mail. We find the tool useful in our software-development activities and now make the tool publicly available.

1. Overview

The objective of CHECKDEP is to provide feedback on the consequences of code changes in terms of dependencies. Our project was motivated by a concrete problem in an industrial project: controller software was continuously extended for new product versions, and the originally well-designed software degenerated. The problem for maintenance in this project is the amount of inter-dependencies between the various subsystems. A first measure to approach the problem is to make sure that no new dependencies are introduced, and therefore all new changes (commits) are inspected for dependency reduction. CHECKDEP provides a list of changes in the dependency relations for any two versions of the project. A second measure is to continuously refactor the system —whenever the product cycle allows— in order to stepwise transform the system to a better subsystem structure, and to evaluate every refactoring step wrt. the amount of dependencies introduced and removed. CHECKDEP compares the workspace copy with the head revision before check-in, and notifies the developer of introduced dependencies. A change that reduces the number of inter-subsystem dependencies is considered good under this measure, and a change that introduces more new dependencies than it removes is considered suspicious and the developer is alarmed. In addition to immediate textual feedback, we provide a clustering visualization in order to locate and investigate the dependency changes.

CHECKDEP can be used as a command-line tool or as Eclipse plug-in. Command-line invocations are necessary in automatic processes, such as being called from a Subversion hook script automatically after each commit. The plug-in works in two steps: First, the user has to specify the dependency types to be extracted (combination of call, inheritance, and field access) and the two versions to compare (either by paths of working directories, or by URLs of Subversion repositories). Second, the processing is done and the textual and visual results are shown. The textual results are a brief summary of the dependency changes and a list of all added and removed dependencies in RSF format. The visualization is based on a clustering layout (CCVISU) that easily identifies the area of change in a (large) project graph.

CHECKDEP is free software, released under the Apache 2.0 license. The source code, example screen shots, and more information on the tool are available at: `http://www.sosy-lab.org/~dbeyer/CheckDep`

Several existing tools have addressed similar problems (CREOLE, DA4JAVA, DEPAN, DEPENDENCYFINDER). The contribution of CHECKDEP is to provide a lightweight tool for analyzing dependency changes in different versions via a smooth integration with Subversion. A unique characteristic of CHECKDEP is its special visualization feature. The plug-in supports Java, C, and C++ as input, and we use DOXYGEN and DEPENDENCYFINDER as fact extractors.

2. Features

Dependency Differences. The goal of CHECKDEP is to track and visualize dependency changes between different versions of software. The result is returned as a dependency graph where added and removed artifacts and dependencies are highlighted in different colors.

Subversion Integration. CHECKDEP can be applied to different local workspace copies, but also to remote version repositories (specified by URL and revision number via choice list on the configuration screen).

978-0-7695-4113-6/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPC.2010.51

project (types / members)	rev	co	bld	extr
CCVISU (236 / 1296)	28	26.4	6.2	4.8
	34	117	1.4	
CPACHECKER (680 / 4656)	569	35.5	12	12.3
	574	33.1	3.0	
CHIC4WEB (133 / 1198)	114	65.2	0.9	2.9
	120	84.8	0.8	

Table 1. Tool Performance (in seconds)

Visualization. For the visual presentation of the dependency differences, we obtained the best results using a clustering layout. Artifacts (classes, members) are drawn as discs which are placed close to each other if they are coupled by many dependencies, and at distant positions if they are loosely coupled [1]. An edge represents the dependency between the two connected artifacts. New dependencies are highlighted using red, removed dependencies using green edges. This layout reflects the subsystem structure of the software, and at the same time helps the user to find the parts of the system that have changed and navigate through the graph. CHECKDEP also provides a simple graph-query mechanism to focus on graph changes only. E.g., CHECKDEP can be configured to show a subgraph that contains only the added and removed edges in the most recent view.

Extraction of Dependency Graphs. CHECKDEP uses different fact extractors to retrieve various kinds of dependencies between classes and methods, from Java, C, and C++ programs. Dependencies can be method calls, inheritance relations, type-field usage relations, etc.

Performance. Table 1 shows for each project the total number of distinct types (classes), the total number of distinct members, the revision number, the times in seconds needed for check-out, build, and fact extraction. The input repositories for the performance tests were CCVISU[1], CPACHECKER[2] and CHIC4WEB[3]. The tests were run on a dual core processor at 2.10 GHz with 4 GB of memory.

3. Application Examples

Development. The tool can be used to compare the developer's working copy against the head revision of the repository wrt. dependencies, before committing new changes. The differences in dependencies can be investigated graphically (clustering layout with changed dependencies highlighted) or textually (RSF format). Filters and zoom-in can be used to restrict the result to a certain part of the software. A search feature can be used to locate specific software elements in the graphical view.

Refactoring. The artifacts that participate in a refactoring are related, and therefore, they are closely placed in the layout and easy to locate. The colored edges are highlighting the dependency changes that a refactoring is responsible for. Short edges are not very important, because the artifacts that are placed closely together are already related. The longer an edge is, the more important and critical the dependency: very long edges represent inter-subsystem dependencies, and removal of such a dependency is a large gain, but introducing such a dependency is degenerating the overall structure of the system in most cases (according to classic definitions, a structured system consists of cohesive subsystems that are loosely coupled). In Fig. 1, CHECKDEP illustrates a local refactoring.

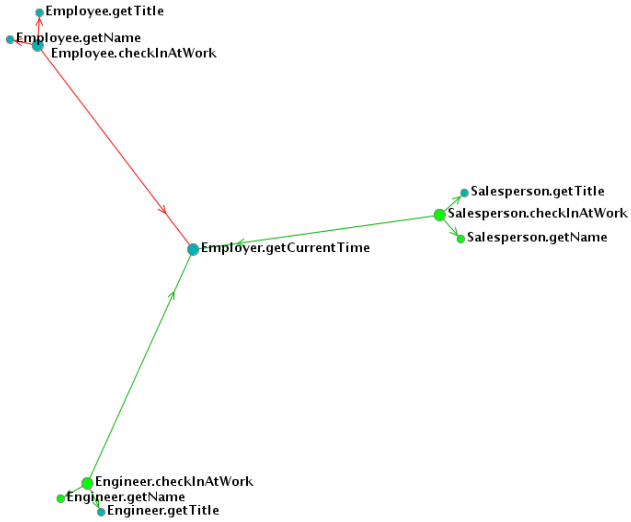

Figure 1. A 'pull-up method' refactoring removes 6 dependencies (green) and adds 3 (red), which improves the software structure

Design-Change Identification. Recently, an approach was introduced to automatically determine if a change in a program impacts the design (i.e., UML class diagram) of the related system [2]. According to the paper, 'design changes' are identified in a series of steps: first by exploring the addition or deletion of classes, then methods, and finally changes in dependency relationships (e.g., generalization, association). CHECKDEP can be a valuable complement: most of the aforementioned changes are directly identifiable and highlighted within various dependency graphs provided by CHECKDEP. For example, added or deleted 'generalizations' and 'associations' are pointed out by added or removed dependencies and nodes in the 'inheritance' and the 'type-field' graphs, respectively.

[1] http://ccvisu.sosy-lab.org
[2] http://cpachecker.sosy-lab.org
[3] http://www.sosy-lab.org/~dbeyer/Chic4web

References

[1] D. Beyer. CCVISU: Automatic visual software decomposition. In *Proc. ICSE*, pages 967–968. ACM, 2008.
[2] M. Hammad, M. L. Collard, and J. I. Maletic. Automatically identifying changes that impact code-to-design traceability. In *ICPC*, pages 20–29, 2009.