Chapter 8
Software Reuse in Open Source:
A Case Study

Andrea Capiluppi
Brunel University, UK

Klaas-Jan Stol
Lero (The Irish Software Engineering Research Centre), University of Limerick, Ireland

Cornelia Boldyreff
University of East London, UK

ABSTRACT

A promising way to support software reuse is based on Component-Based Software Development (CBSD). Open Source Software (OSS) products are increasingly available that can be freely used in product development. However, OSS communities still face several challenges before taking full advantage of the "reuse mechanism": many OSS projects duplicate effort, for instance when many projects implement a similar system in the same application domain and in the same topic. One successful counter-example is the FFmpeg multimedia project; several of its components are widely and consistently reused in other OSS projects. Documented is the evolutionary history of the various libraries of components within the FFmpeg project, which presently are reused in more than 140 OSS projects. Most use them as black-box components; although a number of OSS projects keep a localized copy in their repositories, eventually modifying them as needed (white-box reuse). In both cases, the authors argue that FFmpeg is a successful project that provides an excellent exemplar of a reusable library of OSS components.

INTRODUCTION

Reuse of software components is one of the most promising practices of software engineering (Basili & Rombach, 1991). Enhanced productivity (as less code needs to be written), increased quality (since assets proven in one project can be carried through to the next) and improved business performance (lower costs, shorter time-to-market) are often pinpointed as the main benefits of developing software from a stock of reusable components (Sametinger, 1997; Sommerville, 2004).

Although much research has focused on the reuse of Off-The-Shelf (OTS) components, both Commercial OTS (COTS) and Open Source Software (OSS), in corporate software production

DOI: 10.4018/978-1-4666-2937-0.ch008

Copyright © 2013, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

(Li *et al.*, 2009; Torchiano & Morisio, 2004), the reusability *of* OSS projects *in* other OSS projects has only recently started to draw the attention of researchers and developers in OSS communities (Lang *et al.*, 2005; Mockus, 2007; Capiluppi & Boldyreff, 2008). A vast amount of code is created daily, modified and stored in OSS repositories, and the inherent philosophy around OSS is indeed promoting reuse. Yet, software reuse in OSS projects is hindered by various factors, psychological and technical. For instance, the project to be reused could be written in a programming language that the hosting project dislikes or is incompatible with; the hosting project might not agree with the design decisions made by the project to be reused; finally, individuals in the hosting project may dislike individuals involved in the project to be reused (Senyard & Michlmayr, 2004). A search for the "*email client*" topic in the SourceForge repository (http://www.sourcforge.net) produces 128 different projects (SourceForge, 2011): this may suggest that similar features in the same domain are implemented by different projects[1], and that code and features duplication play a significant role in the production of OSS code.

The interest of practitioners and researchers in the topic of software reuse has focused on two predominant questions: (1) from the perspective of *OSS integrators* (Hauge *et al.*, 2007), how to select an OSS component to be reused in another (potentially commercial) software system, and (2) from the perspective of end-users, how to provide a level of objective "trust" in available OSS components. This interest is based on a sound reasoning; given the increasing amount of source code and documentation created and modified daily, it starts to be a (commercially) viable solution to browse for components in existing code and select existing, working resources to reuse as building blocks of new software systems, rather than building them from scratch.

Among the reported cases of successful reuse within OSS systems, components with clearly defined requirements, and hardly affecting the overall design (i.e., the "S" and "P" types of systems following the original S-P-E classification by Lehman (1980)) have often proven to be the typically reused resources by OSS projects. Reported examples include the "internationalization" (often referred to as I18N) component (which produces different output text depending on the language of the system), or the "install" module for Perl subsystems (involved in compiling the code, test and install it in the appropriate locations) (Mockus, 2007). To our best knowledge, there is no academic literature about the successful reuse of OSS, and an understanding of internal characteristics of what makes a component reusable in the OSS context is lacking.

The main focus of this paper is to report on the FFmpeg project (http://ffmpeg.org/), and its build-level components, and to show how some of these components are currently reused in other projects. This project is a cornerstone in the multimedia domain; several dozens of OSS projects reuse parts of FFmpeg, one of the most widely reused being the libavcodec component. In the domain of OSS multimedia applications, libavcodec is the most widely adopted and reused audio/video codec (**co**ding and **dec**oding) resource. Its reuse by other OSS projects is so widespread since it represents a crosscutting resource for a wide range of systems, from single-user video and audio players to converters and multimedia frameworks. As such, FFmpeg represents a unique case (Yin, 2003, p.40), which is why we selected the project for this study.

In particular, the study is an attempt to evaluate whether the reusability principle of "high cohesion and loose coupling" (Fenton, 1991; Macro & Buxton, 1987; Troy & Zweben, 1981) has an impact on the evolutionary history of the FFmpeg components.

This paper makes two contributions:

1. It studies how the *size* of FFmpeg components evolve: the empirical findings show that the libavcodec component (contained in

24 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage:
www.igi-global.com/chapter/software-reuse-open-source-case/74667

Related Content

Ecology and Dynamics of Open Source Communities
Michael Weissand Gabriella Moroiu (2007). *Emerging Free and Open Source Software Practices (pp. 46-67).*
www.irma-international.org/chapter/ecology-dynamics-open-source-communities/10082

MOOCs: Exploiting Networks for the Education of the Masses or Just a Trend?
Vanessa Camilleri, Leonard Busuttiland Matthew Montebello (2015). *Open Source Technology: Concepts, Methodologies, Tools, and Applications (pp. 1282-1300).*
www.irma-international.org/chapter/moocs/120969

The System for Population Kinetics: Open Source Software for Population Analysis
Paolo Vicini (2009). *International Journal of Open Source Software and Processes (pp. 29-43).*
www.irma-international.org/article/system-population-kinetics/38904

Requirements to Class Model via SBVR: RECM via SBVR TOOL
Murali Mohananand Imran Sarwar Bajwa (2019). *International Journal of Open Source Software and Processes (pp. 70-87).*
www.irma-international.org/article/requirements-to-class-model-via-sbvr/233514

Facilitating the Integration of Open Educational Courses
Yolanda Debose Columbus (2013). *Open-Source Technologies for Maximizing the Creation, Deployment, and Use of Digital Resources and Information (pp. 26-33).*
www.irma-international.org/chapter/facilitating-integration-open-educational-courses/70117