View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by AIS Electronic Library (AISeL)

Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2009 Proceedings

Americas Conference on Information Systems (AMCIS)

2009

An Empirical Investigation of the Key Factors for Refactoring Success in an Industrial Context

Yi Wang
City University of Hong Kong, ywang43@student.cityu.edu.hk

Christian Wagner
City University of Hong Kong, iscw@cityu.edu.hk

Rachael K.F. Ip
City University of Hong Kong, isip@cityu.edu.hk

Follow this and additional works at: http://aisel.aisnet.org/amcis2009

Recommended Citation
Wang, Yi; Wagner, Christian; and Ip, Rachael K.F., "An Empirical Investigation of the Key Factors for Refactoring Success in an Industrial Context" (2009). *AMCIS 2009 Proceedings*. 577.
http://aisel.aisnet.org/amcis2009/577

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 2009 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

An Empirical Investigation of the Key Factors for Refactoring Success in an Industrial Context

Yi Wang
Department of Information System
City University of Hong Kong
ywang43@student.cityu.edu.hk

Christian Wagner
Department of Information System
City University of Hong Kong
iscw@cityu.edu.hk

Rachael K.F. Ip
Department of Information System
City University of Hong Kong
isip @cityu.edu.hk

ABSTRACT

Refactoring is an increasingly practiced method in industrial software development. Stated simply, refactoring is an ongoing software improvement process that simplifies the internal structure of existing software, without changing its external behavior. The purpose is to improve the software and facilitate future maintenance and enhancement. Existing studies on refactoring mainly focus on its technical aspects and thus do not consider the team and human factors that influence its success. To identify the major facilitating factors for the success of refactoring, we interviewed 10 industrial software developers, and combined their responses with a study of the existing literature, formulated a model of refactoring success. The resulting conceptual model comprises both technical and non-technical factors. Technical factors include: level, testing and debugging, and tools, while the non-technical factors include: communication and coordination, support activities, individual capability/skills, and programmer participation. We propose to verify this model empirically through a survey of professional software developers (main body of refactoring practitioners). The survey design is provided.

Keywords

Refactoring, software development, empirical software engineering, maintenance, human factors

INTRODUCTION

Refactoring, as a software engineering method used to incrementally improve the design of existing code, is being increasingly adopted in industrial software development. Considerable numbers of software engineers already use refactoring in their daily developments (Xing and Stroulia, 2006). Meanwhile, mainstream software development environments, such as Eclipse[1] and Visual Studio. net[2], now provide semi-automated refactoring modules, and thus further facilitate the adoption of refactoring in industrial software development environments. The value proposition of refactoring is its power to reconstruct existing software according to well-defined mechanics and principles, hence reversing the software decay process caused by traditional development methods (Fowler, 1999). Refactoring is now also a baseline approach of agile software development methodology (Beck, 2000) and plays an important role in open source development practices. All these developments suggest that refactoring is an important aspect of software design, whose impact is going to grow in future.

Despite the popularity and importance of refactoring in industrial software development, much remains unknown about the factors that influence the success of refactoring practices. Specifically, most research on refactoring has focused on technical aspects but has ignored human factors. However, refactoring, as most software development methods, is human-centric in nature. As software artifacts are created by and for human beings, human and organizational factors also play an important role in the success use of a specific development approaches; the human dimension is at least as important as the technical dimension (Constantine, 2001). Although several works (e.g. Murphy *et al*, 2006) addressed this point, studies of behavioral factors in refactoring are often organized in an ad hoc way and have not generated the needed empirical evidence. Therefore

[1] http://www.eclipse.org
[2] http://msdn.microsoft.com/en-us/vstudio/products/default.aspx

more and better formalized empirical research is necessary to improve our understanding of refactoring activities. Besides, identifying the key factors for refactoring success is a promising way to bring benefits to future software development practices. This can help practitioners to better deal with issues in refactoring and avoid potential failures.

This paper proposes an empirical study attempting to identify the key factors that influence the success of refactoring practices in industrial software development. Through a combination of literature review and 10 interviews, we built a conceptual model of the key factors affecting refactoring success. The model not only contains the technical factors but also human and team factors which so far have not been considered. We plan to conduct a survey to software developers to test this model in a subsequent research stage.

The rest of this article is organized as follows. Section 2 provides the formal definition of refactoring, and a brief literature review. Section 3 presents the conceptual model, which comprises seven independent, one dependent and one moderating variable. The corresponding research hypotheses are also introduced. Section 4 suggests a survey design and measurement instruments for future data collection. Section 5 draws conclusions.

RESEARCH BACKGROUND

What is Refactoring?

In a real-world environment, software needs to evolve continuously. However, with the evolution of the software systems, their source code becomes more complex and drifts away from its original design, thereby lowering the quality of the software. Therefore, maintenance becomes the most resource-consuming task during the whole software life cycle. This problem cannot be solved through better programming environments. While such programming environments may automate some development or maintenance tasks, they introduce new features to the system, making the software system even more complex.

Refactoring is a technique exactly applicable to this problem. The term *"refactoring"* was formally introduced to software engineering by William Opdyke in his PhD dissertation (Opdyke, 1992), but the underlying methods likely has been practiced informally for as long as programmers have been writing programs. According to Opdyke, refactoring refers to: *"the process of changing a (object-oriented) software system in such a way that it does not alter the external behaviors of the code, yet improves its internal structure"*. The potential benefits of refactoring include reduced complexity and increased comprehensibility of the code. Improved comprehensibility makes software maintenance relatively easy and thus provides both short-term and long-term benefits. Refactoring was popularized by Martin Fowler (1999). Fowler developed a rich catalog of refactoring scenarios and methods, each refactoring scenario capturing a structural change observed repeatedly in various languages and application domains.

Research on Refactoring

The academic and industrial literature has recognized refactoring, introducing not only new methods but also extending refactoring to new programming environments and paradigms. Roberts (1999) built a formal model and the most comprehensive tool, the Refactoring Browser for smalltalk. Rura (2003) extended the applicability of refactoring from Object-Oriented Programming (OOP) to Aspect-Oriented Programming (AOP). Wloka (2008) explored tool-supported refactoring for aspect-oriented programs. Some studies focused on assessing the influence of refactoring on program quality (e.g. Sands, 1996). Other studies tried to provide formalisms to verify refactoring's "preservation principle" (no change in functionality) through using Hoare's logic (Opdyke, 1992; Roberts, 1999; Mens, 1999). Still other research created guidelines to help programmers to decide whether they need to refactor their code or not (e.g. Dudziak and Wloka, 2002), or discussed the inter-relationships between different refactoring approaches (Counsell, 2008).

Besides these technology focused research streams, a few empirical studies paid attention to the actual use of refactoring in industrial software development. Opdyke, for example, spent one chapter (chapter 13) to discuss this issue in Fowler's book (Fowler 1999), however, he addressed the problem without empirical evidence. Murphy *et al.* (2006) provided some empirical information on refactoring when discussing the use of Eclipse among Java developers. Based on Murphy's data, Murphy-Hill and Black (2008a) summarized principles for tool design and selection. In another paper, Murphy-Hill and Black (2008b) further demonstrated three new refactoring tools and provided a brief user study. Nevertheless, the empirical study base regarding refactoring is still limited, and lacks a formal, model based approach to the analysis of refactoring practice.

CONCEPTUAL MODEL AND HYPOTHESES

Conceptual Model Development

Seeing the establishment of a formal framework for refactoring success, we began to conduct empirical research on refactoring use in an industrial context in the second half of 2008. To accomplish our goal, we first analyzed the related literature including empirical software engineering, organizational and team research studies. Through the literature study, we identified possible factors that influence the process and results of refactoring. Combining these factors, we developed a comprehensive conceptual framework of refactoring. Between September and November in 2008, we conducted 10 interviews with experienced refactoring practitioners (4 mangers and 6 developers/testing specialist). Based on the data we gathered from the informants, we refined the conceptual framework extracting the most important factors, hence generating a more comprehensive framework.

The resulting research model thus derives its theoretical foundations by combining prior literature review on software engineering, organizational and team research, along with an explorative empirical study of 10 project/program managers, developers, and testing specialists.

The framework is depicted in figure 1. It contains seven facilitating factors along two dimensions: a technical dimension, and a team and human dimension. The technical dimension contains three factors: level, testing and debugging, and quality of tools. The team and human dimension includes another four factors: communication and coordination effectiveness, support activities, programmers' participation, and individual capability/skills. The model also contains one moderating factor (environmental conditions), and the dependent variable, refactoring success, which contains two items: perceived level of success and success of software artifacts.

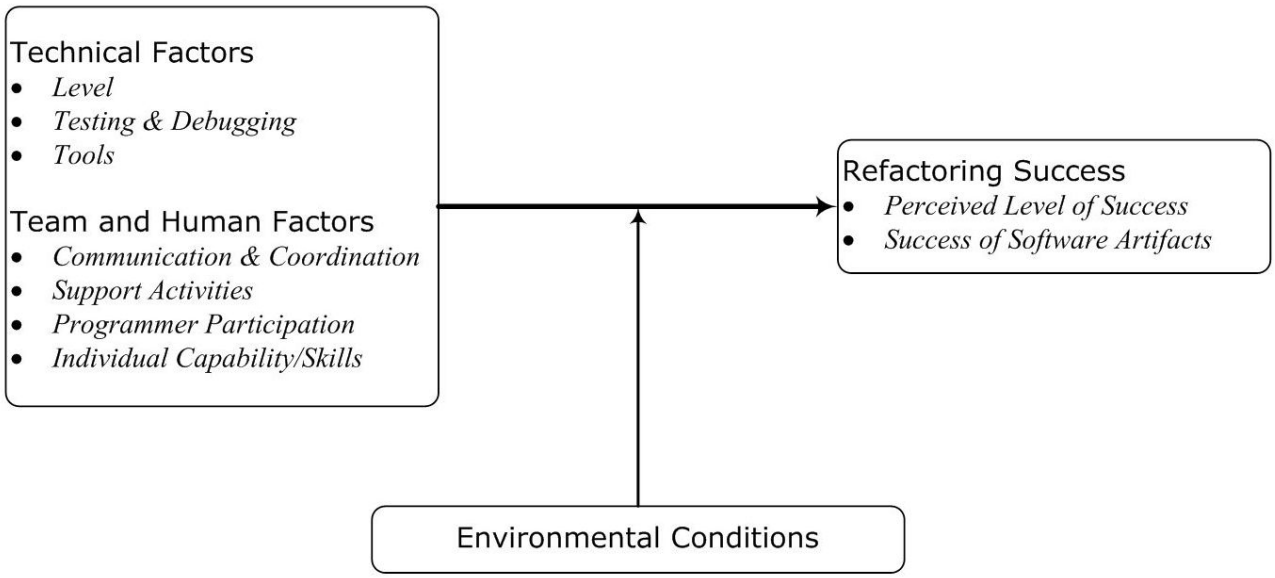

Figure 1. Conceptual Research Model

Hypotheses Development for Technical Factors

Level

"Level" is a characteristic concept of refactoring. Refactoring can be divided into two categories: Low level and high level (Opdyke 1992, Roberts 1999). Low level (primitive) refactoring is fine-grained (at the class or interface level), while high level refactoring operates at coarser granularity (major design changes), which can be achieved through composition of low level refactoring activities.

The idea of decomposition and composition in software development dates back to early 1970s. Parnas' modular design (Parnas, 1972) decomposes complex system design into simple modules that are easy to manipulate. This is similar to refactoring composition. The philosophical foundation for this kind of composition is the *principle of compositionality* in semantics, which translates into *compositionality in programming languages* in computer science. If low level (naïve) components are composed according to the compositionality principle, the overall high level result are expected to be favorable. Our prior interviews also revealed the developers treat *"level"* with high importance. According to the interview results, 8 out of 10 interviewees indicated *"level"* is an important issue in refactoring practice.

Therefore, the correctness of low level composition is assumed to be an essential factor for high level design improvement. Determining the proper low level composition mechanism should be important for successful high level refactoring.

Hypotheses 1. *Refactoring Success is positively associated with the correctness of low level refactoring composition.*

Testing and Debugging

Testing and debugging are important and time consuming tasks in software development (Gelperin and Hetzel, 1988). Due to the complexity of software, people can not completely test a program of moderate complexity (Beizer, 1990). Furthermore, there are no direct quantitative links between testing and software quality. For example, although the importance of the testing and debugging in software development is indisputable, one can never claim that the 10 hours testing definitely brings better or bug-free software than 1 hour testing does. Testing and debugging can help software developers determine whether the software systems fulfill their requirement specifications (in both functional and non-functional aspects) and find potential errors, hence bring the developers more confidence concerning the correctness of their programs.

Due to the source code changes in refactoring, distortions to the original design may be introduced accidentally, which constitute a violation the basic premise of refactoring (behavior preservation). Testing is the major technique used in practice to identify and avoid this violation. In refactoring practice, satisfying the original test cases is compulsory (Dinh-Trong *et al*, 2008). Furthermore, the refactoring process may also introduce new bugs, and hence, finding them through debugging is also important. Altogether, testing and debugging can enhance the programmer confidence towards their code and potentially bring better software deliverables (Pipka, 2002). Six of the 10 interviewed software developers and managers identified *"testing and debugging"* as an important criterion.

Hypotheses 2. *Refactoring Success is positively associated with the intensity and the effectiveness of testing and debugging.*

Tools

It is nowadays almost impossible to develop any software systems completely manually in an industrial software development environment (Schmitt, 1991, Erickson, 1996). CASE (Computer Aided Software Engineering) tools that provide some automatic support for the software development process are the necessary choice for every software developer. From a design theory perspective, the software development needs to use appropriate tools (Goel and Pirolli, 1992). Similarly, tools are important for the refactoring activities. Some semi-automatic tools (e.g. Eclipse) have integrated major primitive refactoring techniques among their standard functions. Besides, as testing and debugging tools also widely adopted in the refactoring practice, some high level design and reverse engineering tools are also used for refactoring.

Empirical results have demonstrated the importance and high frequency of tool use in refactoring. For example, Murphy *et al.* (2006) provide some empirical information concerning refactoring tool use frequency in the context of Eclipse IDE application among Java developers. Xing and Stroulia (2006) also conducted a case study involving the Eclipse IDE and determined the need for more powerful refactoring tools. Seven of the 10 interviewed software developers and managers addressed the importance of the quality of development tools, and treated it as a key factor for refactoring success.

Based on the above discussion, it is reasonable to assume that refactoring and other related software engineering tools can help practitioners in the refactoring process, and can facilitate the overall success of refactoring.

Hypotheses 3. *Refactoring Success is positively associated with the quality of software development tools.*

Hypotheses Development for Team and Human Factors

Communication and Coordination

One of the main features of software development is its collaborative nature. In collaborative teams, effective communication and coordination (e.g., interacting often, updating one another, discussing issues openly, and conveying all necessary information to one another) is crucial to team and personal achievements (Curtis *et al*., 1988). From the theoretical

perspective of teamwork quality (TWQ), communication and coordination are among the six major facets of the TWQ constructs (Hoegl and Gemuenden, 2001). Furthermore, Convey's law holds within the software development domain, which proclaims that *"any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure."* (Herbsleb, 1999). The importance of communication can be directly derived from this well accepted law.

The importance of communication has also been documented through the development of various models (e.g. Sealman, 1997) in software design and/or other design domains. Furthermore, many empirical studies (e.g. Kraut & Streeter, 1995; Wu *et al*, 2003; Cataldo, 2006) provided detailed descriptions of communication and coordination activities in software development. In recent years, intra-team communication and coordination has become even more important for the success of geographically distributed development (Herbsleb, 2005).

As a typical software engineering approach, refactoring needs effective communication and coordination. Many refactoring tasks are divided up among a group of developers, and all of them have to work from an existing code base whose design principles can often be only understood through discussion with the original developers. Ineffective communication and coordination may thus lead the effort overlaps, task omissions, or general misunderstanding, and hence can impair the overall success of the refactoring. Eight of the 10 interviewed software developers and managers identified communication and coordination as a key factor for refactoring success. Thus, we argue that communication and coordination is a key factor to facilitate the successful refactoring practice in an industrial context.

Hypotheses 4. *Refactoring Success is positively associated with h the effectiveness of communication and coordination in the refactoring process.*

Support Activities

Support Activities refer to the mutual help provided between team members, including knowledge sharing and reviews (inspections) at the team level, as well as individual acts of help from other team members. It also includes activities that aim to improve team cohesion and the interpersonal relationships between team members. Tjosvold (1995) pointed out that, for interdependent tasks, mutual support is more productive than the forces of competition. Thus, in a team high in frequency of support activities, team members often display mutual respect, grant assistance when needed, and develop other team members' ideas and contributions rather than trying to outdo each other. Competitive behaviors in a team lead to distrust and frustration, whereas mutual support fosters the integration of team members' expertise and is a critical aspect of the quality of collaboration in teams. All four interviewed managers treated *"support activities"* as an important factor for the team level success.

Empirical research suggests that team activities have an influence on team performance. For example, Guinan *et al*. (1998) pointed out that team experience enables more effective team processes than do software development tools and methods. Other studies such as Sabherwal *et al*. (2006) also provided evidence for the influence of team activities on team performance and project success.

Hypotheses 5. *Refactoring Success is positively associated with the effectiveness of the support activities in the software development team.*

Programmer Participation

Within the scope of our study, programmers' participation is defined as the extent to which the programmers use their skills to decide, act and take the responsibility for refactoring. Employee participation has been treated as a critical factor in organizational/team performance for a long time. Mayo's studies (1945) on productivity at the Western Electric's Hawthorne plant provided groundbreaking results in this area. From the perspective of socio-technical systems, business organizations should seek the best match between the requirements of independent social and technical system (Cataldo, 2008). In other words, people participation should match the level of technical developments in business organizations.

With the broad adoption of refactoring, if programmers' participation cannot fulfill the requirement of the refactoring technology, it thus fails to meet the requirement of social-technical congruency. This may impair competitive advantages at organizational and team levels (Conradi *et al*, 2001). If the programmers' participation is excellent, however, programmer performance will also benefit from this culture of excellence. Approaching the issue of culture differently, organizations might motivate their employees by instilling the belief that it is the *responsibility of being programmer*s to refactor code. Creating this kind of responsibility is especially important for teams that treat refactoring as an infrequently used development policy. Seven of the 10 interviewed software developers and managers addressed the importance of this issue in the software development practice.

In summary, organizations and teams should offer opportunities and encourage programmers' participation. This should help programmers to improve their work and should create an atmosphere of contribution to the team and its mission. It can also facilitate the programmers' responsibility and hence contribute to the success of refactoring.

Hypotheses 6. *Refactoring Success is positively associated with programmers' participation.*

Individual Capability/Skills

To build high quality software, people with good *programming* skills are needed; to collaborate with other individuals, one also needs to have good *interpersonal* skills (Acuña *et al*, 2006). Thus both hard and soft skills are important in software development practice. Consequently, finding the right people for a software development project is always a crucial issue in software organizations.

In addition, from the perspective of human resource management and knowledge management, an individual's knowledge greatly affects the success of projects and organizations. MacKinnon (2007) demonstrates this with evidence for the relationship between individual skill growth/decay and personal performance.

Refactoring, as typical knowledge intensive activity, needs practitioners to have enough individual capability and skills. Therefore, programmer capability and skills are also important for the success of refactoring. Nine out of the 10 interviewed software developers and managers indicated the criticality of individual capability and skills to the refactoring process.

Hypotheses 7. *Refactoring Success is positively associated with the practitioners' individual capability/skills.*

RESEARCH METHODOLOGY AND MESUREMENTS

Survey Design

We plan to employ a survey to gather data from a broad sample of the software development professionals. Therefore, software developers and software project managers are considered as the target population for this study. Based on our 10 prior interviews, we are aware of potential differences in the perceptions of these two subject groups and will evaluate the responses accordingly. We will exclude individuals acting in senior managerial roles, because senior level managers generally do not participate in software development and maintenance activities directly. The process of survey design follows the SEI guideline of survey design[3] strictly.

The selection of organizations and subjects will follow a random sampling process. After the sample selection, we will try to contract the potential informants to encourage them to answer the questionnaire. The survey will be focused on the Shanghai area, a region with more than 30 million inhabitants and the most developed software development industry in China. Shanghai is also one of the most diverse areas in China, which ensures the wide diversity of this survey. Shanghai is also home to numerous multi-national and joint venture software development firms, including IBM, SAP, and Oracle, who maintain research and development centers there.

Variables and Measures

Based on the definitions of the factors described in the conceptual model (figure 1), one dependent, seven independent, and one moderating variables will be used to collect the data in this study.

Independent Variables

To measure the extent to which each of the six independent variables is perceived, we will define multi-item, five-point, bipolar Likert scales that range from "strongly disagree" (1) to "strongly agree" (5) for all indicators. Item ratings will be aggregated to form a summative rating scale for each independent variable. The items for each variable's are based on prior literature, modified in part to fit for the objectives of this study. The independent variable measures are introduced respectively as follows.

Level. Level is a unique feature of refactoring. Since this is the first study of its kind with refactoring, new items while formulated specifically for measuring this variable.

[3] www.sei.cmu.edu/pub/documents/05.reports/pdf/05hb004.pdf

Tools. The evaluation of software development tools is still under intensive research. Although it is not possible to find a universal criterion to assess all software development tools (Nahouraii and Kavi, 1996), an assessment of most software development tools often can be achieved according to the aspects of accessibility, usability, functionality, support, and extendibility (Fisher, 1988; Brown, 1994). We will thus introduce these five items to measure the *Tools* variable.

Testing and Debugging. Testing and debugging are persistently important topics in software engineering. Given the significant breadth of research on testing and debugging, it is impossible to cover all corresponding issues in our survey. Therefore, we will focus on the basic rules in testing and debugging, derived from best practices and international standards on these two issues (e.g. IEEE, 1998; Grötker *et al*, 2008).

Communication and Coordination Effectiveness. Effective coordination of contributions from different functional areas is of critical importance to the successful design and implementation of innovations. To capture this criticality, communication effects are measured partly based on the items presented Pinto *et al.* (1990). In Pinto's work, 10 items are used to measure communication, with further factor analysis yielding three factors that all together explain almost 90% of the variance. For coordination impact on success, measurement items will be generated through combining Hoegl and Gemuenden's (2001) work on TWQ and Espinosa *et al*.'s (2007) work on coordination mechanism in distributed software development.

Programmer participation. The items for programmer participation were derived mainly from Dyba's (2005) empirical studies on key factors for success in software process improvement. Dyba created 7 items to measure the programmer participation towards SPI. We will exclude the specific items for the SPI and will revise the remaining ones according the features of refactoring.

Individual Capability/Skills. There are numerous objective methods to test the individual's capability and skills. However, to keep the data collection process feasible, we adopt a self-reported approach here, taking into account that software engineers often know considerably more about their colleagues than test scores would reflect, especially with respect to soft skills. To assess individual capability and skills, we will divide survey participants according to their different job roles into two categories (project managers and developer), as previously mentioned.

Dependent Variable

The assessment of the dependent variable follows the same rationale. We will again use multi-item, five-point, bipolar Likert scales for the dependent variable indicators. For perceived refactoring success, five items will be used. For the success of software artifacts, several objective criteria can be directly applied. An item for *behavior preservation* will be added to reflect this unique feature of refactoring.

Environmental conditions

Environmental conditions are included in the study to capture the most influential sources of variation in software organizations (Dyba, 2000). Environmental conditions will be measured using three semantic differential items, which are rated on a five-point scale. Two items will be adopted from Dyba's study, namely stability ("*stable versus unstable*") and predictability (*"predictable versus unpredictable*"). We will also add a third item to describe the process, which is controllability ("*controllable versus uncontrollable*").

SUMMARY AND CONCLUSION

In this paper, we described a study in progress that aims to model and confirm the key factors influencing refactoring success in industrial software development. An analysis of the existing literature identified the value of and need for a formal, model-based approach to the determination of refactoring success. Consequently, based on a literature study and interviews with 10 refactoring practitioners, we developed a conceptual model containing seven facilitating factors and encompassing both a technical and a human dimension. Corresponding hypotheses developments were also introduced. We further briefly outlined the survey design and measurement instruments. It is our expectation that a test of the model will reveal new and unique insights concerning the success of refactoring efforts. From the theoretical perspective, the proposed formal model adds an important new dimension to refactoring research for the identification of the importance of human factors for refactoring success.

In reflecting on the model, we must also question whether it possibly omits important variables that could determine software refactoring success. This cannot be excluded, but due to our dual process of literature search and developer interviews, is therefore less likely. Specifically, we did exclude variables from an earlier, more encompassing model that covered among others, task complexity, availability of resources, and barriers. These variables were not supported by interview responses.

Conceptually, other important variables could have an influence, but are beyond the scope of this model. For instance, organizational factors, such as organizational culture and reward systems could be of influence. Similarly, the maturity of the software development process could have an influence, or whether the refactorer is the author of the original code. We expect to control for these aspects, for instance by choosing organizations that promote refactoring as part of their organizational practice, and by focusing on modern software development practices. Clearly, this will require more reflection on appropriate control variables, an exercise beyond the scope of this short article.

ACKNOWLEDGMENTS

The authors want to show their appreciations to all the participants of this study. They also want to thank the anonymous reviews of this paper for their insightful comments.

REFERENCES

1. Acuña S.T Juristo, N., and Moreno, Ana M. (2006) Emphasizing human capabilities in software development, *IEEE Software*, no. 9, 94-101.
2. Beck, K. (2000). Extreme programming explained: embrace change, Addison Wesley.
3. Beizer, B. (1990) Software testing techniques. Second edition.
4. Brown, A.W. (1994) Why evaluating CASE environments is different from evaluating CASE Tools, *Proceedings of the third symposium Assessment of Quality Software Development Tools*, IEEE Computer Society, 4-13.
5. Cataldo, M., Wagstrom, P., Herbsleb, J.D., Carley, K. (2006) Identification of coordination requirements: Implications for the design of collaboration and awareness tools, *Proceedings of ACM Conference on Computer-Supported Cooperative Work,* Banff Canada, 353-362.
6. Cataldo, M., Herbsleb, J.D., Carley, K. (2008) Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. ESEM 2008, 2-11.
7. Conradi, R., and Dyba, T. (2001) An empirical study on the utility of formal routines to transfer knowledge and experience, Pro*ceedings of Joint Eighth European Software Engineering Conference (ESEC) and Ninth ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE).*
8. Constantine, L. (2001) Peopleware Papers: The notes on the human side of software, Prentice Hall.
9. Counsell, S., and Swift, S. (2008). "Refactoring steps, Java refactorings and empirical evidence," *Proceedings of 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2008),* pp.176-179, IEEE Computer Society.
10. Curtis. B., Krasner, H., and Iscoe, N. (1988) A field study of the software design process for large systems. *Communications of the ACM*, vol. 31, no. 11, 1268-1287.
11. Dinh-Trong, T., Geppert, B., Li, J.J., and Roessler, F. (2008). Looking for more confidence in refactoring? How to assess adequacy of your refactoring Tests, *Proceedings of the 8th International Conference on Quality Software, IEEE Computer Society.*
12. Dudziak, T., and Wloka, J. (2002) Tool-supported discovery and refactoring of structural weak-nesses in code, M.S. thesis, Faculty of Computer Science, Technical University of Berlin, February 2002.
13. Dyba, T. (2000) "Improvisation in small software organizations," *IEEE Software*, vol. 17, no. 5, 82-87.
14. Dyba, T. (2005) An empirical investigation of the key factors for success in software process improvement, *IEEE Transactions on Software Engineering*, Vol.31, No.5, 410-424.
15. Erickson, T. (1996) Methods and tools: Design as storytelling, *Interactions*, vol. 3, no. 4, 30-35.
16. Espinosa, J. A., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D. (2007) Team knowledge and coordination in geographically distributed software development, *Journal of Management Information Systems*, 24(1), 135-169.
17. Fisher, Alan S. (1988) CASE: Using Software Development Tools, Wiley.
18. Fowler, M. (1999) Refactoring: Improving the design of existing programs, Addison-Wesley.
19. Gelperin, D., and B. Hetzel. (1988) The growth of software testing, *Communications of ACM*, Vol 31, No.6.
20. Goel, V. and Pirolli, P. (1992) "The structure of design problem spaces, Cognitive Science, vol. 16, 395-429.
21. Grötker, T., Holtmann, U., Keding, H., and Wloka, M. (2008) The Developer's Guide to Debugging, Springer.

22. Guinan, Patricia J., Cooprider, J G., Faraj, S. (1998) Enabling software development team performance during requirements definition: A behavioral versus technical approach, *Information System Research*, vol. 9, no. 2, 101-125.
23. Guindon, R. (1990) Designing the design process: exploiting opportunistic thoughts, *Human-Computer Interaction*, vol. 5, 305-344.
24. Herbsleb, J.D., and Grinter, R.E. (1999) Splitting the organization and integrating the code: Conway's law revisited," *Proceedings of 21st International Conference on Software Engineering (ICSE),* 85-95.
25. Herbsleb, J.D., Paulish, D.J., & Bass, M. (2005). Global software development at Siemens: Experience from nine projects. *Proceedings of 27th International Conference on Software Engineering (ICSE),* St. Louis, MO*,* May 15-21, 524-533.
26. Hoegl, M., and Gemuenden, H. G. (2001) Teamwork quality and the Success of Innovative Projects: A theoretical concept and empirical evidence, *Organization Scienc*e, Vol. 12, No. 4, 435–449.
27. IEEE. (1998) IEEE standard for software test documentation. New York, IEEE.
28. Kraut, R.E. and Streeter, L. (1995) Coordination in software development, *Communications of the ACM*, vol. 38, no. 3, 69-81.
29. MacKinnon, D. (2007) How individual skill growth and decay affect the performance of the project organizations, PhD Dissertation, Stanford University.
30. Mayo, E. (1945) The social problems of an industrial civilization. Boston: Harvard Univ. Press.
31. Mens, T. (1999). "A formal foundation for object-oriented software evolution," Ph.D. thesis, Department of Computer Science, Vrije Universiteit Brussel, Belgium, September 1999.
32. Mens, T., Tourwé, T. (2004) A survey of software refactoring, *IEEE Transaction on Software Engineering*, Vol. 30, No. 2, Feb. 2004 pp.126-139.
33. Murphy-Hill, E., Black, A.P. (2008a) Refactoring Tools: Fitness for Purpose, *IEEE Software*, Vol. 25, No. 5, Sept.-Oct. 2008, 38 – 44.
34. Murphy-Hill, E., and Black, A. P. (2008b) Breaking the barriers to successful refactoring: Observations and tools for extract method, *Proceedings of 30th International Conference on Software Engineering*, Leipzig, Germany, May 2008. IEEE Computer Society.
35. Murphy G.C., Mik Kersten, and Leah Findlater. (2006) How are Java software developers using the Eclipse IDE?, *IEEE Software*, Vol. 23, No. 4, 2006, pp. 76-83.
36. Nahouraii, E., and Kavi, K. (1996) Software tools assessment, *IEEE Software*, Volume: 13, Issue: 5, 23-26.
37. Opdyke, W. F. (1992). Refactoring: A program restructuring aid in designing object-oriented application frameworks, Ph.D. thesis, University of Illinois at Urbana-Champaign,
38. Parnas, D.L., A technique for software module specification with examples, *Communications of the ACM*, 15, 5, May 1972, 330-336.
39. Pipka, J. U. (2002). Refactoring in a "test first"-world, *Proceedings of the 3rd International conference on agile process and eXtreme Programming in software engineering (XP 2002).*
40. Pinto, M. B., and Pinto, J. K. (1990). Project team communication and cross functional cooperation in new program development. *Journal of Product Innovation Management,* no. 7, 200–212.
41. Roberts, D. (1999) Practical analysis for refactoring, Ph.D. thesis, University of Illinois at Urbana-Champaign.
42. Rura, S. (2003) Refactoring aspect-oriented software, Technical Report in Computer Science. Williamstown, Massachusetts, Williams College.
43. Sabherwal, R., Jeyaraj, A., and Chowa, C. (2006) Information system success: Individual and organizational determinants, M*anagement Science,* vol.52, no.12, 1849-1864.
44. Sands, D. (1996) Total correctness by local improvement in the transformation of functional programs, *ACM Transaction on Programming Languages and Systems,* vol. 18, no. 2, March 1996, ACM, 175–234.
45. Schmitt, G.N. and Chen, C.C. (1991) Classes of design - classes of methods - classes of tools, *Design Studies*, vol. 12, no. 4, 246-251.
46. Seaman, C. B. and Basili, V. R. (1997) Communication and organization in software development: An empirical study, *IBM Systems Journal*, vol 36, no. 4.

47. Tjosvold, D. (1995) Cooperation theory, constructive controversy, and effectiveness: Learning from crisis, R. A. Guzzo, E. Salas and Associates, eds. *Team Effectiveness and Decision Making in Organizations*. Jossey-Bass, San Francisco, CA, 79–112.
48. Wloka, J., Hirschfeld, R., and Hänsel, J. (2008) Tool-supported refactoring of aspect-oriented programs, P*roceedings of the 7th International Conference on Aspect-Oriented Software Development (AOSD 2008)*, Brussels, Belgium, ACM, 132-143.
49. Wu, J., Graham, T.C.N., and Smith, P.W. (2003) A study of collaboration in software design, P*roceedings of the 2003 International Symposium on Empirical Software Engineering*, IEEE Computer Society.
50. Xing ZH, and Stroulia, E. (2006) Refactoring Practice: How it is and how it should be supported - An Eclipse Case Study. In *Proceedings of International Conference on Software Maintenance* (ICSM 2006), 458-468.