The Journal of Systems and Software 84 (2011) 1233–1243

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Controversy Corner

An empirical study of software architectures' effect on product quality

Klaus Marius Hansen[a,*], Kristjan Jonasson[b], Helmut Neukirchen[b]

[a] Department of Computer Science, University of Copenhagen, Njalsgade 128, Building 24, Floor 5, 2300 København S, Denmark
[b] Department of Computer Science, University of Iceland, Sæmundargötu 2, 101 Reykjavík, Iceland

ARTICLE INFO

Article history:
Received 19 July 2010
Received in revised form 1 December 2010
Accepted 20 February 2011
Available online 5 March 2011

Keywords:
Software architecture
Metrics
Product quality
Empirical study

ABSTRACT

Software architecture is concerned with the structure of software systems and is generally agreed to influence software quality. Even so, little empirical research has been performed on the relationship between software architecture and software quality. Based on 1141 open source Java projects, we calculate three software architecture metrics (measuring classes per package, normalized distance, and a new metric introduced by us concerning the excess of coupling degree) and analyze to which extent these metrics are related to product metrics (defect ratio, download rate, methods per class, and method complexity). We conclude that there are a number of significant relationships between product metrics and architecture metrics. In particular, the number of open defects depends significantly on all our architecture measures.

1. Introduction

It is often claimed that software architecture enables (or inhibits) software quality (cf. e.g., Perry and Wolf, 1992; Garlan and Shaw, 1993). An example would be that an architectural choice of a specific, relational database for an application implies quality constraints on performance, modifiability, etc. However, this claim has not been extensively validated empirically. While much work has focused on measuring software quality, little has focused on measuring software architecture. In the work described here, we investigated the software architecture of open source software projects, defined metrics for software architecture, and analyzed to which extent they correlated with software quality metrics. Specifically, the data that we collected was meta-data on 21,904 projects and source code from 1570 of these. All projects are Java projects and hosted on the SourceForge[1] repository. Based on the meta-data and source code, we computed and analyzed the results of various metrics. Our objective was to compare software architectures and product quality according to various perspectives on software quality.

Our main contributions are a new metric for modeling of coupling and the actual empirical study of software architectures' effect on product quality including the analysis of relationship between individual metrics using uni- and multi-variate models.

The rest of this article is structured as follows: first, we present some foundations in Section 2. Section 3 presents and discusses metrics on software quality and on software architecture. Next, Section 4 presents our operationalization process, in particular our study method, including how data was gathered and how metrics were calculated. Our analysis is presented in Section 5 and Section 6 summarizes and concludes our work.

2. Background

Our view on software quality originates in the work of Garvin (1984). Garvin defined a set of views on quality which are also applicable to software (Kitchenham and Pfleeger, 1996). The characteristics of quality in these views are:

- In the *transcendental* view, quality can be recognized but not defined. This is the view that is espoused by Christopher Alexander in his patterns work (Alexander, 1979) and to a certain extent in the software patterns literature (Gamma et al., 1995).
- In the *user view*, a system has high quality if it fulfills the needs of its users. This view is highly related to usability and is in line with "quality in use" as defined in the ISO 9126 standard (ISO/IEC, 2001) as shown in Fig. 1.
- In the *manufacturing* view, a product is seen as being of high quality if its development conforms to specifications and defined processes. This view is to a certain extent part of CMM(I) (C Product Team, 2006) or SPICE (ISO/IEC, 2004) and to the "process quality" concept briefly mentioned in ISO 9126 as shown in Fig. 1. In the sense of conformance to specifications, aspects of "external" quality related to faults are also related to this view.

* Corresponding author.
E-mail addresses: klausmh@diku.dk (K.M. Hansen), jonasson@hi.is (K. Jonasson), helmut@hi.is (H. Neukirchen).

[1] http://www.sourceforge.net.

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.02.037

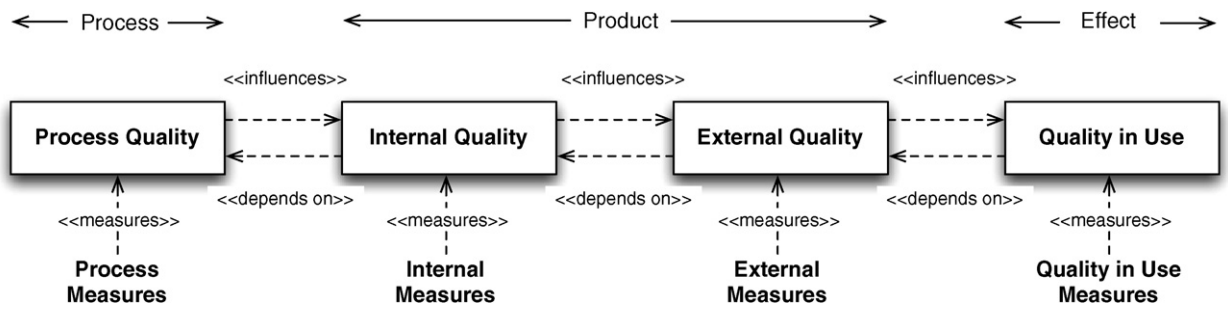

Fig. 1. ISO 9126 quality views (adapted from ISO/IEC, 2001).

- The *value-based* view equates quality to the amount a customer is willing to pay for a product.
- In the *product view*, quality is tied to properties of the product being developed. This is the primary view of "internal" and "external" quality in ISO 9126 as shown in Fig. 1.

Turning to software architecture, there are many definitions of software architecture.[2] An influential and representative definition by Bass et al. (2003) states that:

> The software architecture of a computing system is the structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them

In other words, software architecture is concerned with structures (which can, e.g., be development or runtime structures) and abstracts away the internals of elements of structures by only considering externally visible properties.

Recently, focus has also been on decisions made when defining system structures (Tyree and Akerman, 2005; Jansen and Bosch, 2005). This leads to definitions such as:

> A software system's architecture is the set of principal design decisions made about the system (Taylor et al., 2009).

We are here concerned with a large set of open source projects and thus necessarily have to rely on (semi-)automated analyzes. Thus we take the definition of Bass et al. as our basis for a definition of software architecture.

Concerning metrics to measure software quality, a huge set of metrics to chose from exists as metrics are widely practiced and researched (Kan, 2002). However, only few of these metrics are suitable to measure the quality of software architectural.

In principle, software architecture quality can be seen in any of the views of Garvin. As an example, Grady Booch is applying a value-based view in his selection of software architecture for the Handbook of Software Architecture.[3]

However, prevailing software architecture analysis methods (Dobrica and Niemela, 2002) tend to take a user-based or manufacturing-based view on software architecture quality. The Architecture Trade-off Analysis Method (ATAM; Kazman et al., 2000), e.g., aims at finding trade-offs and risks in a software architecture compared to stakeholder requirement. ATAM's focus on stakeholders gives it to a large extent a user-based quality view, but a manufacturing-based view is also included (e.g., in determining whether a specific trade-off is a potential risk). Architecture analysis methods do not often, however, include specific metrics on software architecture; rather they focus on the software architecture-specific parts of analyzes. Clements et al. (2002), e.g., describe metrics for complexity only (e.g., "Number of component clusters" and "Depth of inheritance tree" to predict modifiability and sources of faults).

Very little has been written specifically on metrics for software architecture. We looked systematically at papers from architecture-related conferences that contain metrics (Hansen et al., 2009). None of these metrics were appropriate for our purposes nor did we find any other publication that investigates empirically the effect of software architectures on product quality.

Applying statistical models and linear regression in the way we do, to investigate relationships between different metrics, is also novel in the software engineering literature, although it is common in some other disciplines.

3. Metrics

We divide the metrics that we consider into "product metrics" which are metrics related to software quality that are not architectural in nature and "architecture metrics" which are architectural in nature. Section 3.1 presents and discusses product metrics, Section 3.2 presents and discusses architecture metrics, while Section 3.3 summarizes our choice of metrics for this work.

3.1. Product metrics

We are concerned with metrics that can measure quality from any of the five views described in Section 2. With our data, we can measure quality (to some extent) from three of the views.

3.1.1. Metrics related to the manufacturing view

Here we can use defect count as a direct measure of quality to extent that defects are introduced during manufacturing.

Definition 1 *(Open Defect Ratio (ODR)).* The Open Defect Ratio (ODR) for a project p is the ratio of the number of open defects (plus 1) to the total number of open and closed defects (plus 1).

3.1.2. Metrics related to the value-based view

The value users put on an open source software project could be quantified indirectly in a number of ways: number of downloads of a project, usage count, communication about the project. Our data contains usage count, and we can use usage rate as a direct measure of quality:

Definition 2 *(Rate Of Usage (ROU)).* The Rate Of Usage (ROU) of a project p is the ratio of total number of downloads to the project age (in days).

We explicitly exclude payment since the projects we are concerned with can all be used without paying for the use.

3.1.3. Metrics related to the product view

In the product view, quality is not measured directly, but rather through measuring internal characteristics of the product. Basili

[2] http://www.sei.cmu.edu/architecture/start/definitions.cfm.
[3] http://www.handbookofsoftwarearchitecture.com.