A study on third party component reuse in Java enterprise open source software

Conference Paper · June 2013

DOI: 10.1145/2465449.2465468

2 authors:

Widura Schwittek
University of Duisburg-Essen
13 PUBLICATIONS **57** CITATIONS
SEE PROFILE

Stefan Eicker
University of Duisburg-Essen
72 PUBLICATIONS **251** CITATIONS
SEE PROFILE

A Study on Third Party Component Reuse in Java Enterprise Open Source Software

Widura Schwittek
paluno – The Ruhr Institute
for Software Technology
University of Duisburg-Essen
45141 Essen
widura.schwittek@paluno.uni-due.de

Stefan Eicker
paluno – The Ruhr Institute
for Software Technology
University of Duisburg-Essen
45141 Essen
stefan.eicker@paluno.uni-due.de

ABSTRACT

Recent studies give empirical evidence that much of today's software is to a large extent built on preexisting software, such as commercial-off-the-shelf (COTS) and open source software components. In this exploratory study we want to contribute to this small but increasing body of knowledge by investigating third party component reuse in 36 Java web applications that are open source and are meant to be used in an enterprise context. Our goal is to get a better understanding on how third party components are reused in web applications and how to better support it.

The results are in line with existing research in this field. 70 third party components are being reused on average. 50 percent of the 40 most reused third party components are maintained by the Apache Foundation. Further research questions based on the study results were generated and are presented at the end of this paper.

Categories and Subject Descriptors

D.2.13 [**Reusable Software**]: Reusable libraries

Keywords

Off-the-shelf, third party, Java, component, web application, component identification, reuse, study

1. INTRODUCTION

Reusing third party components has become a key success factor in software development projects [1] leading to reduced costs, faster time-to-market and better software quality [2]. Recent studies ([3], [4]) give empirical evidence that much of today's software is indeed to a large extent build on preexisting software, such as commercial-off-the-shelf (COTS) and open source software components. This especially holds true for web applications, where architects and developers are faced with the steady proliferation of new technologies and standards [5].

In this exploratory study we want to contribute to this small but increasing body of knowledge of third party component reuse as part of the larger field of software reuse. We analyze 36 end-user products such as content management or enterprise resource planning systems to get insights on how these reuse third party components. We start our investigation with Java web applications that are open source and are meant to be used in an enterprise context. We focus on the huge open source Java web application market since the software is freely available and thus analyzable without any restrictions. As such, all identified third party components also follow an open source license which excludes COTS components from this study. We focus on black-box software components reuse compared to white-box reuse where source code is reused or software components are adapted and reassembled. We rely on a study by Heinemann et al. [3] who have shown that white-box usually makes up only a small portion of reuse compared to black-box reuse and is as such negligible. Speaking in terms of general reuse, our study also does not consider the reuse of third party components which reside outside the web application as part of an external platform. For instance, references to the Java Persistence API, which implementation is provided by an application server are not counted. Also, other technologies which are outside the scope of the Java programming language but are part of a typical web application are not considered such as the popular Javascript library JQuery.

A small tool has been developed in order to retrieve the reuse data by extracting, transforming and analyzing deployment artifacts. With this tool we automate most of the analysis which enables us to process as many projects as possible and lays the foundation for achieving a critical mass of reuse data. Nonetheless, this tool is not the main contribution of this paper.

This paper is structured as follows. In Section 2, the study implementation and execution is explained in detail, including the automated data generation mechanism using the tool. In Section 3 and Section 4 the study results are described and discussed. These two sections represent the main contribution of this paper. The threats to validity of the study results are explained in Section 5 while we point to and discuss related work in Section 6. Section 7 ends this paper with the conclusions and a set of new research questions which arise from this exploratory study.

2. STUDY IMPLEMENTATION AND EXECUTION

In this section, the study implementation and execution is described in more detail.

2.1 Criteria of inclusion

Due to time constraints we decided to include Java web applications that address primarily enterprises rather than private end users assuming that third party components are more carefully selected when used in production in an enterprise context.

Our final set includes web applications from the following domains: business intelligence (BI), business process management (BPM), customer relationship management (CRM), software development (SD) such as continuous integration and artifact repository software, content management (CM) under which we subsume all variants such as web CM and enterprise CM as well as document management, enterprise resource planning (ERP), human resources (HR), knowledge management (KM) learning management (LM) and social media (SM) such as blog and wiki software. Additionally, we subsume specialized software such as research administration, controlling and student information under the category "Other domains".

2.2 Retrieving reuse data

Retrieving reuse data from Java open source web applications is a repetitive task for which we developed a tool. Automating many steps allows us to process a large number of projects. This lays the foundation for achieving a critical mass of usage data in future studies. From the scientific point of view the level of automation enabled by the tool also allows others to reproduce our results more easily. The name of the tool "Component Decision ETL Pipeline" reflects our perspective considering the appearance of third party components as "Component Decision" (see literature on architectural decisions [6]) but also hints at the automated data generation referring to the data warehouse terminology "ETL pipeline". This term stands for processing data along the three consecutive activities of extraction, transformation and loading (ETL). These steps were also conducted for our study supported by our tool and are described in more detail in the following.

2.2.1 Extraction

Having selected a set of study objects – following the inclusion criteria in our case –, the initial step of the pipeline is to extract all relevant data from all the different sources.

In the case of open source Java web projects, a concrete URL of the project's website is used to download the web archive (files have the suffix .war in their filename). It includes all files to run this application on an application server. Web archives are basically compressed packages of files comparable to zip files, which follow a standardized directory structure. The third party components are located within the subdirectory /WEB-INF/lib.

The pipeline is able to receive a list of URLs pointing to web archives each representing a web application to be analyzed. It automatically extracts all third party components from the aforementioned location each consisting of one or more Java archives (files have the suffix .jar in their filename). Java archives can be considered as a compressed package of compiled class and resource files organized in a standardized directory structure, which can be integrated into existing applications as a component or serve as an executable for stand-alone applications.

Then, from each Java archive the following data fields are extracted:

- the Web archive it belongs to
- the Java archive's filename
- the SHA1 hash value of the Java archive
- if available, the version number from the /META-INF/manifest.mf file

Additionally, "The Central Repository"[1] is queried via its RESTful web services to get even more data on a specific Java archive. This repository is used by the Java build tool Maven[2], to automatically resolve third party component dependencies within a project and download it. While Maven becomes increasingly state-of-practice in the Java software development community, also its "Central Repository" gains popularity making it more attractive for component developers to publish their components there too.

The pipeline makes use of this repository to get more data on Java archives by sending in the SHA1 hash value generated from the file contents pointing to an exact dataset if available. On failure, the pipeline does a second request but now using the filename which in most cases contains the name and version. If the Java archive is found in the repository additional data fields are returned from which the following are relevant:

- GroupId
- ArtifactId
- Version
- Publishing date

To make sure that Java archives are not only part of the web archive but their methods and data structures are really used from the web application, all references are matched against the class names including the package name provided by the third party components. This matching also includes transitive dependencies where Java archives themselves require another third party component (as shown in Figure 1 for JAR C and JAR D). Since these Java archives do not contain the source code but the compiled class files in a Java byte code representation, special tools for processing Java byte code such as BCEL[3] or Javassist[4] become necessary. The pipeline uses Javassist to extract Java archive dependency information which is stored in the ClassPool section in every class file.

The output of this activity is a dataset with each entry containing the aforementioned data fields if available.

2.2.2 Transforming

In this step, the raw data from the extraction step is transformed into the target format which is used for the evaluation. The target format consists of three data fields, one for storing Java archives, one for storing third party components consisting of one or multiple Java archives and finally the last one for storing projects which reuse these components (see Figure 1).

[1] http://search.maven.org/#api

[2] http://maven.apache.org/

[3] http://commons.apache.org/bcel/

[4] http://www.jboss.org/javassist/

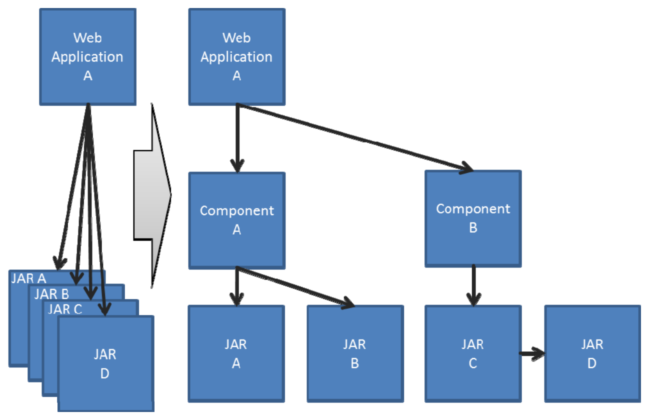

Figure 1. Transformation of the dataset.

Two problems arise during the transformation of the input dataset which need to be solved and which would otherwise require manual input:

1. As mentioned earlier, third party components consist of one or multiple Java archives. Since we are interested in usage relationships between web applications and third party components and not Java archives, we need to determine these third party components that span multiple Java archives (see Figure 1: starting on the left side solely with the web application to Java archive dependencies, Component A and Component B are unknown entities that need to be determined).

   Our solution to this problem is twofold. First, the pipeline tries to guess related Java archives on basis of the data field GroupId, which might have been received from the Central Repository. If a Java archive's dataset exists in the Central Repository and follows the good practice proposed by Maven[5], the pipeline can derive a third part component from the GroupId, since it groups all Java archives that belong to one project. If this data field is not used as expected or is not available at all, related Java archives need to be grouped to components manually. Anyway, in order to fully automate the process, the transformation from the existing data set to manually grouped Java archives is stored in a mapping table (see an excerpt in Table 1) so that it can be reused each time the pipeline is triggered. The larger the dataset the more groupings are known and less manual intervention becomes necessary.

Table 1. Mapping of artifacts to third party components (excerpt)

Artifact name	Third party component name
icu4j	com.ibm.icu
icu4j-3_8_1	com.ibm.icu
icu4j-charsets	com.ibm.icu
icu4j-localespi	com.ibm.icu
slf4j-api	slf4j
slf4j-jcl	slf4j
slf4j-log4j12	slf4j

[5] http://maven.apache.org/pom.html#Maven_Coordinates

2. The second problem is that not only third party components are part of the input dataset. In some cases web application modules are stored as Java archives and thus are part of the input dataset which need to be filtered out. It is the pipeline's task to identify and decide whether to include or exclude Java archives.

   This would actually require manual intervention to sort out all Java archives that are not considered as third party components. To avoid this, the pipeline uses the package information extracted from the web application's source code to identify its own Java archives. Chances are high that if packages match that this Java archive is not a third party component but belongs to the web application.

The output of this activity is a dataset containing the aforementioned data plus a component field with the guessed component relationship and cleaned up from all Java archives which originates from the web application's development.

2.2.3 Loading

All steps are performed on data within a MySQL database. So, in our case "loading" means splitting the output data set from the transformation step originating from a single database table into a relational model consisting of five tables: the three target tables web_application, third_party_component and java_archive to store all entities and another two tables to store the relationships between web applications and third party components and between third party components and Java archives (see Figure 1). These tables are the basis for further analysis.

3. RESULTS

In Table 2, a ranking is presented showing all analyzed 36 Java base Open source web applications used in an enterprise context. The ranking is sorted in a descending order beginning with the web application reusing most third party components. The number of reused components ranges from 16 to 161 components with an average number of about 70.The table also shows the version as well as the assigned category of the analyzed web application which was discussed in Section 2.1.

Table 2. Overview of the selected Java web applications

Web Application	Version	Domain	No. of Comp.
Alfresco	4.2c	CM	161
Liferay Portal CE	6.1.1GA2	CM	144
XWiki Enterprise	4.5RC1	SM	130
dotCMS	2.2	CM	127
Pentaho CE	4.8.0	BI	118
openKM	6.2.2	KM	97
jallInOne SOA	2.8.2	ERP	91
Kuali People Management	1.2.2	HR	89
Kuali Coes	5.0.1	Other	89
Kuali Student	2.0.0M5	Other	87
openCMS	8.5.1	CM	83
openOLAT	8.3.3	LM	79
logicalDOC	6.6.1	CM	75
Magnolia Author	4.5.7	CM	71
Magnolia Public	4.5.7	CM	71
Jenkins	1.501	SD	70
Ametys	3.4.0	CM	70

DSpace XMLUI	3.1	CM	69
Hipergate	6.0.0RC1	CRM	68
Kuali Mobility	2.0	Other	66
hippoCMS	7.7.0	CM	61
openWGA CE	6.0.7	CM	60
DSpace JSPUI	3.1	CM	60
Tntnconcept	0.21.16	Other	55
Nexus	2.3.1_01	SD	52
Bonita Open Solutions	5.9.1	BPM	49
JRoller	5.0.1	SM	44
hippoCMS site	7.7.0	CM	42
Daisy	2.4.2	CM	41
Walrus CMS	1.5	CM	41
Alfresco Share	4.2c	CM	38
Ametys site	3.4.0	CM	37
jallInOne	2.8.2	ERP	29
vosaoCMS	0.9.14	CM	28
Jamwiki	1.2.4	SM	27
Agorum	7.0.4	CM	16

After having added all 36 web applications, the mapping table contained 863 mappings grouping 3311 different artifacts into 651 third party components.

Table 3 presents the top 40 of the most reused third party components after the analysis of all 36 web applications.

Table 3. Overview of the top 40 reused third party components

Libray	Times reused	Library	Times reused
commons-collections	34	org.springframework	21
commons-codec	34	org.apache.poi	21
org.apache. httpcomponents	32	aopalliance	20
commons-lang	32	org.antlr	19
commons-beanutils	32	org.objectweb.asm	19
commons-io	31	org.codehaus.woodstox. wstx-asl	19
commons-fileupload	29	org.bouncycastle	19
commons-logging	28	javax.xml.xml-apis	18
dom4j	26	commons-compress	18
commons-digester	25	net.java.dev.rome	18
org.apache.log4j	25	xpp	17
slf4j	25	org.apache.geronimo. specs	17
org.apache.xerces	25	hibernate	16
org.jdom	25	net.sf.cglib	15
org.apache.lucene	24	com.thoughtworks. xstream	15
commons-pool	23	org.apache.xalan	15
net.sf.ehcache	23	jaxen	15
javax.mail	22	stax	15
org.apache.oro	22	net.sourceforge.nekohtml	14
commons-dbcp	21	org.apache.pdfbox	14

4. DISCUSSION

The average of reused components is 70. Even Agorum with the smallest reuse rate still uses 16 third party components. From an integrator's point of view, it seems to be a tedious activity to keep track of all third party components being reused in a project especially with respect to bugfix or security updates.

Two possible explanations for this seemingly large number of reused third party components are as follows:

First, functionalities of third party components overlap. To give an example: 19 third party components reused by Alfresco are concerned with low-level processing XML formatted data. One reason for that are transitive dependencies. Deployable Java web applications do not only contain directly referenced third party components but also their dependencies to other third party components. This results not always in the optimal reuse setup when for instance two different third party components share the same required functionality but are both separately referenced as transitive dependency.

Second, we observe that especially web applications running in an enterprise context often require being easily embeddable into an existing environment to be successful. For instance, the aforementioned content management system Alfresco – counting 161 third party components – interfaces many different systems and is able to process many different file formats such as Office and PDF documents as well as multimedia files. If we look at the types of third party components included in Alfresco, 40 out of 161 are responsible for processing data, 34 are used for XML processing and Webservices and 26 for accessing external services such as SlideShare, Facebook, Twitter and Google Services.

Looking at the top 40 of reused third party components, 50 percent of the top 40 are third party components are maintained by the Apache Foundation. 11 out of these Apache components are hosted under the umbrella of the Apache Commons project which contains libraries which enhance the basic functionality of the Java platform including data structures, transformation of Java beans and so on. Third party components provided by the Apache Foundation seem to be very popular among web application developers. Some yet to be proved explanations are the popularity of the Apache Foundation in general, the good quality due to a rigorous development process and a good coverage of functionalities which is needed in web development.

In the top 40, we furthermore find 9 third party components which are related to web services and low-level XML processing (dom4j, org.apache.xerces, org.jdom, org.codehaus.woodstox.wstx-asl, javax.xml.xml-apis, xpp, com.thoughtworks.xstream, org.apache.xalan, jaxen, stax). While XML and Webservices are foundational to the web, other typical building blocks of web applications are among the top 40 including net.sf.ehcache (caching framework), org.springframework (web framework), javax.mail (mail library), org.apache.lucene (search engine), net.java.dev.rome (rss library), org.bouncycastle (cryptography framework) and hibernate (object relational mapping framework).

Besides Spring MVC, Struts, JSF and GWT are among the web frameworks.

5. THREATS TO VALIDITY

In this section, the potential threats to the study's validity are discussed.

5.1.1 Internal validity

In our study, the internal validity is threatened whenever manual input is required. This is the case, when the mapping table is built up to derive components from the set of Java archives (see Section 2.2.2) and when components are identified and excluded which are developed for the web application only. We try to address this by making these steps explicit in separate database tables to be inspected by others but also to be reused for reproducing the same results with the same input data set.

5.1.2 External validity

For our study we decided to start with the subset of Java web applications which are open source and meant to be used in an enterprise context. Thus, the small sample size is enough to prove the concept of the pipeline but is not representative for the class of Java web applications, the class of open source applications or the class of applications in general including commercial software. In order to improve the external validity, further studies are necessary that include a larger sample, web applications from other domains, web applications which are developed on other platforms and commercially developed web applications (see Section 7).

Although this preselection was undertaken comprehensibly (see Section 2.1) it is somewhat biased since there is no general accepted list enumerating this kind of software to which one could refer to.

6. RELATED WORK

Heinemann et al. [3] conducted an empirical study about software reuse in 20 open source Java projects. The study gives empirical evidence that third party code is heavily reused in open source projects and that black-box reuse tops white-box reuse. In 9 out of 20 projects more than 50% of the code base is reused.

Compared to our approach, their study differs in also considering white-box reuse. Their black-box analysis also looks at the byte code instead of the source code and does not only count the included Java archives but looks at what parts of the Java archive is actually used. With this approach, it is possible to come up with a ratio of reused third party component's byte code compared to the application's byte code. In our approach, we do not need to go so much into detail since we are interested merely in the usage relationships. We do a check on the real reuse of Java archives on byte code level anyway. Another difference lies in the study objects. While their set of 20 Java projects is quite heterogeneous ranging from Eclipse based applications and web applications to server applications all hosted at SourceForge.com, our selection of 36 focuses on web applications to be used in an enterprise context.

In their paper, Raemaekers et al. [4] investigate the frequency of use of third party components from a large corpus of proprietary and open source systems. The population consists of 106 open source applications from a curated repository called the Qualitas Corpus [7]. Additionally, they analyzed 178 proprietary systems to which they have access to as part of the Software Improvement Group (SIG). Their contribution is a popularity ranking of third party components, a mechanism for rating libraries based on the frequency of use, a mechanisms for rating systems in terms of their dependencies on third party components and a proposal of an indicator for risks present in theses libraries. Their goal is to provide decision support during component selection with the focus on possible risks derived from the ratings. They do not provide decision support for other aspects of component selection or identifying components.

The main difference to our approach is that Raemaekers et al. use curated source code data, while we use the latest web archives from the web application's website. While curated data is good for reproducing scientific results, it is somehow biased by the curator who selected the applications to be considered. The other 178 proprietary systems curated by SIG are not publicly available anyway, making it hard to reproduce these results. Additionally, looking at the population of the Qualitas Corpus, it by itself contains many third party components, while we are interested in end-user software.

Other empirical studies on reuse in open source projects exist. All of these give evidence to support the hypothesis that third party source code and binaries are heavily reused in open source projects. Some of these investigate projects artifacts of repositories as we do [8, 9, 10]. Some others mainly use interviews and questionnaires as data generation techniques [11]. We are specifically interested in improving the analysis of repositories to generate data in order to process it further.

The FOSSology tool [12] analyzes source code with the goal to discover third party code and binaries to mitigate hidden license obligations. This is also offered as a service by companies such as Black Duck[6], OpenLogic[7] or Palamida[8]. Sonatype and their Component Lifecycle Management (CLM) product portfolio[9] as well as Antelink's products[10] both go a step further and provide management and vulnerability notification tools. While some ideas and mechanisms might be interesting for our data generation tool, we plan to further process this data to investigate whether it is usable for decision support or not.

7. CONCLUSIONS AND FUTURE WORK

Recent studies ([3], [4]) give empirical evidence that much of today's software is to a large extent built on preexisting software, such as commercial-off-the-shelf (COTS) and open source software components. With this paper we contributed to this body of knowledge looking at Java open source applications meant to be used in an enterprise context. For this study we focused on web applications since Java is known to provide many third party components especially in this field.

We analyzed 36 Java web applications which reuse 70 third party components on average ranging from a minimum of 16 to a maximum of 161 components. The results are comparable to the existing studies, underlining that software reuse happens in this specific way of third party component reuse, independent from the kind of reuse (calling, extending, inheriting, instantiating).

During the discussion of the study's results (see Section 4) interesting questions arose, which seem worthwhile to be asked as part of future research:

- Considering the average reuse rate of 70 third party components in Java Open source web application development: Do integrators keep track of reused third

[6] http://www.blackducksoftware.com/

[7] http://www.openlogic.com/

[8] http://www.palamida.com/

[9] http://www.sonatype.com/Products

[10] http://www.antelink.com/

party components, especially with respect to bugfix and security updates? What are the risks of not doing it?

- We gave some possible explanations for the large number of third party components being reused in the analyzed web applications, which need further approval: Are inefficient transitive dependencies and the "easily embeddable" requirement of enterprise software the reasons for the huge number of reused third party components? Can the former be improved?
- Some third party components are reused by almost all of the analyzed web applications (e.g. Apache Commons and those concerned with XML processing and Webservices). Some other are more seldom reused. From the integrator's perspective the following questions arise: How can this knowledge be used to support the selection process of third party components? How does the reuse of certain sets of third party components relate to specific application domains?

Having laid the foundation with this work, we plan to find usage patterns in the data set to generate component recommendations during component identification and selection [13]. This is the reason for creating an almost automated data generation mechanism which we called "Component Decision ETL Pipeline".

Furthermore, we are currently investigating how to create similar data generation mechanisms for other platforms such as PHP, Ruby or .NET and integrate it into the current infrastructure. Besides the analysis of development artifacts from open source projects we are looking into getting usage data from developers and project managers of closed source projects for which we are piggy backing data generation on top of component management tool.

8. REFERENCES

[1] Gartner. 2008. *The Evolving Open-source Software Model. Predicts from December 2008.*

[2] Li, J., Conradi, R., Bunse, C., Torchiano, M., Slyngstad, O. P. N., and Morisio, M. 2009. Development with Off-the-Shelf Components: 10 Facts. *IEEE Softw.* 26, 2, 80–87.

[3] Heinemann, L., Deissenboeck, F., Gleirscher, M., Hummel, B., and Irlbeck, M. On the Extent and Nature of Software Reuse in Open Source Java Projects. In *Top productivity through software reuse* (International Conference on Software Reuse) 6727. Springer, Berlin, 207–222.

[4] Raemaekers, S., van Deursen, A., and Visser, J. 2012. An Analysis of Dependence on Third-party Libraries in Open Source and Proprietary Systems. In *Proceedings of the Sixth International Workshop on Software Quality and Maintainability (SQM 2012)*.

[5] Rossi, G. 2008. *Web engineering. Modelling and implementing web applications*. Human computer interaction series 12. Springer, London.

[6] Bosch, J. 2004. Software Architecture: The Next Step. In *Software Architecture, First European Workshop, EWSA 2004, St Andrews, UK, May 21-22, 2004, Proceedings*. Lecture Notes in Computer Science 3047. Springer, 194–199.

[7] Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Hayden, M., and Noble, J. 2010. The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In *17th Asia Pacific Software Engineering Conference (APSEC), 2010. Nov. 30 - Dec. 3 2010, Sydney, Australia*. IEEE, Piscataway, NJ, 336–345.

[8] Mockus, A. 2007. Large-Scale Code Reuse in Open Source Software. In *2007 First International Workshop on Emerging Trends in FLOSS Research and Development. (FLOSS 2007) ; Minneapolis, Minnesota, 20 - 26 May 2007 ; [ICSE 2007 workshops]*. IEEE, Piscataway, NJ, 7–11.

[9] Haefliger, S., Krogh, G. von, and Spaeth, S. 2008. Code Reuse in Open Source Software. *Management Science* 54, 1, 180–193.

[10] Ruiz, I. J. M., Nagappan, M., Adams, B., and Hassan, A. E. 2012. Understanding reuse in the Android Market. In *2012 20th IEEE International Conference on Program Comprehension (ICPC). Proceedings ; June 11-13, 2012, Passau, Germany*, IEEE, 113–122.

[11] Sojer, M. and Henkel, J. 2010. Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments. *Journal of the Association for Information Systems* 11, 12.

[12] Gobeille, R. 2008. The FOSSology project. In *Proceedings of the International Conference on Software Engineering & co-located workshops*, A. E. Hassan, M. Lanza and M. W. Godfrey, Eds. ACM, New York, NY, USA, 47–50.

[13] Schwittek, W. and Eicker, S. 2012. Decision Support for Off-the-Shelf Software Selection in Web Development Projects. In *Current trends in web engineering*. Lecture Notes in Computer Science 7703. Springer, Berlin, 238–243.