Does Software Architecture Matter?

An empirical study of the effect of software architecture on open source software product quality

Klaus Marius Hansen
University of Iceland
Reykjavík, Iceland
kmh@hi.is

Kristján Jónasson
University of Iceland
Reykjavík, Iceland
jonasson@hi.is

Helmut Neukirchen
University of Iceland
Reykjavík, Iceland
helmut@hi.is

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 1,141 open source Java projects, we calculate three software architecture metrics (measuring classes per package, normalized distance, and degree of coupling) and analyze to which extent these metrics are related to defect ratio and download rate. We conclude that there are a number of significant relationships. In particular, the number of open defects depend significantly on all our architecture measures.

1. INTRODUCTION

It is often claimed that software architecture enables (or inhibits) software quality. 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 reported 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, i.e. whether the claim that software architecture enables software quality holds or not.

Concerning software architecture, many definitions can be found. An influential and representative definition by Bass et al. [1] 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. This leads to definitions such as:

> A software system's architecture is the set of principal design decisions made about the system [16].

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 software metrics, the IEEE Standard Glossary of Software Engineering Terminology" [11] provides the following definition:

> Metric: a quantitative measure of the degree to which a system, component, or product possesses a given attribute

Software metrics can be calculated based on source code, but also based on other meta-data, e.g., bug data bases or download statistics. In the remainder, we distinguish between metrics that aim at measuring attributes of software architecture and metrics that aim at measuring products, in particular product quality. While the later kind of metrics are widely practiced and researched [13], not much work on software architecture metrics has been published so far.

The main contribution of this paper is the analysis of a huge body of software projects and the found relationships between product quality metrics and software architecture metrics. Specifically, the data that we collected were meta-data on 21,904 projects and source code from 1,570 of these. By a further filtering of projects that were considered too small to be representative, 1,141 Java projects were investigated in detail. Based on the meta-data and source code, we computed and analyzed the results of various metrics. As part of this, we introduced the Degree of Coupling (DOC) metric for measuring the coupling aspect of a (package) architecture.

In this paper, we present just a summary of the most interesting findings of our investigation. For more detailed results that involve further metrics, please refer to our technical report that is available on-line [9].

The rest of this paper is structured as follows: following this introduction, Section 2 presents the used metrics and how the investigated data were gathered and processed. Our analysis of the obtained data is presented in Section 3. Finally, Section 4 discusses our results and concludes our paper.

2. MATERIALS AND METHODS

We analyzed data from Java projects in the SourceForge open source repository. In doing so, our method consisted of the following steps:

1. Gather data in the form of meta-data and source code of projects,
2. Apply metrics on gathered data
3. Analyze measures statistically
4. Discuss and conclude.

For readability, we describe our choice of metrics next (Section 2.1) followed by a description of data gathering and measurement (Section 2.2). Section 3 presents our statistical analysis and results.

2.1 Choice of Metrics

Our goal is to compare aspects of software architecture and software product quality. In distinguishing, we regard "software architecture" metrics as software metrics that pertain to overall structure, i.e., (in an object-oriented context) that do not measure individual objects, classes, or statements.

2.1.1 Architecture Metrics

We did a systematic literature review of the METRICS, MSR, and WICSA conferences [9], but found few specific architecture metrics. We, however, identified three metrics that we will use in the context of this paper; they are listed in Table 1. To compare across a number of projects, we need a single measure for each project. Concas et al. [7] note that many system properties may not be normally distributed, but we chose to average property measures since we do not have any a priori assumptions of the distribution of our measurements.

The AND metric in the table is based on "normalized distance" which is defined as being the absolute value of the sum of "instability" and "abstractness" minus 1, i.e., the value is normalized to be in the range 0 to 1 [15]. "Instability" is the number of outgoing dependencies from a package divided by the number of outgoing plus incoming dependencies. "Abstractness" is the number of abstract classes in a package divided by the number of abstract plus concrete classes. A related principle of [15] states that "normalized distance" should be close to 0 such that, e.g., if a package has high instability, then it should not be abstract.

Metric	Full Name	Explanation	Source
ACP	Average Classes per Package	The total number of classes divided by the total number of packages	[5] (adapted for software architecture)
AND	Average Normalized Distance	A measure of how abstract and instable packages are on average	[15]
DOC	Degree Of Coupling	The degree to which packages are coupled to other packages	Own, described more detailed in [9]

Table 1: Architecture metrics

We define the DOC metric based on the hypothesis that the more coupled a (package) architecture is, the harder a project is to maintain. If we consider the dependency graph of the packages of the project (and include dependencies from packages to themselves), a graph of n packages will have between n and n^2 edges. That is, the number of edges will be, $E = n^k$ with k between 1 and 2. The value of k should tend to 1 as the size of a project grows (since $k = \log E / \log n$, the average number of imports per package would otherwise grow out of limit with the project size).

We defined a model that has flexibility to describe our data while giving $k \approx 1$ for large n and having a finite limit for n going to 0. A simple function that fulfills this is a rational function on the form $1 + a/(1 + bn)$. To add flexibility, we add an exponent, c, on n, which together with an error term gives the model:

$$k = 1 + \frac{a}{1 + bn^c} + \frac{1}{\log n} \cdot \varepsilon \quad (1)$$

where ε is an $N(0, \sigma^2)$-distributed error term. Notice that the error term tends to 0 with increasing n.

Maximum likelihood estimation can now be used to determine the parameters a, b, c, and σ^2; we return to this in the results section, Section 3. DOC can then be defined as the model residual

$$\varepsilon = \log E - (1 + \frac{a}{1 + bn^c}) \log n \quad (2)$$

2.1.2 Product Metrics

We note that in [9], we also analyze product metrics related to the number of methods per class and cyclomatic complexity of methods.

For "product quality", we identified two metrics that we use in the context of this paper; Table 2 summarizes the product metrics. Specifically, we consider the number of (reported) open defects divided by the total number of (reported) defects in the project historically (ODR) and the number of downloads of the project per day (ROU). We thus look at quality from a manufacturing-based/product-based and a value-based perspective of quality [8].

Metric	Full Name	Explanation
ODR	Open Defect Ratio	The ratio of open defects to the total number of defects
ROU	Rate Of Usage	The number of downloads per month the project has existed

Table 2: Product metrics

2.2 Data Gathering and Metrics Calculation

We collected meta-data (such as name, number of developers, development status, project age, and number of downloads) of the 21,094 most highly ranked Java projects on 2009-03-17 from SourceForge. These characteristics are illustrated in Figure 1.

Based on the meta data we defined a set of "relevant" projects for which we attempted to gather source code. Source code was gathered by either downloading all modules (for those projects that use CVS for storing the source code) or assuming that the project followed the "trunk" best practice (for those projects that use SVN for storing the source code).

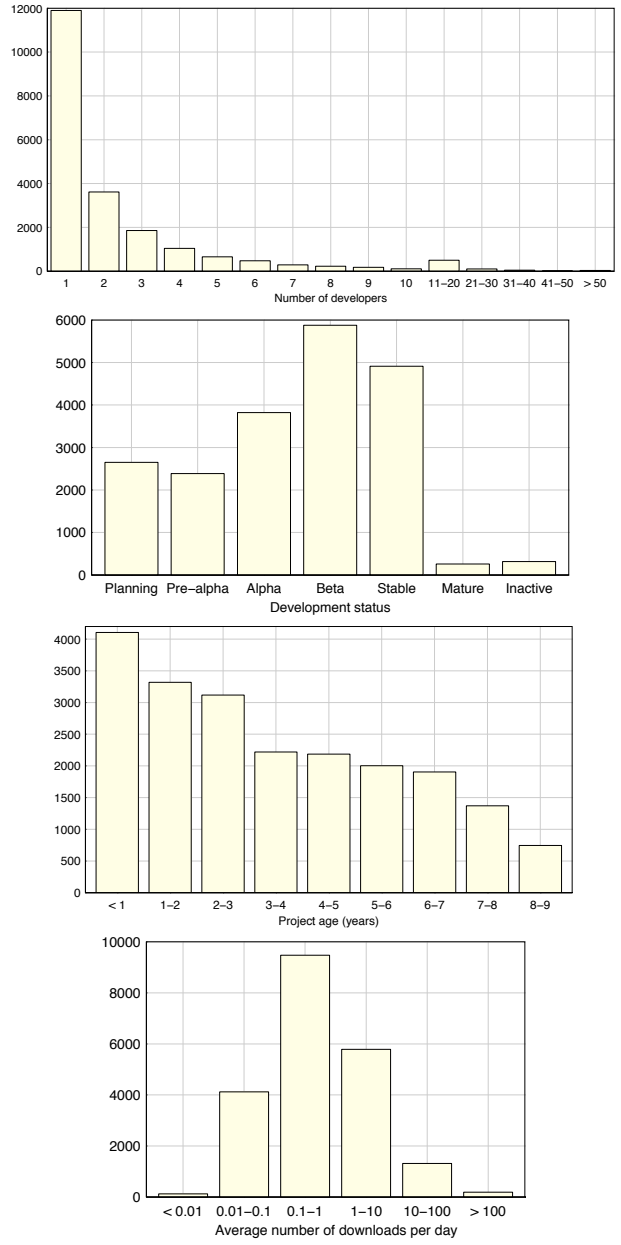

Figure 1: Meta-data characteristics of 21,904 SourceForge Java projects. Number of developers, development status, project age and download rate

	Relevance filtering ("All projects")	*Classification filtering ("Mature projects")*
Num of defects	≥ 1	≥ 1
Project age (days)	≥ 180	≥ 180
SLOC	≥ 2000	≥ 2000
Development status	4, 5, 6	5, 6
Downloads per day	≥ 2	≥ 7
Number of dev.	≥ 2	≥ 4
Total num of projects	*1,141*	*282*

Table 3: Filtering and classification summary

The download was done on 2009-03-30. For these projects we again applied a filter so that we ended up with two groups (where "mature projects" is a subset of "all projects") as shown in Table 3. For source lines of code (SLOC) we use physical source lines of code, which is the total number of non-blank, non-comment lines in the code. The criteria were chosen to end up with projects for which software architecture could be hypothesized to play a role.

Our metrics were calculated either directly from meta-data (for ODR and ROU) or based on the source code downloaded from SourceForge. We used four techniques to gather facts from project source code:

- We used Python and regular expressions on the contents of Java files to populate an (SQLite) database with data on public classes, packages, and "import"s. We only detect package level imports that are due to "import" statements
- We used SLOCCount[1] to calculate SLOC. This data is also put into a database
- We use JavaNCSS[2] to calculate cyclomatic complexity and method count of projects. This data is exported to Rigi Standard Format [17]
- Finally, we use a Java parser and analyzer (built upon the Java grammar included in the JavaCC[3]) to extract data on inheritance (and implementation) and on classes (and interfaces) also in Rigi format

Thus, effectively, we have two types of relational data sets: i) database relations and ii) Rigi relations. We initially worked with database relations only, but found out (in line with Beyer et al. [3]) that relational queries were inefficient in handling our data and thus also worked with data in Rigi format and used Crocopat [3] on this data.

3. RESULTS

We first estimated the parameters of our DOC model. Using equation (1) with the data for all the 1,141 projects used in the study, maximum likelihood estimation gives $a = 0.614$,

[1] `http://www.dwheeler.com/sloccount/`
[2] `http://javancss.codehaus.org/`
[3] `http://javacc.dev.java.net/`

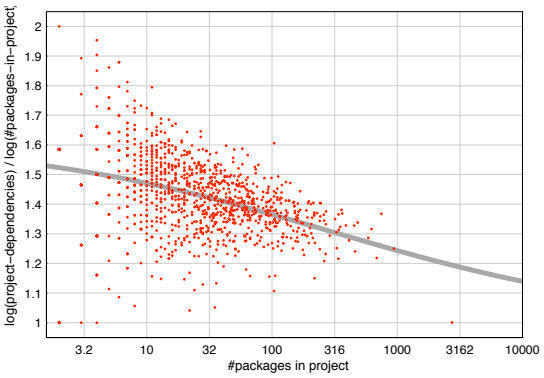

Figure 2: Scatter plot of the relationship between project size, n, and coupling exponent, k, for 1,141 studied projects. The gray line is the model (1) for the parameters estimated

$b = 0.136$, c=0.804 and $\sigma^2 = 0.0185$ yielding the model

$$k = 1 + \frac{0.614}{1 + 0.136n^{0.804}} + \frac{1}{\log n} \cdot \varepsilon \qquad (3)$$

where ε is $N(0, \sigma^2 = 0.0185)$. The model is illustrated in Figure 2 with data for the 1,141 projects we studied.

Next, for the gathered metrics data, Figure 3 shows histograms of of the architecture measures and Figure 4 shows histograms of the product measures. The raw values of two of the five metrics have highly positively skewed distributions. For one of these, ACP, it sufficed to take logarithms to produce approximately normal distributions (meaning that ACP is approximately log-normally distributed). For ROU the distribution was still quite skewed even after taking logarithms. This skew was removed using a Box-Cox power-transformation [4] (meaning that, approximately, ROU has a truncated power-normal distribution). Finally, we decided not to try to normalize ODR. We note that it is a metric that depends highly on each project's culture of defect reporting.

For each of the six pairs of architecture and product metrics we have investigated both a straight line and a parabolic linear regression model taking the architecture metric as an independent variable. Using a 5% significance level, all pairs gave a model significantly different from a constant model. Figure 5 shows a scatter plot of the metric pairs together with the models (in cases where the second order term was not significantly different from 0, a straight line model is given).

From the top left graph of Figure 5 we observe that projects with ACP around 15–20 (minimum is at 14.7 for all projects, and at 18.9 for most mature ones) have significantly fewer open defects than the projects where ACP is either low or high. In addition we see that the most mature projects have a lower defect ratio. The defect ratio also depends on AND in a similar way, for AND around 0.3 (minimum at 0.27 for all, 0.33 for mature) the defect ratio is on average significantly lower than for low and high AND values. It is in particular interesting to see that low AND values seem to give more defects, in view of the principle put forward in Martin [15], that good architecture should have normalized

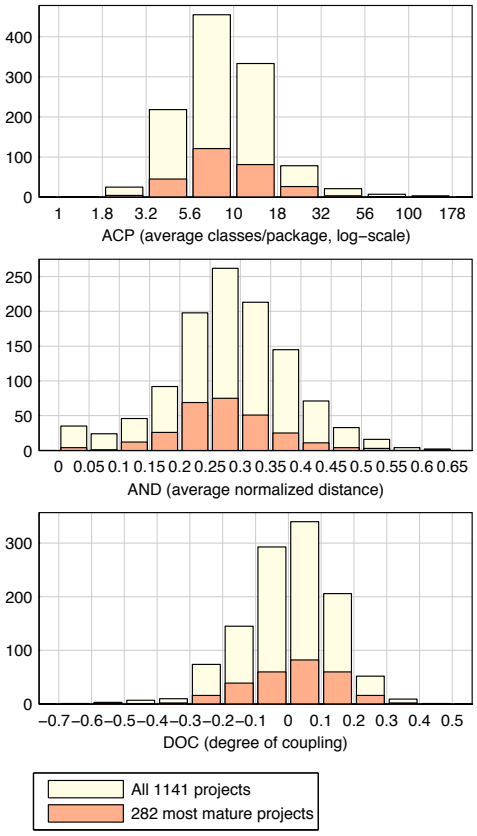

Figure 3: Distribution of measurements of architecture metrics (ACP, AND, DOC)

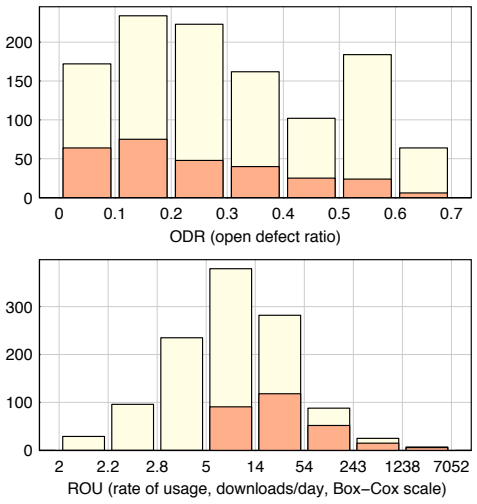

Figure 4: Distribution of measurements of product metrics (ODR, ROU)

distance close to zero. Regarding the pair DOC-ODR we see that for all projects the relationship is weak, but for the most mature projects, it seems again to be advantageous to have close to average coupling, rather than low or high (minimum is at 0.08 for mature).

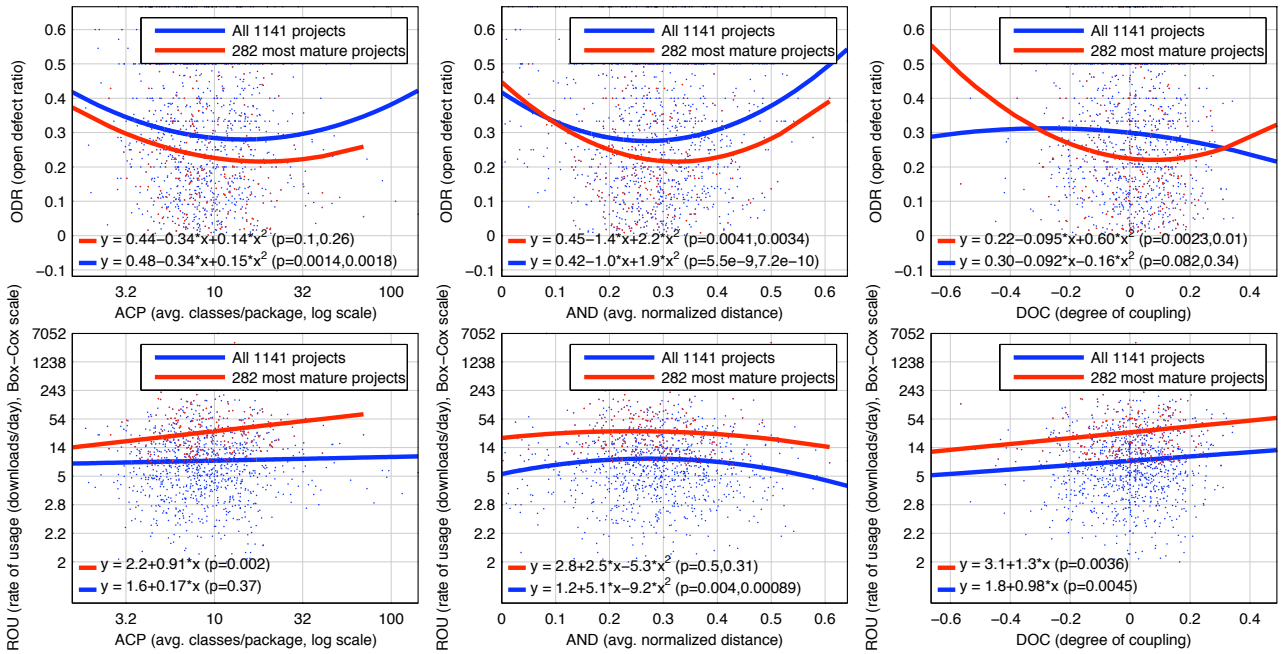

Figure 5: Analysis of relationships between product and architecture metrics (When two p-values are given the first applies to the x coefficient and the second applies to the x^2 coefficient)

Turning attention to ROU, we observe that the most downloaded projects among the mature ones have a tendency to have high ACP. The effect of AND on ROU matches its effect on ODR: For average AND there are significantly more downloads than when AND is low or high (the maxima occur at at 0.27 for all, 0.24 for mature projects). The DOC-ROU graph indicates that projects with low coupling tend to have fewer downloads. For the ACP and DOC relationships one might have expected the opposite effect, i.e. that fewer classes per package and low coupling might be beneficial.

In addition to the models shown in Figure 5, we have constructed multiple regression models using step-wise regression (see, e.g., [14]). The analysis showed that all architecture metrics are significant components of the models for ODR and ROU. Even though the p-values for these models show high significance since we analyze many projects, the R^2-values are not very impressive since there is a large amount of spread in our data. We discuss these results in [9].

3.1 Limitations

We here briefly discuss important limitations and threats to our study. These include

- The projects that we surveyed are all open source projects. Furthermore, it has been observed that many SourceForge projects are not active [2, 10]. In our analysis, we use projects where there is activity in terms of download. Even if a project has no activity it may still have a software architecture that is of interest to investigate
- Our analysis is automated. This means that we did not check, e.g., if the downloaded source code could compile. The large set of projects is meant to counter the effects of this
- We have analyzed a limited number of metrics (few architecture metrics appear available). Further, there is a current interest in software architecture research in non-product aspects of software architecture design, e.g., in design decisions [12] and organizations [6], something which our analysis does not regard

4. CONCLUSIONS

We classified metrics of (Java) software projects into architecture metrics, which try to measure the high-level design of software, and product metrics, which try to measure software implementation and the resulting quality. Five metrics were computed for a large body of open source Java projects, and subsequently analyzed statistically. To our knowledge this is the first study of this type.

The analyzed projects consist of 1,141 open source software projects selected from the SourceForge repository. Criteria for inclusion in the study included that the projects were pure Java projects and not brand new, used SourceForge to keep track of bugs, had at least 2000 source lines of code, had at least two developers, had been downloaded at least twice daily on average, and had reached development status beta. An addition we selected a subset of 282 "mature" projects, which had at least four developers, had been downloaded at least seven times daily, and had reached development status stable.

For both sets of projects (i.e., all 1,141, and the 282 mature ones) we constructed regression models for all pairs of product-architecture metrics. In all cases statistically significant relationships were discovered. The relationships are

in general stronger for the mature set of projects. For this set and the ODR metric, all three architecture metrics give rise to convex parabolic relationships, meaning that when these metrics give medium values, less error prone software results than when the metric values are extreme, whether low or high. ODR as predicted by the models ranges from a minimum of around 0.2 to a maximum of around 0.4. The relationship is similar for both ACP and AND in the larger project set.

In general, the effect of the architecture metrics on the product quality metrics agrees with what has been proposed in the literature. The most notable exception is AND. It was formulated as ideally being 0, but our results indicate that it is better to strive for a "compromise" on average when designing architecture, e.g., a value around 0.3. A similar tentative conclusion can be reached for ACP: to produce quality software one should aim for about ten classes per package on average. The effect of DOC on quality is more inconclusive.

In summary, we have presented evidence of an effect of architecture quality on product quality in a set of 1,141 open source Java projects. Further research is needed to be able to make predictions on a per-project basis, but the effect we have found is quite significant statistically, and may be relied on to draw conclusions about expected software quality given a set of projects.

References

[1] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2nd edition, 2003.

[2] K. Beecher, C. Boldyreff, A. Capiluppi, and S. Rank. Evolutionary success of open source software: an investigation into exogenous drivers. In *Proceedings of the Third International ERCIM Symposium on Software Evolution (Software Evolution 2007)*, pages 124–136, 2007.

[3] D. Beyer, A. Noack, and C. Lewerentz. Efficient relational calculation for software analysis. *IEEE Transactions on Software Engineering*, 31(2):137–149, 2005.

[4] G. Box and D. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B*, 26:211–252, 1964.

[5] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, Jun 1994.

[6] P. Clements, R. Kazman, M. Klein, D. Devesh, S. Reddy, and P. Verma. The duties, skills, and knowledge of software architects. In *Proceedings of The Sixth Working International IEEE/IFIP Conference on Software Architecture (WICSA 2007)*, pages 20–23, Jan. 2007.

[7] G. Concas, M. Marchesi, S. Pinna, and N. Serra. Power-laws in a large object-oriented software system. *IEEE Transactions on Software Engineering*, 33(10):687–708, 2007.

[8] D. A. Garvin. What does "product quality" really mean? *Sloan Management Review*, 26(1):25–43, 1984.

[9] K. M. Hansen, K. Jónasson, and H. Neukirchen. An empirical study of open source software architectures' effect on product quality. Technical Report VHI-01-2009, Engineering Research Institute, University of Iceland, July 2009. `http://www.hi.is/~kmh/doc/vhi-01-2009.pdf`.

[10] I. Herraiz, J. M. González-Barahona, and G. Robles. Determinism and evolution. In A. E. Hassan, M. Lanza, and M. W. Godfrey, editors, *Fifth International Workshop on Mining Software Repositories, MSR 2008 (ICSE Workshop), Leipzig, Germany, May 10-11, 2008, Proceedings*, pages 1–10. ACM, 2008.

[11] IEEE. *IEEE Standard Glossary of Software Engineering Terminology*, 1990. IEEE Std 610.12-1990.

[12] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA 2005)*, pages 109–120, 2005.

[13] S. Kan. *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2002.

[14] B. W. Lindgren. *Statistcal Theory*. McMillan, New York, 3 edition, 1976.

[15] R. C. Martin. Design principles and design patterns, 2000. `http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf`. Accessed 20 July 2009.

[16] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

[17] K. Wong. *Rigi User's Manual*. Department of Computer Science, University of Victoria, July 1996. `http://www.rigi.cs.uvic.ca/downloads/rigi/doc/user.html`.