validation of this long list of refactorings is time consuming. Thus, they appreciate that DINAR suggests refactoring one by one and updates the list based on developer feedback.

4. THREATS TO VALIDITY

Some threats need to be considered when interpreting our study results.

The first threat is the limited number of subjects and evaluated systems (11 participants and 5 systems), which externally threatens the generalizability of our results. In addition, we cannot conclude that DINAR can perform very well also when helping developers since our study was limited to the use of 23 types of refactorings and evaluating 8 types of code smells. Future replications of this study are necessary to confirm our findings.

The second threat is related to the variation of correctness and speed between the different groups when using DINAR and other tools such as JDeodorant. In fact, DINAR may not be the only reason for the improved performance because the subjects have different programming skills and familiarity with refactoring tools. However, developers were not assigned to groups randomly but according to their programming experience to reduce the gap between the different groups. Another threat concerns the data about the actual refactorings of the studied systems. In addition to the documented refactorings, we used Ref-Finder which is known to be efficient. Indeed, Ref-Finder was able to detect refactoring operations with an average recall of 95% and an average precision of 79% [11]. To ensure the precision, we manually inspect the refactorings found by Ref-Finder by randomly selecting a set of detected changes and evaluate them by the participants of our experiments.

5. CONCLUSION AND FUTURE WORK

We proposed, in this paper a novel interactive recommendation tool, called DINAR, for software refactoring that dynamically adapts and suggests refactorings to developers based on their feedback and introduced code changes. To evaluate the effectiveness of DINAR, we conducted a human study with 11 software developers who evaluated the tool and compared it with the state-of-the-art refactoring techniques. Our evaluation results provide strong evidence that DINAR improves the applicability of software refactoring and proposes a novel way for software developers to refactor their systems.

Future work should validate DINAR with additional refactoring types, new systems and code smell types in order to draw conclusions about the general applicability of our methodology. We will also compare DINAR with other refactoring techniques [14][15]. Furthermore, in this paper, we only focused on the recommendation of refactorings. We are planning to extend the approach by automating the test and verification of applied refactorings. In addition, we will consider the importance of code smells during the correction step using previous code changes, class complexity, etc. Another future research direction related to our work is to adapt our interactive refactoring recommendation tool to several other software engineering problems such as software remodularization, change detection and the next release problem.

ACKNOWLEDGEMENTS

This work was supported, in part, by the Institute for Advanced Vehicle Systems-Michigan grant and the Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre.

6. REFERENCES

[1] Basseur, M. Liefooghe, A. Le, K. and Burke, E. K. 2012. The efficiency of indicator-based local search for multi-objective combinatorial optimisation problems. *Journal of Heuristics.* vol. 18, issue 2, pp 263-296.

[2] Fowler, M. Beck, K. Brant, J. Opdyke, W. and Roberts, D. 1999. Refactoring – Improving the design of existing code. Addison Wesley, ISBN 978-0201485677.

[3] Murphy-Hill, E. R. Parnin, C. and Black, A. P. How we refactor, and how we know it. *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 5–18, 2012.

[4] Ouni, A. Kessentini, M. Sahraoui H. and Boukadoum, M. 2012. Maintainability Defects Detection and Correction: A Multi-Objective Approach. *Journal of Automated Software Engineering*, Springer. vol. 20, issue 1, pp 47-79.

[5] Harman, M. and Tratt, L. Pareto optimal search based refactoring at the design level. *GECCO07.* pp. 1106-1113.

[6] Kessentini, M. Kessentini, W. Sahraoui, H. Boukadoum, M. and Ouni, A. 2011. Design Defects Detection and Correction by Example, 19th IEEE International Conference on Program Comprehension. pp. 81-90.

[7] Deb, K. Pratap, A. Agarwal, S. and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *TEC*. vol. 6, pp. 182–197.

[8] Harman, M. Mansouri, S. A. and Zhang, Y. Search-based software engineering: 2012. Trends, techniques and applications. *ACM Computing Surveys*, vol. issue 45, no. 1.

[9] Deb, K. and Srinivasan, A. Innovization: innovating design principles through optimization. 2006. *GECCO.* pp. 1629-1636.

[10] Bansiya, J. and Davis, C. G. 2002. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*. vol. 28. pp. 4–17.

[11] Prete, K. Rachatasumrit, N. Sudan, N. and Kim, M. 2010. Template-based reconstruction of complex refactorings. *Proceedings of the 26th IEEE International Conference on Software Maintenance*. pp. 1–10.

[12] Hall, M. Walkinshaw, N. McMinn, P. 2012. Supervised software modularization. *ICSM12.* pp. 472-481.

[13] Bavota, A. Carnevale, F. De Lucia, A. and Di Penta, M. 2012. Putting the Developer in the Loop: An Interactive GA for Software Re-modularization. *SBSE*. pp. 75-89

[14] Lucia, Lo, D. Jiang, and L. Budi, A. 2012. Active refinement of clone anomaly reports. *International Conference on Software Engineering.* pp. 397-407.

[15] Gong, L. Lo, D. Jiang, L. and Zhang, H. 2012. Interactive fault localization leveraging simple user feedback. *ICSM2012.* pp. 67-76.

[16] Fokaefs, M. Tsantalis, N. Stroulia, E. and Chatzigeorgiou, A. 2011. JDeodorant: identification and application of extract class refactorings. *ICSE2011.* pp. 1037-1039.

[17] Kessentini, W. Kessentini, M. Sahraoui, H. Bechikh, S. and Ouni, A. 2014. A Cooperative Parallel Search-Based Software Engineering Approach for Code-Smells Detection, *IEEE Transactions on Software Engineering*, 2014, to appear.