Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 77 (2005) 225–240
The Journal of Systems and Software
www.elsevier.com/locate/jss

A language-independent software renovation framework

M. Di Penta [a,*], M. Neteler [b], G. Antoniol [a], E. Merlo [c]

[a] Department of Engineering, RCOST—Research Centre on Software Technology, University of Sannio, Via Traiano, 1-82100 Benevento, Italy
[b] ITC-irst Istituto Trentino Cultura, Via Sommarive, 18-38050 Povo (Trento), Italy
[c] Département de Génie Informatique, Ecole Polytechnique de Montreal, P.O. Box 6079, Succ. Centre-Ville, Montreal, Quebec, Canada H3C 3A7

Received 1 April 2003; received in revised form 16 July 2003; accepted 2 March 2004
Available online 8 December 2004

Abstract

One of the undesired effects of software evolution is the proliferation of unused components, which are not used by any application. As a consequence, the size of binaries and libraries tends to grow and system maintainability tends to decrease. At the same time, a major trend of today's software market is the porting of applications on hand-held devices or, in general, on devices which have a limited amount of available resources. Refactoring and, in particular, the miniaturization of libraries and applications are therefore necessary.

We propose a Software Renovation Framework (SRF) and a toolkit covering several aspects of software renovation, such as removing unused objects and code clones, and refactoring existing libraries into smaller more cohesive ones. Refactoring has been implemented in the SRF using a hybrid approach based on hierarchical clustering, on genetic algorithms and hill climbing, also taking into account the developers' feedback. The SRF aims to monitor software system quality in terms of the identified affecting factors, and to perform renovation activities when necessary. Most of the framework activities are language-independent, do not require any kind of source code parsing, and rely on object module analysis.

The SRF has been applied to *GRASS*, which is a large open source Geographical Information System of about one million LOCs in size. It has significantly improved the software organization, has reduced by about 50% the average number of objects linked by each application, and has consequently also reduced the applications' memory requirements.

Keywords: Refactoring; Software renovation; Clustering; Genetic algorithms; Hill climbing

1. Introduction

Software systems evolution often presents several factors that contribute to deteriorate the quality of the system itself (Lehman and Belady, 1985). First, unused components, which have been introduced for testing purposes or which belong to obsolete functionalities, may proliferate. Second, maintenance and evolution activities are likely to introduce clones, while, for example, adding support and drivers for an architecture similar to an already supported one (Antoniol et al., 2002). Third, library sizes tend to increase, because new functionalities are added and refactoring is rarely performed; for the same reasons, also the number of inter-library dependencies, some of which are circular, tends to increase. Finally, sometimes, new functionalities logically related to already existing ones are added in a non-systematic way and they result in sets of modules which are neither organized nor linked into libraries. As a consequence, systems become difficult to maintain. Moreover, unused objects, big libraries, and circular dependencies significantly increase application sizes and memory requirements. This is clearly in contrast

* Corresponding author.
E-mail addresses: dipenta@unisannio.it (M. Di Penta), neteler@itc.it (M. Neteler), antoniol@ieee.org (G. Antoniol), merlo@info.polymtl.ca (E. Merlo).

0164-1212/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2004.03.033

with today's industry hype towards porting existing software applications onto hand-held devices, such as Personal Digital Assistants (PDA), onto wireless devices (e.g., multimedia cell phones), or, in general, onto devices with limited resources.

This paper proposes the SRF to monitor and control some of the quality factors which have been described above. When the number of unused objects and clones increase, or when library sizes become unmanageable, some actions may be taken among the several possible ones. First and foremost, unused code may be removed and clones may be monitored or factored out. Furthermore, some form of restructuring, at library and at object file level, may be required. Together with monitoring and improving maintainability, the SRF eases the miniaturization challenge of porting applications onto limited resources devices.

Most of the SRF activities deal with analyzing dependencies among software artifacts. For any given software system, dependencies among executables and object files may be represented via a dependency graph, which is a graph where nodes represent resources and edges represent dependencies. Each library, in turn, may be thought of as a subgraph in the overall object file dependency graph. Therefore, software miniaturization can be modeled as a graph partitioning problem. Unfortunately, it is well known that graph partitioning is an NP-hard problem (Garey and Johnson, 1979) and thus heuristics have been adopted to find a "good-enough" solution. For example, one may be interested to first examine graph partitions by minimizing cross edges between subgraphs which correspond to libraries. More formally, a cost function describing the restructuring problem has to be defined and heuristics to drive the solution search process must be identified and applied.

We propose a novel approach in which hierarchical clustering and *Silhouette* statistics (Kaufman and Rousseeuw, 1990) are initially used to determine the optimal number of clusters and the starting population of a Software Renovation Genetic Algorithm (SRGA). This initial step is followed by a SRGA search aimed at minimizing a multi-objective function which takes into account, at the same time, both the number of inter-library dependencies and the average number of objects linked by each application. Finally, by letting the SRGA fitness function also consider the experts' suggestions, the SRF becomes a semi-automatic approach composed of multiple refactoring iterations, which are interleaved by developers' feedback. To speed up the search process, heuristics based on a Genetic Algorithm (GA) and a modified GA (Talbi and Bessière, 1991) approach were proposed. Performance improvement was also achieved by means of a hybrid approach, which combines GA strategies with hill climbing techniques.

The SRF has the advantage of being language independent. All activities, except clone detection, rely on information extracted from object files; furthermore, the clone detection algorithm adopted in the SRF is not tied to any specific programming language, provided that a set of metrics can be extracted from the source code.

The SRF has been applied to a large Open Source software system: a Geographical Information System (GIS) named *GRASS* [1] (Geographic Resources Analysis Support System). *GRASS* is a raster/vector GIS combined with integrated image processing and data visualization subsystems (Neteler and Mitasova, 2002) composed of 517 applications and 43 libraries, for a total of over one million LOCs.

The number of team members is small and it is about 7–15 active developers. Decisions are usually taken by the members most capable to solve specific problems. Developers are also *GRASS* users and they often focus on their needs within the general project.

This paper is organized as follows. First, a short review on related work (Section 2) and on main notions of clustering and GAs (Section 3), will be presented. Then, the SRF is presented in Section 4. The case study software system (i.e., *GRASS*) is described in Section 5, while results are presented and discussed in Section 6, and are followed by conclusions and work-in-progress in Section 7.

2. Related work

Many research contributions have been published about software system modules clustering and restructuring, identifying objects, and recovering or building libraries. Most of these work applied clustering or Concept Analysis (CA).

An overview of CA applications to software reengineering problems was published by G. Snelting in his seminal work (Snelting, 2000). Snelting applied CA to several remodularization problems such as exploring configuration spaces (see also Krone and Snelting, 1994), transforming class hierarchies, and remodularizing COBOL systems. Kuipers and Moonen (2000) combined CA and type inference in a semi-automatic approach to find objects in COBOL legacy code. Antoniol et al. (2001a) applied CA to the problem of identifying libraries and of defining new directories and files organizations in software systems with degraded architectures. As according to Krone and Snelting (1994), Kuipers and Moonen (2000), and Antoniol et al. (2001a), we believe that with the present level of technology a programmer-centric approach is required, since programmers are in charge of choosing the proper remodularization strategy based on their knowledge

[1] http://grass.itc.it