Poster: Discovering Code Dependencies by Harnessing Developer's Activity

Martin Konopka, Pavol Navrat, Maria Bielikova
Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
{martin_konopka, pavol.navrat, maria.bielikova}@stuba.sk

***Abstract*—Monitoring software developer's interactions in an integrated development environment is sought for revealing new information about developers and developed software. In this paper we present an approach for identifying potential source code dependencies solely from interaction data. We identify three kinds of potential dependencies and additionally assign them to developer's activity as well, to reveal detailed task-related connections in the source code. Interaction data as a source allow us to identify these candidates for dependencies even for dynamically typed programming languages, or across multiple languages in the source code. After first evaluations and positive results we continue with collecting data in professional environment of Web developers, and evaluating our approach.**

***Index Terms*—Source code dependency, interaction data, task context, implicit feedback, dynamic typing.**

I. Introduction

From a software developer's view, dependencies between various source code entities are of the most interest during development and maintenance [2, 4]. Syntactic analysis, which is prominently used for discovering source code dependencies, is however insufficient for identifying hidden task-related connections [2, 3], for dynamically typed languages, or when combining programming languages [5]. Much of important knowledge about source code is not present in its contents but remembered by developers themselves [2, 5, 6].

In this work, we consider developer's interactions recorded in an integrated development environment (IDE) [1, 2] as an alternative source for identification of potential source code dependencies – identified from *implicit feedback* to source code. Because of the low-level information of an interaction, we also identify developer's activities, e.g., to distinguish navigation in source code when studying (no changes in code) from debugging (between start and stop of debugging) or refactoring. Our contribution is to: (1) identify direct task-related connections between source code components, not just their groups; and (2) supplement dependencies for dynamically typed languages without analyzing their contents.

II. Related Work

Existing works use software developer's activity to identify relations between software artifacts or developers, ranging from recommending task assignees [6], through capturing task contexts [2], to recommending artifacts of developer's interest [1]. Many approaches attempt to reduce a space of task-related software artifacts, e.g., for bug fixing, but further research is required to uncover their direct dependencies and thus reduce the sparsity of an identified task context [7].

Quite some contextual information may be supplied by processing developer's activity, thus complementing insufficient outcomes of a syntactic analysis. Even though static typing have been introduced for some dynamically-typed languages, e.g., TypeScript for JavaScript, it is still difficult to identify inter-language dependencies, e.g., JavaScript client of a web service in C# [5], or for configuration files.

The most frequent developer's action is navigation in source code space [4]. Developer may navigate using references inferred by a syntactic analysis, but may also pick components from codebase with her own intent, even at random. Navigation paths have been used for uncovering relations between developers [6], though they may be used for components too, as well as with other interactions, such as copy-pasting a code fragment, or committing files [3].

III. Potential Source Code Dependencies

We use developer's interactions in an IDE, as an implicit feedback to source code [1], to identify *potential* source code dependencies [3] independently from source code contents. We define three kinds of such dependencies:

- *navigation* – from navigation between components,
- *content* – copy-pasting a code, and
- *commit* – committing changes to a code repository.

A developer performs *interactions* within *activities* to accomplish her *task*. Such interactions are open a source code document, navigate to a definition, or paste a code fragment [1]. Activities can be just studying her own but old code, someone else's code, or implementing a new feature. When a developer navigates between certain components often, it implicitly raises possibility that some connections exist between them, even when they are syntactically independent at all [2, 3]. However, developer's navigation when studying unknown code is more exploratory and may be repetitive to strengthen understanding of an implementation, thus less likely to reveal potential dependencies. But navigation when debugging is more focused on a problem, may be more precise and significant. Similar applies to copy-paste actions correcting all callers of a refactored method in contrary with copy-pasting of repetitive but required code when implementing new feature.

We use a tuple $d_{imp} = (s_1, s_2, t, a, p, w)$ to describe a potential dependency, with s_1 and s_2 for source and target components, t for timestamp of a sourced interaction, activity a which an interaction belongs to, weight w, and property p specific for each kind of dependency [3]:

- navigation – dwell time in a target component,
- copy-paste – content of a copy-pasted fragment,
- commit – total count of changed files in a commit.

Introduction of activity $a \in A$ in the tuple allows us to weight potential dependencies of the same kind differently for different activities, e.g., debugging, testing, or code studying. Identifying the set of activities A from recorded interactions is our ongoing research whose results are applicable here.

IV. Prioritizing Potential Dependencies

The identification process [3] produces many potential dependencies from large set of developer's interactions in IDE. To determine important ones and to compare them, we do:

1) *Weighting* each d_{imp} with the weight $w \in \langle 0,1 \rangle$ using property p and activity a, differently for each kind, e.g., for navigation kind the *dwell time* is used to filter out developer's mistakes and the activity to promote focused and task-important navigation to just exploratory and erroneous ones.

2) Validating dependencies with exponential decay function to promote relevant dependencies to a selected point in time, i.e., a dependency is less relevant when it is older [3].

3) Aggregating dependencies between pairs of source code components into oriented edges $e_{imp} \in E_{imp}$, in dependency graph, where $e_{imp} = (v_1, v_2, w_e, D_{imp}')$, with both source and target vertices v_1 and v_2, weight $w_e \in \mathbb{R}^+$, as in Eq. 1, and subset D_{imp}' of all potential dependencies from v_1 to v_2.

$$w_e = \sum_{d_{imp} \in D'_{imp}} validity(d_{imp}) * w \qquad (1)$$

We then construct a graph of potential dependencies $DG_{imp}(V,E_{imp})$ with vertices V of source code components and weighted oriented edges E_{imp}. DG_{imp} is an extension to traditional dependency graph identified from *explicit* statements in source code (type and member references, inheritance, and method calls), it can be similarly visualized or further analyzed.

V. Dynamic typing and Inter-language Support

Because of not depending on a syntactic analysis, we see application of our work for dynamically typed languages and inter-language implementations. We have evaluated it on a set of five C#/.NET projects [3], three of which were ASP.NET MVC projects with multiple languages used. We asked developers to consider each edge in created dependency graphs (we had excluded edges covered by explicit code dependencies) whether an edge represented existing but hidden dependency in the source code that would be required to check if connected components were changed. We achieved precision ranging from 76.67% to 91.96%, possibly because of projects being of small size (2.4 to 5.3 KLOC), of short duration (one year), and only one developer of whole team participated for each project. Developers welcomed that we had identified dependencies in loosely coupled code, even though they were hidden or resolved in runtime only, such as webpages and scripts, configuration files, or webpages and data classes.

We continue with collecting interaction data of 20 professional developers working on Web projects in HTML, JavaScript, and Java. We also extend monitoring of developers in IDE with recording more detailed interaction events.

VI. Discussion

In contrary to using syntactic analysis for identifying source code dependencies, properties of our approach depends on sourced interaction data and of evaluated source code. We are not able to identify potential dependencies in source code which we have few or none interactions recorded for. We also assume that decay function may be insufficient for validation when source code structure often changes. Nevertheless, we are able to identify hidden or task-related connections in source code, as well as connections when dynamic typing or multiple languages are used, thus to improve source code maintenance.

Our work is part of the research project PerConIK – Personalized Conveying of Information and Knowledge (perconik.fiit.stuba.sk) – to employ data of developer's activity in software evaluation and maintenance [1].

Acknowledgment

This work was partially supported by the Scientific Grant Agency of Slovakia, grants No. VG 1/0752/14 and VG 1/0646/15, and it is the partial result of the Research & Development Operational Programme for the project PerConIK, ITMS 26240220039, co-funded by the ERDF.

References

[1] M. Bieliková, I. Polášek, M. Barla, E. Kuric, K. Rástočný, J. Tvarožek, P. Lacko, "Platform independent software development monitoring: Design of an architecture," in Proc. of SOFSEM 2014, Springer-Verlag, 2014, pp. 126-137.

[2] M. Kersten, G.C. Murphy, "Using task context to improve programmer productivity," in Proc. of SIGSOFT '06/FSE-14, ACM, 2006, pp. 1-11.

[3] M. Konôpka, M. Bieliková, "Software developer activity as a source for identifying hidden source code dependencies," in Proc. of SOFSEM 2015, Springer-Verlag, 2015, pp. 449-462.

[4] J.-P. Krämer, T. Karrer, J. Kurz, M. Wittenhagen, J. Borchers, "How tools in IDEs shape developers' navigation behavior," in Proc. of CHI '13, ACM, 2013, pp. 3073-3082.

[5] H.V. Nguyen, C. Kästner T.N. Nguyen, "Building call graphs for embedded client-side code in dynamic web applications," in Proc. of FSE 2014, ACM, 2014, pp. 518-529.

[6] R. Robbes, D. Röthlisberger, "Using developer interaction data to compare expertise metrics," in Proc. of MSR '13, IEEE Press, 2013, pp. 297-300.

[7] D. Zeleník, "Reducing the sparsity of contextual information for recommendation," in Information Sciences and Technologies Bulletin of the ACM Slovakia, vol. 6, no. 2, 2014, pp. 21-28.