Weekly Meeting

1. PROBLEM STATEMENT

Developers frequently use source code examples as the basis for interacting with an application programming interface (API) to obtain the needed functionality. In this case, the source code example serves as the explicit origin for a reuse task when it fits a developer's context sufficiently well [4,12]. We define a reuse task as a series of copy-paste-modify actions to transform a piece of code to the target system, aiming to obtain desired functionalities by invoking a sequence of APIs with corresponding control structure. During the process of reuse, developers often queries for a set of examples considering the task, and integrate examples that match the task and can be reused in the target context [5]. Both phases can be iterative as there is no guarantee that a selected example will be appropriate until the integration results are examined.

While many existing works focus on helping developers locate relevant source code examples [1, 6, 10, 11, 14, 17], few approaches exist to support integration of source code [2]. The integration phase is not trivial: First, there exist multiple examples that provides desired functionality for the task. Recent study [5] shows that it is unlikely to find a 'perfect' example to provide all desired functionalities for the reuse task, and the desired functionalities might scatter in multiple examples. Even with help of code recommendation tools, developers still need to manually investigate multiple recommended code examples that are found based on the relevance to the user query rather than the ability to fit for the target context. Second, for any selected example in non-trivial system, the number of structural dependencies to follow is much too large to be completely covered by a developers [7]. As a result, developers have to rely on their intuition to determine which elements should be integrated. Finally, when users try to modify the selected example to the target context, they must measure structural and semantic similarity between the example and target context [2], determine which functionality should be or can be transformed to the target, and eliminate as much irrelevant reused code as practical to save maintenance cost.

To overcome these challenges, we propose to automatically identify common functionalities across different source code examples as well as common coupling elements that interact with common functionalities in a common way. We define coupling elements as ones that are data dependent and control dependent on the common functionalities, callers/callees of these functionalities [13], and sibling elements that share the same callers/callees [15]. Given a set of examples for a desired reuse task, our tool returns a reuse task suggestion in the format of a sequence of API calls and corresponding control structure after transforming them to fit for the target context [2]. Our tool also lists a set of variants from multiple source code examples so that users can select the functions they prefer to integrate into the target context. We assume that *common functionalities are the major functionalities that user intends to reuse based on the user query and their coupling elements also interact with these functionalities in a common way.*

Although some approaches advocate refactoring code rather than reuse code in this manner [3], recent researches have found that these kind of clone cannot be easily refactored [9] and have to be modified to meeting requirements in new context [16]. Gilligan [8] and Procrustes [7] try to address the problem of example integration in the context of large-scale reuse tasks by suggesting program elements based on structural relevance and cost of reuse. They aim to ease the process of choosing elements that are easy to reuse but developers still have to manually go through relevant elements for each example and modify them to integrate into the desired context. Jigsaw [2] supports small-scale integration of source code into target system between the example and target context. By considering structural and semantic similarity measures, it greedily matches each element between two contexts and simply copies the source element to the target if it does not correspond with any element in the target. Unfortunately, developer has to provide source and target to enable a one-to-one transformation and resolve all dependencies when pasting reused code to the target.

2. REFERENCES

[1] S. Chatterjee, S. Juvekar, and K. Sen. SNIFF: A search engine for java using free-form queries. In *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, pages 385–400, 2009.

[2] R. Cottrell, R. J. Walker, and J. Denzinger.

Semi-automating small-scale source code reuse via structural correspondence. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, Atlanta, Georgia, USA, November 9-14, 2008*, pages 214–225, 2008.

[3] M. Fowler. *Refactoring - Improving the Design of Existing Code*. Addison Wesley object technology series. Addison-Wesley, 1999.

[4] W. B. Frakes and K. Kang. Software reuse research: Status and future. *IEEE Trans. Software Eng.*, 31(7):529–536, 2005.

[5] R. Holmes, R. Cottrell, R. J. Walker, and J. Denzinger. The end-to-end use of source code examples: An exploratory study. In *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada*, pages 555–558, 2009.

[6] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, pages 117–125, 2005.

[7] R. Holmes, T. Ratchford, M. P. Robillard, and R. J. Walker. Automatically recommending triage decisions for pragmatic reuse tasks. In *ASE 2009, 24th IEEE/ACM International Conference on Automated Software Engineering, Auckland, New Zealand, November 16-20, 2009*, pages 397–408, 2009.

[8] R. Holmes and R. J. Walker. Supporting the investigation and planning of pragmatic reuse tasks. In *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007*, pages 447–457, 2007.

[9] M. Kim, V. Sazawal, D. Notkin, and G. C. Murphy. An empirical study of code clone genealogies. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2005, Lisbon, Portugal, September 5-9, 2005*, pages 187–196, 2005.

[10] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman. Jungloid mining: helping to navigate the API jungle. In *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005*, pages 48–61, 2005.

[11] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu. Portfolio: finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, pages 111–120, 2011.

[12] J. Parsons and C. Saunders. Cognitive heuristics in software engineering: Applying and extending anchoring and adjustment to artifact reuse. *IEEE Trans. Software Eng.*, 30(12):873–888, 2004.

[13] M. P. Robillard. Automatic generation of suggestions for program investigation. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2005, Lisbon, Portugal, September 5-9, 2005*, pages 11–20, 2005.

[14] N. Sahavechaphan and K. T. Claypool. Xsnippet: mining for sample code. In *Proceedings of the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA*, pages 413–430, 2006.

[15] Z. M. Saul, V. Filkov, P. T. Devanbu, and C. Bird. Recommending random walks. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007*, pages 15–24, 2007.

[16] R. W. Selby. Enabling reuse-based software development of large-scale systems. *IEEE Trans. Software Eng.*, 31(6):495–510, 2005.

[17] S. Thummalapenta and T. Xie. Parseweb: a programmer assistant for reusing open source code on the web. In *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*, pages 204–213, 2007.