2006

Comprehension and Maintenance of Large Scale Multi-Language Software Applications: Open Issues and Challenges

Kostas Kontogiannis

Panos K. Linos
Butler University, linos@butler.edu

Kenny Wong

Comprehension and Maintenance of Large Scale Multi-Language Software Applications

Kostas Kontogiannis
University of Waterloo, Canada
kostas@swen.uwaterloo.ca

Panos Linos
Butler University, USA
linos@butler.edu

Kenny Wong
University of Alberta
kenw@cs.ualberta.ca

Abstract

During the last decade, the number of software applications that have been deployed as a collection of components implemented in different programming languages and paradigms has increased considerably. When such applications are maintained, traditional program comprehension and reengineering techniques may not be adequate. In this context, this working session aims to stimulate discussion around key issues relating to the comprehension, reengineering, and maintenance of multi-language software applications. Such issues include, but are not limited to, the formalization, management, exploration, and presentation of multi-language program dependencies, as well as the development of practical toolsets for automating and easing the comprehension and maintenance of multi-language software.

1. Introduction

Today, software engineers often use multiple languages to implement a single software application. In 1998, Capers Jones documented in his book that at least one third of the software applications at that time were written using two different programming languages [1]. In addition, he estimated that ten percent of all applications incorporate three or more languages. Today, these percentages are much higher.

There are several advantages associated with the development of multi-language (ML) software applications. For instance, database queries can be handled easily and efficiently when various query languages such as SQL are embedded in general purpose languages such as C++ and/or Java. In addition, reuse of existing software is often an important reason behind building ML software. For example, predefined and pretested libraries implemented in procedural languages may be included and used by newer software applications developed using object-oriented languages. As another notable example, Microsoft's Visual Studio .NET environment is widely used to develop ML (i.e., combination of Visual Basic, C#, C++, etc.) Windows and/or Web applications.

ML software must evolve constantly to accommodate changes in the application domain. This evolution demands continuous modifications that may increase software complexity and eventually reduce quality. Therefore, the need for broad research efforts and open discussions related to the maintenance and evolution of ML software becomes a compelling issue.

2. Background

It has been documented that one important factor in understanding, reengineering, and maintaining programs written in just one programming language is the detection of program dependencies [6]. A program dependency is present when one entity within a program relates to other entities within another piece of code. For instance, a program written in one language may call functions or access other entities of code written in another language. To date there has been little standardization effort in this area and, as a consequence, the resulting ML programs become difficult to understand and maintain.

Moreover, it has been shown that the visualization of such program dependencies can facilitate program comprehension and reengineering activities [2]. Finally, recent prototype tools for measuring and visualizing the complexity of ML software in the VS.NET environment have been developed [3].

3. Open issues for discussion

The working session will focus on the following open research issues related to the comprehension, reengineering, and maintenance of ML software.

Formalization and modeling

ML systems pose a whole new range of challenges on extracting information and modeling dependencies between the components. Some of these challenges relate to the use of different language constructs with varying semantics, and the diverse designs such ML systems possess. Some of the interesting questions that arise in the formalization of ML system dependencies include how we deal with different language semantics, whether we need to define specialized metamodels for ML system dependencies, and whether we need to adapt existing mathematical theories (e.g. graph theory, set theory) to better formalize and capture ML system dependencies.

Extraction, discovery, and storage

It is not enough to have program understanding tools that consider each language independently as an isolated island. We need to also bridge these islands to form a more complete understanding. For example, programmers often need to follow control flows in software, and this activity should not be constrained by language boundaries. Thus, an important issue is how to manage ML dependencies, including integrated fact extraction for constituent individual languages, static and dynamic analysis to discover ML dependencies, pattern recognizers that exploit cross-language linkage conventions [4, 5], and suitable data models and efficient storage mechanisms to represent the gathered information.

Exploration, queries, and knowledge management

Programmers often need to follow control and data flows in software systems, without being constrained by language boundaries. Thus, we need effective graph navigation techniques to explore dependencies in ML software. Also important are queries to match patterns that span languages, to discover high-level graph abstractions, or reveal potential anomalies or inconsistencies such as malformed or missing stubs in linkage mechanisms. If changes are necessary, a key challenge is how to relate corrections to graph information back to the original code.

Presentation and usability

Another challenge is how to transfer the discovered information into the minds of software engineers to assist with their required evolution tasks. Many issues in software visualization arise, including appropriate presentations, aesthetic criteria for graph layout [7], and the relation of information presented across multiple views. To effectively highlight key issues in ML systems, we need abilities to filter and progressively disclose information.

Tool support

Processing large volumes of information that can be extracted from ML systems requires the use of sophisticated tools and tractable algorithms. Consequently, some of the challenges that need to be answered include: the type of tools that are required to understand, analyze and maintain ML software; the coordination and architecture model that is required for such tools to interoperate and share data; and how new tools can be seamlessly integrated in such an environment (possibly using a publish-subscribe paradigm or service-oriented architectural style.

Impact on software maintenance processes

The use of tools to analyze, maintain, and evolve large ML systems is bound to have an effect on maintenance and evolution process models. Some questions that arise in ML analysis and maintenance include the type of metrics or other quantifiable means to reason about the evolution process, to measure the impact of ML software on evolution effort, and to establish a common measurement framework. Finally, an interesting challenge is to investigate the issues pertaining to designing and conducting effective experimental studies that can evaluate different ML system analysis and maintenance techniques in practical settings.

4. Working session activities

We expect that this session will attract researchers with an interest in understanding, reengineering, and maintaining multi-language software applications.

The working session will start with a few pre-invited short anchor presentations. The speakers will review in advance the open issues outlined in Section 3 and present their views during their talks.

After each presentation, we will allow for some clarification questions and a brief discussion. The major objective is to structure the working session around an interactive brainstorming period where the attendees will lay a roadmap of possible research directions and challenges to address the problem of analyzing, maintaining, and evolving large ML applications.

Furthermore, all results produced by the working session will be consolidated in a final report, which includes a copy of all presentation slides, a summary of important points made during the discussions, a suggested research framework, and a list of session participants and possibly interested researchers. We will post this report at the conference Web site for post-conference networking and collaboration.

5. Background of organizers

Panos Linos is a Professor of Computer Science and Software Engineering at Butler University. Before joining Butler, Panos was a Professor and the Chair of the Computer Science Department at Tennessee Technological University. His research activities focus on the areas of software maintenance, reengineering, and comprehension of

software. While at Butler University, he founded and currently directs the Center for Applied Software Engineering Research (CeASER), where he conducts funded research with his students, colleagues, and local information technology companies. He has also founded the EPICS (Engineering Projects In Community Service) program at Butler University.

Kostas Kontogiannis is an Associate Professor at the Department of Electrical & Computer Engineering at the University of Waterloo, Canada. He is working in the areas of software reverse engineering, software reengineering, and software systems integration. He has been the recipient of three IBM University Partnership Awards and a Canada Foundation for Innovation (CFI) Award. He is also a visiting scientist at the IBM Center for Advanced Studies in the IBM Toronto Laboratory.

Kenny Wong is an Associate Professor in the Department of Computing Science at the University of Alberta. His main areas of research include software comprehension, software evolution, and software visualization. This research includes building and using integrated environments for reverse engineering, and devising strategies to understand and evolve diverse software systems (supported in part by an Eclipse Innovation Grant). He is General Chair of the 2007 International Conference on Program Comprehension in Banff, and Program Chair of the 2008 International Conference on Software Maintenance in Beijing.

References

[1] C. Jones. *Estimating Software Costs*. McGraw-Hill, New York, 1998.

[2] P. Linos, Z. Chen, S. Berrier, and B. O'Rourke. A tool for understanding multi-language program dependencies. In *Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC 2003)*, pages 64–72, May 2003.

[3] G. McCullough, E. Maier, and P. Linos. A metrics tool for multi-language .NET software applications. In *Proceedings of the 18th Undergraduate Research Conference*, pages 42–43, April 2006.

[4] D. Moise and K. Wong. Extracting and representing cross-language dependencies in diverse software systems. In *Proceedings of the 12th Working Conference on Reverse Engineering (WCRE 2005)*, pages 209–218, November 2005.

[5] D. Moise, K. Wong, H. Hoover, and D. Hou. Reverse engineering scripting language extensions. In *Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC 2006)*, pages 295–304, June 2006.

[6] N. Wilde and R. Huitt. Maintenance support for object-oriented programs. *IEEE Transactions in Software Engineering*, 18(12):1038–1044, 1992.

[7] K. Wong and D. Sun. On evaluating the layout of UML diagrams for program comprehension. *Software Quality Journal*, 14(3), 2006. To appear.