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 the following categories: software evolution analysis, system understanding, feature analysis and code analysis.

As we are hosted by 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”\(^1\). This report only lists the results obtained in the context of the Recast project over the last period \(i.e.,\ 2005-2006\). For results of the previous periods please read the corresponding report. Scientific results not related to the Recast project are presented in the report of the project mentioned previously.

Note that some papers are not yet on print or presented and as such are not in the direct time span of this report, still we include them in this report under a specific category (See Section 2.7) since logically belong to the Recast research effort.

1 Recast Results

The results of the Recast project for the current period can be sorted into software evolution analysis, system understanding, feature analysis and code analysis.

1.1 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 worked on the analysis of such an evolution by designing an extensible and dedicated meta-model called Hismo. Hismo supports various analysis at various level of abstractions. It served as the basis for several applications [4][7]. We included a human dimension in the analysis of the evolution by taking into account the behavior of the developers [2][26]. We also analyzed how developers copy code [8].

We also reported our experience as maintainers of a large open-source project so that the practical experience that we gain from the field could serve to stimulate research on software evolution [24].

1.2 System Understanding

We continue our research on the class understanding [25] but at the same time during this period we broadened our research on system wide analysis. We worked on two axes: using symbolic information and supporting package analysis.

To support the understand at the system level we developed a new visualization called Distribution Map which presents the distribution of properties over set of elements [33]. We also developed a new environment for building new visualization dynamically by scripting objects [36].

Symbolic information and semantic clustering. Understanding a software system by just analyzing the structure of the system reveals only half of the picture, since the structure tells us only how the code is working but not what the code is about. What the code is about can be found in the semantics of the source code: names of identifiers, comments etc. We analyzed how these terms are spread over the source artifacts using Latent Semantic Indexing, an information retrieval technique. We identified implementation topics by using the assumption that parts of the system that use similar terms are related [1][12][32].

Package Analysis. Understanding sets of classes, or packages, is an important activity in the development and reengineering of large object-oriented systems. Packages represent the coarse-grained structure of an application. They are the artefacts to deploy and structure software, and therefore more than a simple generalization of classes. The relationships between packages and their contained classes are key in the decomposition of an application and its (re)-modularisation. However, it is difficult to quickly grasp the structure of a package and to understand how a package interacts with the rest of the system. We tackle this problem using butterfly visualizations, i.e., dedicated radar charts built from simple package metrics based on a language-independent meta-model [5][9]. We also proposed to take into account the context and the use interface of the packages [16][19]. We experimented with the exploration of semantic clusters [30] and the identification of visual patterns to guide package exploration [14].

1.3 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. We continued our work on feature analysis and strengthen it by proposing a consistent set of views. We proposed a deeper approach and set of views to understand the way classes and features interact with each others [31]. In particular we analyzed the evolution of features [11].

We explored the use of 3D visualization to support the management of large traces [3][27] and also how the system evolves [34]. We applied symbolic analysis to feature traces [28]. We also applied signal processing techniques to identify the similarity between traces [35].

Related but not totally linked to features, program traces as one source of precious information about the internal working of a system. We showed that we can use this information
to support tests definitions. We represented trace information as logic facts and show how we could express tests using the run-time of the application [10].

1.4 Code Analysis

At the code analysis levels several paths have been followed: We used FCA to support the transformation of code to introduce traits [13], we presented some simplified implementation of object-oriented metrics [15]. We continued our work on code duplication analysis [6] taking into account the developers [8].

1.5 Others

We presented reengineering patterns to broader audiences [20] [22]. We presented our experience building the Moose environment and stressed the importance of meta-models in reverse and reengineering [21]. We participated to the definition of exchange formats in the context of code repository mining effort [29]. We participated in the definition of simple example for teaching refactorings [23].

2 Publications

2.1 Masters Theses


2.2 Ph.D. Theses


2.3 Publications

Journal Papers


Conference Papers


Tool Demo


Technical Reports


Invited Papers


Magazine


Workshop papers


2.4 Contributions of Collaborators

Mr. Girba finished his PhD and integrated deeply his history model into the Moose re-engineering environment. He significantly rewrote large parts of the environment and a new release of the Moose environment is included in the ESUG’2006 DVD. In addition Mr. Girba developed Mondrian a new visualization environment that is used by the Moose environment. Mr. Girba participates in the NOREX project, which is concerned with building a distributed environment for reengineering.

Mrs Ponisio finishes her PhD.

2.5 Network and Project Participation

We are currently participating in a new ERCIM working group on Software Evolution. S. Ducasse got a position as full professor at the University of Annecy and got a ANR (Agence National de la Recherche, the new French research agency) a research project related to the current one. The project is named Cook: Rearchitecturing Object-Oriented Applications.

2.6 Organized Events

- Organization of one workshop at the European Conference on Object-Oriented Programming ECOOP’2006 entitled Object - Oriented Reengineering.

- Organization of the Annual European Smalltalk User Group Conference (100 participants) at Prague.

2.7 Recast Pending Publications

Journals


International Conferences

