Guest Editorial: Special Issue on Program Comprehension

1. Program Comprehension

Program comprehension is a vital software engineering and maintenance activity. It is necessary to facilitate reuse, inspection, maintenance, reverse engineering, reengineering, migration, and extension of existing software systems. The International Conference on Program Comprehension (formerly known as IWPC) provides a high-quality venue for researchers and industry practitioners to present and discuss both the state-of-the-art and the state-of-the-practice in the general area of program comprehension.

IWPC typically has a technical papers track in parallel sessions, several working sessions, and in recent years an additional tool demonstration session in which the latest research prototypes are presented to a large audience. The number of participants typically ranges from 80 – 100 attendees every year. IWPC is in its transition phase from a workshop to an international conference and in 2006 it has changed to ICPC – the International Conference on Program Comprehension.

In 2005, there were 54 technical paper submissions (the largest number of submissions to IWPC). Each submission was reviewed by at least three members of the program committee (and many papers received four reviews). On the basis of these reviews the Program Chairs accepted 24 full papers and four short papers for inclusion in the proceedings and presentation in the technical program.

2. The Special Issue

This special issue features a selection of best papers of IWPC 2005, held in St. Louis, Missouri, USA. We invited the authors of seven papers to submit extended versions for consideration to this special issue. After re-review by three or four international experts in the field, four papers were selected for inclusion. All four selected papers were significantly revised and extended. The four papers are summarized below and represent a good cross-section of topics discussed at IWPC 2005.

The paper entitled “Theories, Methods and Tools in Program Comprehension: Past, Present and Future” by Margaret-Anne Storey reviews several of the key cognitive theories of program comprehension that have emerged over the past many years of research in program comprehension. Using these theories as a canvas, the paper then explores how tools that are popular today have evolved to effectively support program comprehension. As an outlook to the future, the paper attempts to predict how human characteristics and program characteristics will change in the future and influence program comprehension tool development and research methods.

The paper entitled “Applying and Combining Three Different Aspect Mining Techniques” by M. Ceccato et al. addresses program understanding at the source-code level by identifying cross-cutting concerns. The authors first apply three independently developed aspect mining techniques to a case study and evaluate and compare their results. Based on this analysis, the paper presents three interesting combinations of these three techniques, and shows how these combinations provide a more complete coverage of the detected concerns as compared to the original techniques individually.

The paper entitled “Evaluating the Layout of UML Diagrams for Program Comprehension” by Ken Wong and Dabo Sun addresses the problem that many algorithms and tools have been developed to generate UML diagrams automatically for program understanding, but they often ignore perceptual factors in the layout of these diagrams. This paper presents key criteria and guidelines for the effective layout of UML class and sequence diagrams from the perspective of perceptual theories such that users can avoid spending too much time on rearranging boxes and lines to make a particular diagram understandable.
The paper entitled “Concise and Consistent Naming” by Florian Deissenboeck and Markus Pizka addresses the fact that about 70% of source code consists of identifiers. This paper presents a formal model, based on bijective mappings between concepts and names, and provides a solid foundation for the definition of precise rules for concise and consistent naming. The enforcement of these rules is supported by a tool that incrementally builds and maintains a complete identifier dictionary while the system is being developed. The identifier dictionary explains the language used in the software system, aids in consistent naming, and improves productivity of programmers by proposing suitable names depending on the current context.

3. Acknowledgements

We are greatly appreciative of all those individuals who helped in the organization and realization of this special issue. The reviewers were extremely diligent and timely during the review process. James Bieman invited and encouraged us to produce this special issue covering important aspects of qualitative software engineering. We also would like to thank Sharon Palleshi from Springer who was in charge of the production process.

We hope that you share our excitement with the latest results on program comprehension research!

James C. Cordy, Queens University, Canada
Harald Gall, University of Zurich, Switzerland
Jonathan I. Maletic, Kent State University, USA