Integrated Solution Engineering

Leiguang Gong, Tim Klinger, Paul Matchen,
Peri Tarr, Rosario Uceda-Sosa, Annie Ying
IBM Thomas J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532 USA
+1 914-784-7278
{leiguang, tklinger, matchen, tarr, rosariou, aying}@us.ibm.com

Jian Xu, Xin Zhou
IBM China Research Laboratory
Zhong Guang Cun Software Park, Haid, Tower A, Diamond Building
Beijing, 100094 China
+86-10-58748008
{xujian, zhouxin}@cn.ibm.com

Abstract
Integrated Solution Engineering helps developers manage software complexity by offering semi-automated support for capturing and mining relationships among artifacts and/or developer tasks at different stages of the software lifecycle, and by aiding developers in the use and management of the information contained in these relationships. The use of these relationships can facilitate traceability, propagation of change, change impact analysis, evolution, and comprehension.

Categories and Subject Descriptors D.2.6 [**Software Engineering**]: Programming Environments – *integrated environments, interactive environments, programmer workbench.*

General Terms Algorithms, Reliability, Design, Human Factors.

Keywords Traceability, end-to-end software lifecycle, usability, propagation of change, consistency management

1. Introduction
Software traceability has been long recognized as one of the most fundamental necessities to achieving effective end-to-end software engineering [1]. It is required to achieve a wide range of activities and goals, including tool integration, propagation of change, impact of change analysis, task assistance and semi-automation, consistency management, IT governance and process assessment. Yet the software traceability problem has remained among the most intractable of problems for more than two decades [2]. At its crux are two key issues. First, people remain largely unmotivated to provide and evolve traceability information, as the stakeholder who knows the information is rarely the one who benefits from it. Second—and most critically—even if the people were motivated, it is infeasible for humans to define and maintain traceability information throughout the course of evolution. There are too many stakeholders, too many software artifacts, too much evolution, and far too many interrelationships for this task to be addressed entirely by people.

Where humans cannot or will not accomplish tasks unaided, automation is a particularly attractive option, and this has been the case in the area of software traceability as well. The literature on software traceability reflects numerous researchers' attempts to automate the identification of artifact interrelationships, using a wide variety of techniques [3, 4, 5, 6 7, 8, 9]. These attempts have all experienced significant problems with scalability and—most notably—with reliability, correctness and usability and, as a result, none of them are currently in use in any real-world applications. Not surprisingly, there are no silver bullets—every technique has demonstrated a different cost-benefit tradeoff, particularly with respect to performance, precision, and recall. None of the techniques used have been considered efficient and reliable enough for use in real software processes. Very significantly, the limited sets of tools that developers have been offered to utilize traceability information have also demonstrated very poor functional and usability characteristics. Finally, fully automating the identification and evolution of interrelationships is not possible, given that some relationship semantics are implicit and depend on human knowledge.

We have, then, a conundrum. Neither humans nor software can solve the traceability problem unaided, and while people greatly appreciate the potential benefits that can be reaped from having traceability, they remain generally unwilling to put much time or effort into maintaining the interrelationship information that underlies a traceability feature.

2. Integrated Solution Engineering
Integrated Solution Engineering (ISE) is an attempt to address this conundrum. Since we cannot fully automate traceability, and since we cannot leave it to people, our goal instead is to find the "sweet spot" between manual input from the user and automation. We accomplish this by leveraging information about the stakeholders' tasks. Our approach has two major parts: a traceability relationship[1] identification and evolution part, and a visualization part. The first part—*semi-automated relationship inferencing* (SARI)—covers both the initial identification of these relationships, as well as the removal of relationships when they become incorrect, useless, or irrelevant. We are researching novel ways of doing this scalably. SARI differs from other relationship identification approaches in several ways:

- For each type of relationship, our approach combines *multiple* techniques to identify new relationships and discard invalid ones. In this way, it leverages the best features of a variety of techniques to produce better results, while minimizing their individual weaknesses. Moreover, the architecture is *open* and *extensible*—new techniques can be incorporated at any time.
- It identifies relationships between existing artifacts—not just when artifacts are created—so it makes no unrealistic greenfield assumptions. It also can capture relationships during

[1] *Traceability relationships* include, for example, the relationship between code and the design it implements, between a test case and the code it tests, and between design and the use case(s) it realizes. ISE addresses a subset of common traceability relationships.

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

the artifact creation process—a time when more information is often available to help with the creation of such relationships. For example, it is easy to tell which code is being tested at the point where a developer creates a test case for that code. It is comparatively more difficult to determine this information once the test case exists.

- SARI assigns qualitative “goodness” ratings to each relationship, as well as an explanation for why it believes the relationship is valid. This enables stakeholders to understand and control the tradeoff between completeness and accuracy.

- It has been designed and developed with a constant eye towards incrementality and scalability.

The second part of the ISE approach addresses the end-user experience and provides context- and role-sensitive assistance to users. Two key limitations of all traceability approaches that we have seen are the poor quality and low usability characteristics of associated tools that help stakeholders to utilize and manage traceability information—there is too much information to render most traditional approaches viable, and the tools themselves fail to take into account developer roles and tasks. The ISE approach, by contrast, specifically provides novel, lightweight visualization technologies that aid stakeholders in using the traceability information effectively, filtering information according to particular stakeholders’ tasks and roles. These include:

- Context-sensitive filtered and conversational explorers: ISE provides a filterable view of the entire “mesh” of interrelated artifacts. This view is most useful for debugging and for detailed work by experts. It contains, however, too much information to be useful for typical developers. Therefore, ISE also provides *conversational explorers*. These start with a particular artifact of interest and allow a developer to explore paths of relationships to other artifacts, where the relationships offered are ones that are relevant to the stakeholder and their task. These explorers keep track of the interaction with the user and help locating and navigating information based on the conversation history. A variety of visualizations are being explored.

- Assisted navigation via interactive, stepwise queries: Queries are a key mechanism for finding information about both relationships and interrelated artifacts, but formulating correct queries and showing their results in a usable fashion is a largely unsolved and inherently difficult problem. ISE provides a visualization component that aids users in formulating queries, offering immediate feedback as the query is being constructed.

Moreover, when the semi-automated inferencing engine identifies problematic relationships—ones it cannot resolve itself—the user assistance part solicits help from the user in a non-intrusive, focused manner, thus minimizing the amount of effort an unwilling stakeholder has to put into maintaining traceability information.

3. Conclusion

ISE is a novel approach to addressing the critical traceability problem. By combining a variety of novel and user-friendly techniques, it helps stakeholders to develop, evolve, and use traceability effectively to manage software complexity. It offers semi-automated support for capturing and mining relationships among artifacts and/or developer tasks at different stages of the software lifecycle, and it potentially will help stakeholders to use these relationships to facilitate traceability, propagation of change and change impact analysis, evolution, task assistance and semi-automation, assessment, and comprehension.

Acknowledgements

We are grateful to Jim Wright, Yit Phang Khoo and Mithun Acharya for their work on the ISE project.

References

[1] Gotel, O. and Finkelstein, A.. An Analysis of the Requirements Traceability Problem. In *Proceedings of the IEEE International Conference on Requirements Engineering (ICRE)*, 1994.

[2] Ramesh, B., Stubbs, C., Powers, T., and Edwards, M.: "Implementing Requirements Traceability: A Case Study", *Annals of Software Engineering 3* (1997), 397-415.

[3] Antoniol G. and Canfora G. and Casazza G. and De Lucia A. and Merlo E. Design-Code traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 2002.

[4] Murphy, G. C., Notkin, D., and Sullivan, K. Software Reflexion Models: Bridging the Gap Between Source and High-Level Models. In *Proceedings of Foundations of Software Engineering*, 1995.

[5] Faisal, M. Toward Automating the Discovery of Traceability Links, Ph.D. Thesis, 2005.

[6] Marcus, A. and Maletic, J. I. Recovering Documentation-to-Source-Code Traceability Links Using Semantic Indexing. In *Proceedings of the 25th International Conference on Software Engineering (ICSE’03)*, 2003.

[7] Mandelin, D., Kimelman, D., and Yellin, D. A Bayesian Approach to Diagram Matching with Application to Architectural Models. In *Proceedings of the 28th International Conference on Software Engineering (ICSE’06)*, 2006.

[8] Gotel, O. and Finkelstein, A. Extending Requirements Traceability Through Contribution Structures. *ACM Transactions on Software Engineering and Methodology*, 1997.

[9] Ren, X., Shah, F., Tip, F., Ryder, B., and Chesley, O. Chianti: A tool for change impact analysis of Java programs” by In *Proceedings of the International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2004)*, October 26-28, 2004