Abstract

The goal of the Recast project is to support the evolution of object-oriented applications by focusing on three main directions: reverse engineering and reengineering, versions analysis, and migration towards components. The results of the Recast project for the current period can be sorted into software evolution analysis, class understanding and visualization, feature analysis, code duplication identification and the use of formal concept analysis to cluster source code elements.

As we are hosted of the Software Composition Group of the University of Bern, we worked on topics related to the Recast project but also on topics related to the project “A Unified Approach to Composition and Extensibility”. This report only lists the results obtained in the context of the Recast project, other results are presented in the report of the project mentioned previously.

1 Recast Results

The results of the recast project for the current period can be sorted into evolution analysis, class understanding, feature analysis and the use of formal concept analysis.

Evolution Analysis. Understanding the evolution of an object-oriented system based on various versions of source code requires analyzing a vast amount of data. We work on the analysis of such an evolution by detecting and visualizing phases in the evolution, i.e., abstractions of time spans where the encapsulated versions all comply with an expression. Our approach is applicable on any level, i.e., not only on system level, but for example also on class level. Our approach furthermore contains a set of measurements on phases that characterize them. Phases help understand an evolution of large systems as they enable studying an evolution on a higher level and offer concurrent phases which enables studying an evolution from different perspectives at the same time [1].

We defined an approach for identifying candidate classes for reverse engineering and reengineering efforts. Our solution is based on summarizing the changes in the evolution

[1](SNF Project No. 200020-105091/1, Oct. 2004 - Sept. 2006)
of object-oriented software systems by defining history measurements. Our approach, named Yesterday’s Weather, is an analysis based on the retrospective empirical observation that classes which changed the most in the recent past also suffer important changes in the near future [18]. We also worked on the analysis of evolution of class hierarchies [11] and correlating changes [19].

**Supporting Class Analysis and Understanding.** Classes are the primary unit of abstractions in object-oriented programs and as such represent key entities to be understood when reengineering legacy systems. The main problem is to quickly grasp the purpose of a class and its inner structure. To help the reverse engineers in their first contact with a foreign system, we defined a new visualization technique: a class blueprint. A class blueprint is a semantically enriched static call flow representation of class internals. Class blueprints support the identification and classification of micro-patterns inside the classes. These micro-patterns conveys part of the behavior of classes and conveys lot of important coding information [6]. The identification of the micro-patterns is based on the human analysis and interpretation of the class blueprint visualization. To discover known and unknown micro-patterns automatically we developed theory of graph pattern recognition, mainly graph edit distance and maximal common subgraph (MCS) algorithms. Using MCS and hierarchical clustering we automatically detect known and unknown patterns [2].

**Features Analysis.** Software developers are constantly required to modify and adapt features of an application in response to changing requirements. The problem is that just by reading the source code, it is difficult to determine how classes and methods contribute to the runtime behavior of features. Moreover, dependencies between system features are not obvious, consequently software maintenance operations often result in unintended side effects. To tackle these problems, we propose a compact feature-driven approach (i.e., summarized trace information) based on dynamic analysis to characterize features and computational units of an application. We extract execution traces to achieve an explicit mapping between features and classes using two complementary perspectives [21].

**Reengineering Environment.** We continued the development of MOOSE, our reengineering environment [14] [15]. In particular now we are able to extend it easily to support new analyses such as dynamic information [12][21] and evolution analysis [10][11][18]. We unified its meta-model to support code generation and better tool integration. In particular, MOOSE now integrates the VAN meta-model that enables version analysis [18][19][20]. Parsing is a complex task and legacy systems are written in non mainstream languages, therefore we investigated an approach to discover the parser of a given language incrementally and driven by examples [3].

**Code Analysis.** To support the identification of unanticipated groups of entities such as collaborating methods, attributes within a class or classes within an inheritance hierarchy, we applied Formal concept analyses [4]. We show how FCA can be used to identified unanticipated dependencies schema among methods spread over class hierarchies [8]. We show how to use FCA to identify design patterns [7]. As FCA is often used on limited samples, we assembled the lessons that we learned during the process of applying FCA to large amount of data and what where the heuristics we developed [9].
Code Duplication. We finished our work on code duplication identification independently of the languages [5] [17]. We investigated how to support the understanding of code duplication tool output. To help reengineers categorizing the output of code duplication analysis tools we proposed some dedicated visualizations [13].

2 Publications

2.1 Masters Theses


2.2 Ph.D. Theses


2.3 Publications

Journal Papers


Conference Papers


Book Chapters


Technical Reports


Workshop Papers


2.4 Contributions of Collaborators

Mr. Girba refined the HisMo model to analyze evolution trends in large object-oriented systems. We are waiting for result of submissions to international journals. Mr. Girba improved significantly the Moose reengineering environment in terms of usability, extensibility and supported analysis.

Mrs Ponisio is working on assessing packages and supporting modularisation of object-oriented applications. We got one publication accepted on metrics for packages.

2.5 Network and Project Participation

The ESF release network ended and we are currently participating in a new ERCIM working group on Software Evolution.

2.6 Organized Events

- Organization of one workshop at the European Conference on Object-Oriented Programming ECOOP’2005 Object-Oriented Reengineering.

- Organization of the Annual European Smalltalk User Group Conference (100 participants). Chair of the Academic Track, editor of the special issue of the journal Computer Languages, Systems and Structures from Elsevier.