Assessing Third-Party Library Usage in Practice

Veronika Bauer
Technische Universität München
bauerv@in.tum.de

Florian Deissenboeck, Lars Heinemann
CQSE GmbH
{deissenboeck, heinemann}@cqse.eu

Abstract

Modern software systems build on a significant number of external libraries to deliver feature-rich and high-quality software in a cost-efficient and timely manner. As a consequence, these systems contain a considerable amount of third-party code. External libraries thus have a significant impact on maintenance activities in the project. However, most approaches that assess the maintainability of software systems largely neglect this factor. Hence, risks may remain unidentified, threatening the ability to effectively evolve the system in the future. We propose a structured approach to assess the third-party library usage in software projects and identify potential problems. Industrial experience strongly influences our approach, which we designed in a lightweight way to enable easy adoption in practice.

1 Introduction

A plethora of external software libraries form a significant part of modern software systems [3]. Consequently, external libraries and their usage have a significant impact on the maintenance of the including software. Unfortunately, third-party libraries are often neglected in quality assessments of software, leading to unidentified risks for the future evolution of the software. Based on industry needs, we propose a structured approach for the systematic assessment of third-party library usage in software projects. The approach is supported by a comprehensive assessment model relating key characteristics of software library usage to development activities. The model defines how different aspects of library usage influence the activities and, thus, allows to assess if and to what extent the usage of third-party libraries impacts the development activities of a given project. Furthermore, we provide guidance for executing the assessment in practice, including tool support.

2 Assessment model

The proposed assessment model is inspired by activity-based quality models [2].The model contains *entities*, "the objects we observe in the real world", and *attributes*, "the properties that an entity possesses" [4]. Entities are structured in a hierarchical manner to foster completeness. The combination of one or more entities and an attribute is called a *fact*. Facts are expressed as [Entities | ATTRIBUTE]. To express the impact of a fact, the model relates the fact to a development *activity*. This relation can either be positive, *i. e.*, the fact eases the affected activity, or negative, *i. e.*, the fact impedes the activity.

Impacts are expressed as [Entity | ATTRIBUTE] $\xrightarrow{+/-}$ [Activity]. Each impact is backed by a *justification*, which provides the rationale for its inclusion in the model. We *quantify* facts with the three-value ordinal scale {*low, medium, high*}.

To *assess* the impact on the activities, we use the three-value scale {*bad, satisfactory, good*}. If the impact is positive, there is a straight-forward mapping from *low* $\rightarrow$ *bad, medium* $\rightarrow$ *satisfactory, high* $\rightarrow$ *good.* If the fact [Library | PREVALENCE], for example, is rated high, the effect on the activity *migrate* is good as the impact relation is positive [Library | PREVALENCE] $\xrightarrow{+}$ [Migrate] as a high prevalence of a library usually gives rise to alternative implementations of the required functionality. If the impact relation is negative, the mapping is turned around: *low* $\rightarrow$ *good, medium* $\rightarrow$ *satisfactory, high* $\rightarrow$ *bad.* The assessment of a single library thus results in a mapping between the activities and the {*bad, satisfactory, good*} scale. To aggregate the results, we count the occurrences of each value at the leaf activities. Hence, the assessment of a library finally results in mapping from {*bad, satisfactory, good*} $\rightarrow \mathbb{N}_0$. We do not aggregate the results in a single number.

Activities With one exception, the activities included in the model are typical for maintenance. We consider *Modify*, *Understand*, *Migrate*, *Protect*, and *Distribute.*

Metrics The model quantifies each fact with one or more metrics[1]. As an example, to quantify the extent of vulnerabilities of a library, we measure the number of known critical issues in the bug database of the library. Some of the facts cannot be measured directly, as they depend on many aspects. For instance, the maturity of a library cannot be captured with a single metric but must be judged according to several criteria by an expert. We do not employ an

[1]The list of metrics, their description and assignment to facts are detailed in [1].

Library	Modify	Understand	Migrate	Protect	Distribute	Overall
Library	🟩 G:2 🟨 S:2 🟥 B:1	🟩 G:2 🟨 S:1 🟥 B:2	🟩 G:1 🟨 S:0 🟥 B:3	🟩 G:0 🟨 S:2 🟥 B:0	🟩 G:0 🟨 S:0 🟥 B:1	🟩 G:5 🟨 S:5 🟥 B:7

Legend: G: # of *good* impacts, S: # of *satisfactory* impacts, B: # of *bad* impacts

Figure 1: Example of the assessment overview of a library. The library's characteristics sufficiently support the activity *modify*. However, it incurs more risks than benefits for the activities *migrate* and *distribute*.

automatic, *e. g.*, threshold-based, mapping from metric values to the {low, medium, high} scale but fully rely on the experts capabilities.

Impacts define how facts influence activities. A justification for each impact provides a rationale for the impact which increases confirmability of the model and the assessments based on the model. The complete list of impacts is available online[2].

Our assessment process provides guidance to operationalize the model for assessing library usage in a specific software project. When assessing a real-world project, the sheer number of libraries require a possibility to address the *most relevant* libraries first. Therefore, the first step of the process structures and ranks the libraries according to their entangledness with the system. This pre-selection directs the effort of the second step of our process: the expert assessment of the libraries. The last step collects the results in an assessment report.

During preselection, we determine the following values for all libraries: The number of *total method calls* to a library allows to rank all external libraries according to the strength of their direct relations the system. The number of *distinct method calls* to a library adds information about the implicit entangledness of libraries and system. The *scatteredness of method calls* to a library describes whether the usage of the library is concentrated to a specific part of the system, or scattered across it. The *percentage of affected classes* gives a complementary overview about the impact a migration could have on the system.

Our model guides the expert during the assessment process. The automated analyses have provided the information which can be extracted from the source code. The expert now needs to evaluate the remaining metrics. For this, he or she requires detailed knowledge about the project and its domain and needs to research detailed information about the libraries.

Subsequent to the assessment, a report can be generated from our model, containing the detailed information for each library in textual and tabular (see Figure 1) form.

[2]`http://www4.in.tum.de/~ccsm/library-usage-assessment/`

3 Tool support

The assessment model includes five metrics that can be automatically determined by static code analyses. The tool support for the assessment is implemented in Java on top of the open source software quality assessment toolkit ConQAT[3]. The current implementation is targeted at analyzing the library usage of Java systems.

4 Case Study

To show the applicability of our approach, we performed a case study on a real-world software system.

The analyzed system is a distributed billing application with a distinct data entry component running on a J2EE application server which is accessed from around 350 fat clients (based on Java Swing). The system's source code comprises about 3.5 MLOC. The system's files include 87 Java Archive Files (JARs).

The results indicate that our approach gives a comprehensive overview on the external library usage of the analyzed system. It outlines which maintenance activities are supported to which degree by the employed libraries. Furthermore, the semi-automated pre-selection allowed for a significant reduction of the time required by the expert assessment.

Remarks

This paper presents a condensed version of previous work [1], published at the International Conference on Software Maintenance, 2012.

References

[1] V. Bauer, L. Heinemann, and F. Deissenboeck. A Structured Approach to Assess Third-Party Library Usage. In *ICSM'12*, 2012.

[2] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, and J. Girard. An activity-based quality model for maintainability. In *ICSM'07*, 2007.

[3] L. Heinemann, F. Deissenboeck, M. Gleirscher, B. Hummel, and M. Irlbeck. On the Extent and Nature of Software Reuse in Open Source Java Projects. In *ICSR'11*, 2011.

[4] B. Kitchenham, S. Pfleeger, and N. Fenton. Towards a framework for software measurement validation. *Software Engineering, IEEE Transactions on*, 21(12):929–944, 1995.

[3]`http://www.conqat.org/`