Weaving Between Non-Functional Requirements and Object-Oriented Re-engineering

Ladan Tahvildari, Kostas Kontogiannis  
Dept. of Elect. & Comp. Eng.  
University of Waterloo  
Waterloo, Ontario  
Canada, N2L 3G1

John Mylopoulos  
Dept. of Computer Science  
University of Toronto  
Toronto, Ontario  
Canada, M5S 3G4

The widespread use of the object-oriented design paradigm and the programming languages is a contributing factor for the increase on the amount of object-oriented software introduced in many organizations. Gradually, a new generation of object-oriented legacy systems is emerging. The technologies for addressing the maintenance and the evolution of such systems will be of vital importance in the years to come. Also, software re-engineering tasks have to conform with hard and soft quality constraints (or non-functional requirements) such as “the re-engineered system must run as fast as the original system”, or “the new system should be more easily maintainable than the original system”. These desired qualities (or, more precisely, desired deltas on these qualities) play a fundamental role in defining the object-oriented re-engineering process and the techniques that support it. We have developed a re-engineering model that is quality-driven in the sense that it uses non-functional requirements to define and guide such a re-engineering process.

The major theme to the proposed approach is to exploit the synergy between requirements analysis, software architecture, and reverse engineering [1]. Understanding the architecture of an existing system aids in predicting the impact evolutionary changes have on specific quality characteristics of the system [3]. The re-engineering approach consists of: i) requirements analysis to identify specific re-engineering goals, ii) model analysis to understand the system’s design and architecture, iii) source code analysis to understand a system’s implementation, iv) remediation specification to examine the particular problem and to select the optimal transformation for the system, v) transformation to apply transformation rules in order to re-engineer a system in a way that complies with specific quality criteria, and vi) evaluation process to assess whether the transformation has addressed the specific requirements set [4].

In a nutshell, once software artifacts have been understood, classified and stored during the reverse engineering phase, their behavior can be readily available to the system during the forward engineering phase. The forward engineering phase aims to produce a new version of legacy system that operates on the target architecture and meets specific non-functional requirements (i.e., maintainability or performance enhancements). To represent information about quality requirements, their inter-dependencies, and the software transformations that affect them, we adopt the NFR framework [2]. According to the framework, software qualities are represented as soft-goals, i.e., goals with no clearcut criterion as to whether they have been fulfilled or not. Soft-goals can be related to other soft-goals in terms of relations such as $A \lor D, O \lor R, + \lor -$. The proposed process is iterative and incremental in nature. It means that at each iteration cycle, an evaluation procedure is applied to ensure that each transformation step conforms with the requirements set for the new system.