Inf Syst Front (2016) 18:825–853
DOI 10.1007/s10796-016-9685-3

# Ranking software components for reuse based on non-functional properties

**Marcus Kessel**[1] · **Colin Atkinson**[1]

Published online: 23 July 2016


**Abstract** One of the biggest obstacles to software reuse is the cost involved in evaluating the suitability of possible reusable components. In recent years, code search engines have made significant progress in establishing the semantic suitability of components for new usage scenarios, but the problem of ranking components according to their non-functional suitability has largely been neglected. The main difficulty is that a component's non-functional suitability for a specific reuse scenario is usually influenced by multiple, "soft" criteria, but the relative weighting of metrics for these criteria is rarely known quantitatively. What is required, therefore, is an effective and reliable strategy for ranking software components based on their non-functional properties without requiring users to provide quantitative weighting information. In this paper we present a novel approach for achieving this based on the non-dominated sorting of components driven by a specification of the relative importance of non-functional properties as a partial ordering. After describing the ranking algorithm and its implementation in a component search engine, we provide an explorative study of its properties on a sample set of components harvested from Maven Central.

**Keywords** Software quality · Software metrics · Software components · Non-functional requirements · Software testing · Software reuse · Software component ranking · Sorting · Multi-criteria decision making

✉ Marcus Kessel
kessel@informatik.uni-mannheim.de

Colin Atkinson
atkinson@informatik.uni-mannheim.de

[1] University of Mannheim, Mannheim, Germany

## 1 Introduction

The reuse of existing software components (Frakes and Kang 2005) from company-wide or open source repositories is often an appealing alternative to the costly development of new components from scratch. However, a developer needs to trade off the instant availability of such potentially reusable candidates against the effort needed to find them and adapt them to achieve the level of quality required in his/her context (Robillard et al. 2014) and usage scenario (Slyngstad et al. 2006; Bauer et al. 2014). The reuse of a software component can be a laborious task depending on the amount of code that needs to be changed and the strictness of the non-functional requirements (Holmes and Walker 2012). Most research in the field of reuse has traditionally focused on offering tool-aided guidance and advice for adapting and integrating code, once selected, into a new application (e.g., Holmes and Walker 2012; Makady and Walker 2013). However, arguably the most crucial part of the whole process is finding the most suitable component in the first place.

Code recommendation tools (Robillard et al. 2010, 2014) and specifically code search engines (Sim and Gallardo-Valencia 2013) provide the basic platform for supporting the "finding" part of the reuse process. However, the current generation of code search engines provide users with little or no support for evaluating potential reuse candidates, especially in the context of pragmatic (Holmes and Walker 2012) or ad hoc reuse (Krueger 1992) which envisages the invasive adaptation of reused components. In particular, users receive no indication of which component will "best" meet his/her non-functional requirements (e.g. according to quality metrics). This is a big weakness, since the initial search and selection phase has a critical impact on the likely success of reuse (Slyngstad et al. 2006; Bauer et al. 2014), and