VERKFRÆÐISTOFNUN
HÁSKÓLA ÍSLANDS

# An Empirical Study of Open Source Software Architectures' Effect on Product Quality

Klaus Marius Hansen, Kristján Jónasson,
Helmut Neukirchen

July 21, 2009

Report nr. VHI-01-2009, Reykjavík 2009

Klaus Marius Hansen, Kristján Jónasson, Helmut Neukirchen. An Empirical Study of Open Source Software Architectures' Effect on Product Quality,
Engineering Research Institute, University of Iceland, Technical report VHI-01-2009, July 2009

Engineering Research Institute, University of Iceland, Hjarðarhagi 2-6, IS-107 Reykjavík, Iceland

## 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 analyze to which extent software architecture metrics has an effect on software product metrics and conclude that there are a number of significant relationships. In particular, the number of open defects depend significantly on all our architecture measures. Furthermore, we introduce and analyze a new architecture metric that measures the density of the package dependency graph. Future research is needed to make predictions on a per-project basis, but the effects found may be relied on to draw conclusions about expected software quality given a set of projects.

# Contents

# 1 Introduction

It is often claimed that software architecture enables (or inhibits) software quality. 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 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. Specifically, the data that we collected was meta-data on 21,904 projects and source code from 1,570 of these. All projects are Java projects. Based on the meta-data and source code, we computed and analyzed the results of various metrics.

Our view on software quality originates in the work of [27]. Garvin defined a set of views on quality which are also applicable to software [38]. 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 [6] and to a certain extent in the software patterns literature [26]
- 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 [33]
- 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) [47] or SPICE [34] and to the "process quality" concept briefly mentioned in ISO 9126. In the sense of conformance to specifications, aspects of "external" quality related to faults is also related to this view
- 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

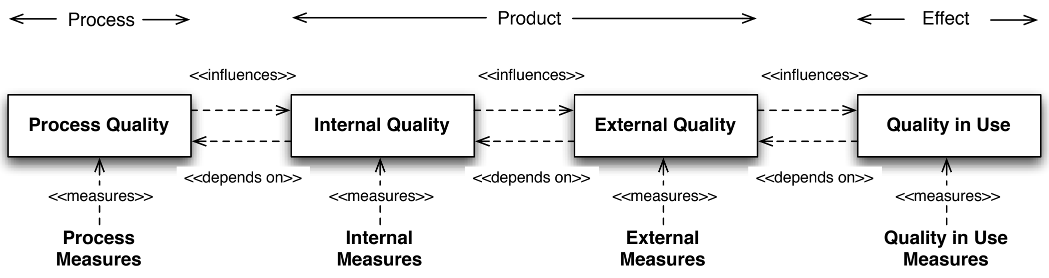

Figure 1.1: ISO 9126 quality views. (Adapted from [33])

Turning to software architecture, there are many definitions of software architecture. An influential and representative definition by Bass et al. [10] 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 [46].

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.

## Report Structure

The rest of this report is structured as follows: Chapter 2 presents and discusses metrics on software quality and on software architecture. Next, Chapter 3 presents our study method including how data was gathered and metrics calculated. Our analysis is presented in Chapter 4 and Chapter 5 discusses our results, future work, and concludes.

# 2 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 2.1 presents and discusses product metrics, Section 2.2 presents and discusses architecture metrics, while Section 2.3 presents our choice of metrics for this work.

## 2.1 Product Metrics

Metrics for software quality has been widely practiced and researched [36]. We are concerned with metrics that can measure quality from any of the five views described in Section 1. With our data, we can measure quality (to some extent) from three of the views.

### 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 given by:*

$$\mathrm{ODR}(p) = \frac{\mathrm{NumOfOpenDefects}(p) + 1}{\mathrm{NumOfOpenDefects(p)} + \mathrm{NumOfClosedDefects}(p) + 1}$$

One issue here is that failure reporting is not uniform across projects which implies that the data about defects may not be accurate or timely. This is, however, an important metric of quality and given that we analyze a large number of projects, the inaccuracies may even out. Furthermore, we exclude project that do not report defects (i.e., project where $\mathrm{NumOfOpenDefects}(p) + \mathrm{NumOfClosedDefects}(p) = 0$) using SourceForge in our analysis.

### 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))**

$$\mathrm{ROU}(p) = \frac{\mathrm{NumOfDownloads}(p)}{\mathrm{AgeOfInDays}(p)}$$

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

### 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 et al. [9] validated a set of design metrics originally proposed by Chidamber and Kemerer [17] as being useful in predicting fault-prone classes.

To limit the analysis, we consider one metric here that was found to significantly predict fault proneness[1]:

**Definition 3 (Weighted Methods per Class (WMC))** *The number of methods defined in a class multiplied by a weight for the each method*

We do not have fault data for specific classes in our data so we do not apply this class level metric directly, but rather average WMC over all classes. Furthermore, as Basili et al. we set the weight of each method to 1:

**Definition 4 (Average Methods per Class (AMC))** *The average number of methods defined in classes in a project*

Other product metrics include McCabe's cyclomatic complexity metric [41] and lines of code. While there has been considerable controversy surrounding these and other metrics (cf. e.g. [45]), the metrics are readily calculated and may be used together to provide a metric of "complexity density" [29]. The cyclomatic complexity of a program corresponds to the number of independent, linear paths through the control graph of the program.

**Definition 5 (Average Complexity Density (ACD))** *ACD for a project is the sum of the cyclomatic complexities for all methods in classes in the project, divided by the total number of methods.*

## 2.2 Software Architecture Metrics

In principle, software architecture quality can seen in any of the views of Section 1. As an example, Grady Booch is applying a value-based view in his selection of software architecture for the Handbook of Software Architecture[2].

However, prevailing software architecture analysis methods [24] tend to take a user-based or manufacturing-based view on software architecture quality. The Architecture Trade-off Analysis Method (ATAM; [37]), 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. [18], 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).

Moreover, very little has been written specifically on metrics for software architecture, however "high-level design" metrics or object-oriented design metrics can also to a certain extent be used for software architecture even though they often work on a detailed level (e.g., on specific classes and their methods and fields). In the following, we first look systematically at papers from architecture-related conferences that contain metrics and subsequently define a set of metrics for software architecture. Following Basili et al. [9] again, we may, e.g., define (analogous to WMC):

**Definition 6 (Average Classes per Package (ACP))** *The Average number of Classes per Package for a project is the total number of classes divided by the total number of packages*

[1]The validation was done using C++, not Java as in our case
[2]`http://www.handbookofsoftwarearchitecture.com`

### 2.2.1 Architecture Metrics from Architecture-Related Conferences

#### Papers from the Working International Conference on Software Architecture (WICSA)

Looking at the three latest WICSA conferences (2008, 2007, and 2005), only three papers mention metrics in their abstract: [44], [22], and [28]. Shaik et al. [44] analyzed two architectures using the "Change Propagation (CP) probability" metric and compared the results to using three coupling-based, object-oriented metrics ("Coupling Between Object Classes (CBO)", "Response For a Class", and "Message Passing Coupling"). The metric is defined in [4] and produces for every pair of elements, $e_i$ and $e_j$, a value, $cp_{ij}$, which is the probability that $e_j$ will change functionality given a change of functionality in $e_i$ and that the system as a whole does not change functionality. De Almeida et al. [22] present an approach for producing domain-specific software architecture in which metrics play a role. They do not, however, precisely state which metrics they are using. Similarly, the only mention of metrics by Giesecke et al. [28] is that the goal/question/metric quality model [8] should be used to define project-specific quality models in the context of exploring software architectures that use different middleware platforms.

#### Papers from METRICS

Judging from the abstracts of the METRICS series of symposia (2005, 2004, 2003, 2002, 2001, 1999, 1998, 1997, and 1996), only 10 out of more than 250 papers are concerned with (software) architecture [43, 39, 5, 12, 48, 25, 11, 42, 50, 7].

Nakamura and Basili [43] present a distance metric for architectural change of individual components using kernel methods. The metric is empirically validated using open source projects.

Li et al. [39] studied OTS (Off-The-Shelf) component usage through structured interviews involving 133 OTS component-based projects. One conclusion was that there was no evidence that components are chosen based on architecture compliance rather than functionality.

Abdelmoez et al. [5] provide metrics for estimating the probability that an error arising in one component propagates to another using architecture-level information on components and connectors.

Baudry et al. [12] consider "micro-architectures" through design patterns in object-oriented designs and introduce the concept of "testing anti-patterns" and a "testability grid" that highlights testability risks in choosing a particular micro-architecture.

Van der Hoek et al. [48] produce metrics for product lines architectures (focusing on service provision and service utilization metrics) that work on a description in an Architectural Description Language (ADL). The metrics specifically aims at giving meaningful data in the context of optionality and variability in product lines.

Evanco [25][3] uses a non-linear model of that relates a "composite complexity measure" to software defects. The model variables are complexity measures that reflect architectural decisions made during top-level design such as a derivative of Chidamber's and Kemerer's CBO metric.

[11] is a precursor to [12] that identifies potential testability weaknesses based on UML class diagrams.

Moses [42] analyzes measurement of subjective software attributes (which may include attributes related to software architecture) and establishes that module length may influence the measurement of such attributes.

[3] also notes that "software complexity cannot be specified by a single software characteristic"

Weyuker [50] uses the number of project-affecting issues found during architecture reviews to predict risk of failure. The metric was validated through the application on 36 large tele-communications projects. The results follow up on and simplifies the metrics proposed in [7].

**Papers from Mining Software Repositories (MSR)**

From the five MSR workshops/symposia (MSR 2008, 2007, 2006, 2005, 2004), searching in abstracts for architecture yields three papers that are concerned with (software) architecture (out of more than 125 papers) [52, 16, 49].

Yang and Riva [52] present scenarios for software architecture evolution analysis and an approach to extracting architecture models from repositories. Examples of scenarios are "Adding a feature", "Restructuring the design", and "Studying the evolution of one logical component". The authors note that correlating software metrics with architectural evolution will be of interest, but judging from citations in Google Scholar[4], it appears that the authors have not worked on this any further.

Breu et al. [16] mine Eclipse for functionality (implementing cross-cutting concerns) that does not align with its architecture. The mining uses formal concept analysis to compute complex cross-cutting concerns based on simple cross-cutting concerns (sets of methods where a call to a specific single method is added).

Wermelinger and Yu [49] analyze the evolution of Eclipse plug ins. Their findings include that most architectural changes take place in milestones and that there is a stable architectural core of Eclipse that has been present since the first release. The unit of analysis is Eclipse plug-ins for which meta-data is extracted (through plugin.xml for Eclipse's old plug-in architecture and through MANIFEST.MF for Eclipse's new, OSGi-based plug-in architecture) and the metrics applied are Number of Plugins (NP), Number of Extension points (NE), Number of Static Dependencies (NSD), Number of Dynamic Dependencies (NDD), and and Number of Unused Extension points (NUE).

### 2.2.2 Architecture Metrics Based on Martin

Martin[5] defines a set of principles and metrics related to (package) architectures. One of his principles is the following

**Definition 7 (The Dependency Inversion Principle (DIP))** *Depend on abstractions. Do not depend upon concretions.*

To measure adherence to this principle, Martin proposes three metrics:

**Definition 8 (INStability (INS))** *The number of outgoing dependencies (from classes) for a package divided by the sum of the number of outgoing and incoming dependencies of the package*

**Definition 9 (ABStractness (ABS))** *The number of abstract classes in a package divided by the sum of the number of abstract and concrete classes in the package*

**Definition 10 (NOrmalized Distance (NOD))** *The sum of instability and abstractness for a package, normalized to be in the range of 0 to 1, i.e., for a package p, NOD is* $|\mathrm{INS}(p) + \mathrm{ABS}(p) - 1|$

[4] http://scholar.google.dk/scholar?cites=8799306918464863868

[5] http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

A value of NOD close to zero indicates that if a package has many outgoing dependencies (INS is high) then it is not abstract (ABS is low) or vice versa. Martin states that if NOD is close to zero then "the package is abstract in proportion to its outgoing dependencies and concrete in proportion to its incoming dependencies".

Assume that for a package, $p$, NOD($p$) is close to zero. If $p$ has a high degree of incoming dependencies in relation to outgoing dependencies (i.e., INS is close to zero), then $p$ is highly abstract (ABS($p$) close to 1). On the other hand, if $p$ is highly concrete (ABS($p$) close to 0), then $p$ has a high degree of outgoing dependencies in relation to incoming dependencies. Thus a low NODs for a project can be said to indicate that the project follows the DIP.

In our case, we include interfaces in "abstract classes" in the ABS metric. Furthermore, for INS, we consider only dependencies expressed on a package level through "import" statements (realizing that this estimate may be slightly off).

Again, to get a project-level metric, we average the normalized distance over all packages in a project and get:

**Definition 11 (Average Normalized Distance (AND))** *AND for a project is the sum of NOD for all packages divided by the number of packages*

### 2.2.3 Further Architecture Metrics

We hypothesize that the more coupled an architecture is, the harder it is to maintain. The dependency graph of the packages of a project has a directed edge connecting two packages if a class from the first package imports the second package (or a class from that package). Thus a graph on $n$ packages will have at least $n$ edges and at most $n^2$ edges. It therefore seems natural to assume the graph having $E = n^k$ edges. We define the *coupling exponent* of a project as the exponent $k$, and attempt to find a model that describes how $k$ depends on $n$. We note that

$$k = \frac{\log E}{\log n} \qquad \text{with } 1 \le k \le 2$$

and that it is not unrealistic to assume that $k$ tends to 1 with increasing $n$ (if this does not hold then the average number of imports per package will grow without limit with project size). This means that $k$ is dependent on the size of projects and not directly usable as a metric (across projects of differing sizes). This is illustrated in Figure 2.1 for the 1,141 projects we studied.

We are thus faced with determining a model form which offers enough flexibility to describe available data, and which gives $k \approx 1$ for large $n$ with the added requirement that the function has a finite limit when $n$ goes to 0. One of the simplest functions to fulfill these requirement would be a rational function on the form $1 + a/(1 + bn)$. To allow a little more flexibility, we add an exponent, $c$, on $n$ together with an error term giving the model:

$$k = 1 + \frac{a}{1 + bn^c} + \frac{1}{\log n} \cdot \varepsilon \tag{2.1}$$

where $\varepsilon$ 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 $\sigma^2$; we return to this in the results section (Section 4). Based on this, we now define:

**Definition 12 (Degree Of Coupling (DOC))** *Given a set of projects, $S$, where package dependencies are modeled using (2.1), the Degree Of Coupling, DOC, of a project $p \in S$ with $n$ packages and*

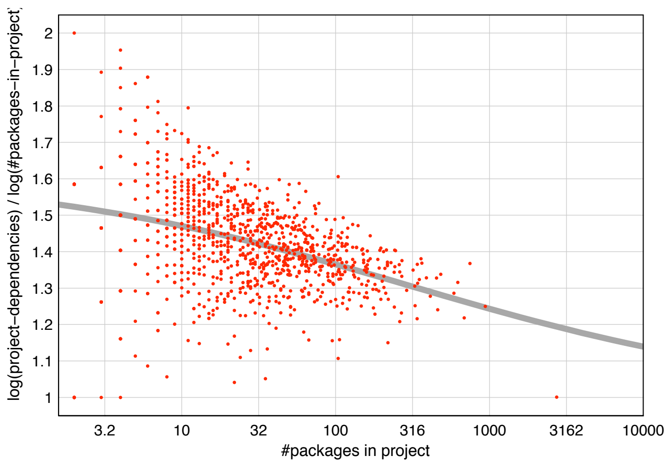

Figure 2.1: Scatter plot of the relationship between project size, $n$, and coupling exponent, $k$, for 1,141 studied projects. The gray line is the model (2.1) for the parameters estimated at the beginning of Section 4

*E dependencies among these packages is the model residual:*

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

*where a, b, and c have been estimated with maximum likelihood according to (2.1)*

## 2.3 Choice of Metrics

We here summarize the product and architecture metrics that we have chosen to analyze further in Table 2.1 and Table 2.2 respectively.

| 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 |
| AMC | Average Methods per Class | The total number of methods divided by the total number of classes |
| ACD | Average Complexity Density | The sum of cyclomatic complexities for all methods divided by the number methods |

Table 2.1: Product metrics

| **Metric** | **Full Name** | **Explanation** |
|---|---|---|
| ACP | Average Classes per Package | The total number of classes divided by the total number of packages |
| AND | Average Normalized Distance | A measure of how abstract (ratio of abstract classes/interfaces to concrete classes) and instable (ratio of outgoing dependencies to all dependencies) packages are on average |
| DOC | Degree Of Coupling | The degree to which packages are coupled to other packages |

Table 2.2: Architecture metrics

# 3 Materials and Methods

Our material is projects on SourceForge[1]. We focus on Java projects since this makes metrics calculation uniform and we hypothesize that statistical correlations are more likely to hold within similar projects. It has been observed that many projects on SourceForge have little activity [32, 13]. In our analysis, we use projects where there is activity in terms of download and furthermore if a project has no activity it may still have a software architecture that is of interest to investigate.

Our method can be divided into three steps:

1. Gathering data on projects, which involved
   (a) Gathering meta-data on projects
   (b) Filtering projects based on meta-data
   (c) Gathering source code for projects
   (d) Filtering projects based on source code
2. Measuring filtered projects by applying selected metrics
3. Statistically analyzing measurements

We describe step 1 ("Data Gathering") and step 2 ("Project Measurement") next. Step 3 (analysis and results) is described in detail in Chapter 4.

## 3.1 Data Gathering

### 3.1.1 Meta-data Gathering

We first collected meta-data on the 21,094 most highly ranked Java projects on 2009-03-17 from SourceForge for which it was possible to get such data. Here "Java projects" were defined as projects belonging to "trove" 198 at SourceForge and "rank" was the SourceForge ranking of projects. The data consisted of characteristics such as number of bugs, time of latest file upload, number of developers, number of open bugs, and SourceForge "rank".

Below is an example record for the most highly ranked Java project, "Sweet Home 3D" showing the characteristics that were used in our analysis.

| | |
|---|---|
| name | sweethome3d |
| url | `http://sourceforge.net/projects/sweethome3d` |
| bugs_closed | 124 |
| bugs_open | 21 |
| development_status | 5 |
| downloads | 2441636 |
| latest_file | 2009-03-13 |
| no_developers | 8 |
| registered | 2005-11-07 |
| repository_modules | ["SweetHome3D"] |
| repository_type | cvs |

Figure 3.1 to 3.4 show the distribution of the number of developers, development status, project age, and download characteristics for these projects

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

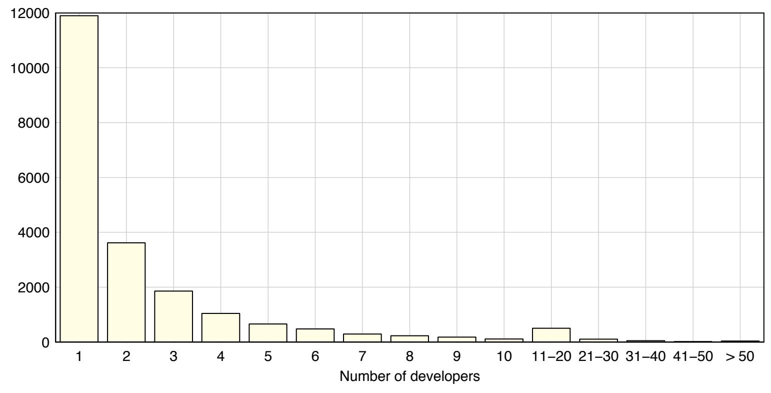

Figure 3.1: Number of developers per project for all projects

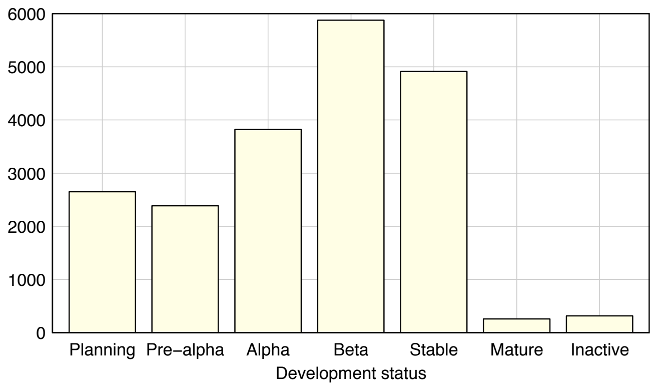

Figure 3.2: Development status of projects for all projects

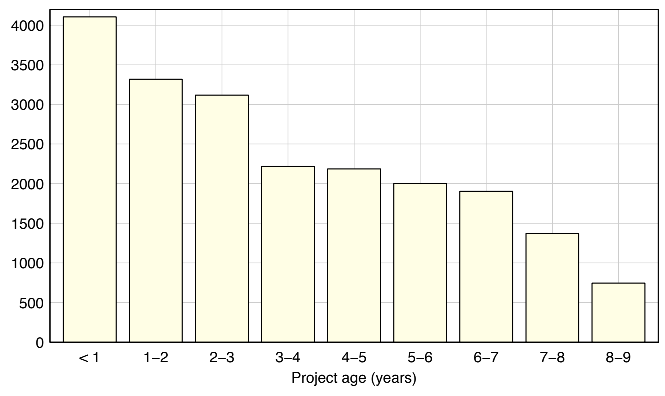

Figure 3.3: Project age for all projects

### 3.1.2 Filtering Based on Meta-data

Based on the meta-data, we defined a set of *relevant* projects, i.e., projects amenable to our analyzes. To be relevant, a project had to:

- *Keep track of bugs* using SourceForge. We defined this as `bugs_closed` + `bugs_open` being greater

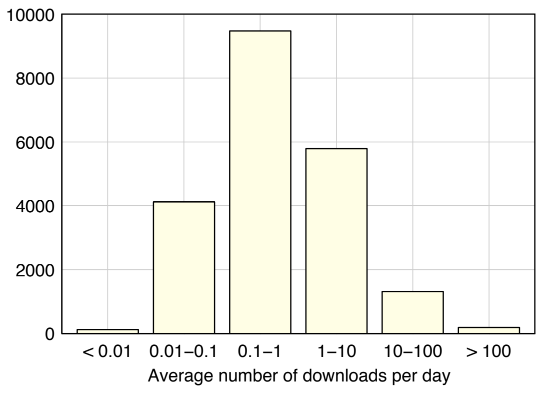

Figure 3.4: Download rate for all projects

than zero. For Sweet Home 3D, the sum is 145 and 12,743 projects did not use SourceForge to keep track of bugs

- Have a reasonable *download rate*. We defined this to be downloaded at least 2 times a days over the project history. For Sweet Home 3D, the download rate estimated on 2009-03-17 was, e.g., approximately 2,441,636/(2009-03-17 - 2005-11-07 days) = 1,191 downloads per day. 16,159 projects did not fulfill this criterion
- Have a sufficient *number of developers* to warrant a focus on software architecture in the project. We defined this as `no_developers` being at least 2. Sweet Home 3D, e.g., had 8 developers. 11,950 projects had less than two developers
- Have sufficiently *advanced development status*. We defined this as having a SourceForge `development_status` of at least 4 which is "beta" status. The status of Sweet Home 3D, is "5" which is "stable". 10,048 projects did not fulfill this
- Have a *development history*. We defined this as `registered` being at least 180 days ago at the time our analysis was made. On 2009-03-17, Sweet Home 3D was, e.g., 1,226 days old. 2,412 projects were too young

The two first criteria are important in that we want to define various quality measures for the projects. The result of this filtering was 1,570 Java projects.

### 3.1.3 Source Code Gathering

We attempted to download source code for the filtered projects on 2009-03-30 or revisions with date stamp 2009-03-30[2]. For projects that used CVS as configuration management tool, we downloaded all current CVS modules. For projects that used SVN as configuration management tool, we assumed that the project used the recommended "Trunk" repository layout [20]. Furthermore, we did not follow external SVN references. This means that we either i) downloaded all the top level "trunk" directory if there was one or ii) attempted to download all "dir/trunk" directories (where "dir" is a top level directory) if there was no trunk top level directory.

After source code download, we deleted all non-Java files since that data is irrelevant for our analysis. In total, 3.3 GB of data and 550,198 Java files were downloaded.

Since our analysis requires source code, we further filtered based on the available number of lines of code. We set 2,000 SLOC[3] as the limit; research by Zhang et al. [53] indicates that for open source Java projects, the average SLOC per class is around 100 yielding 20 classes as the limit in our case. We also removed three projects for which our metrics could not be calculated. This further reduced the number of relevant projects by 429 leaving 1,141 projects.

[2]This was done through "-D2009-03-30" for CVS and "-r2009-03-30" for Subversion

[3]SLOC: physical source lines of code, which is the total number of non-blank, non-comment lines in the code

### 3.1.4 Classification Filtering

We next classified a subset of the relevant projects as mature by further requiring that development status should be at least "stable", that there should be at least four developers, and there should be more than 7 downloads per day.

Table 3.1 summarizes our filtering and the "relevant" projects are listed in Appendix A.1 while the "mature" projects are listed in Appendix A.2.

| | *Relevance filtering ("All projects")* | *Classification filtering ("Mature projects")* |
|---|---|---|
| Number of bugs reported | $\geq 1$ | $\geq 1$ |
| Project age (days) | $\geq 180$ | $\geq 180$ |
| SLOC | $\geq 2000$ | $\geq 2000$ |
| Development status | 4, 5, 6 | 5, 6 |
| Download rate (downloads per day) | $\geq 2$ | $\geq 7$ |
| Number of developers | $\geq 2$ | $\geq 4$ |
| *Total number of projects* | *1,141* | *282* |

Table 3.1: Filtering and classification summary

## 3.2 Metrics Calculation

We use four techniques to gather facts from project source code:

- We use 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 use SLOCCount[4] to calculate the physical source lines of code of projects. This data is also put into a database
- We use JavaNCSS[5] to calculate cyclomatic complexity and method count of projects. This data is exported to Rigi Standard Format [51]
- Finally, we use a Java parser (built upon the Java grammar included in JavaCC[6]) to extract data on inheritance (and implementation), on classes (and interfaces), and on methods. We do a simple semantic analyzes that only uses the current project as classpath to qualify references. Furthermore, we do not take enums or generics into account

Thus, effectively, we have two types of data sets: i) relational data and ii) Rigi data. We initially worked with relational data, but found out (in line with Beyer et al. [14]) that relational queries were inefficient in handling our data and thus also worked with data in Rigi format.

With the relational data, we use simple relational queries, e.g., to calculate the number of dependencies between distinct packages in a project. With the Rigi data, we use Crocopat [14] to, e.g., calculate ABS, INS, and NOD. The end result is in both cases metrics and numbers that can be used directly in our statistical analyzes.

[4]http://www.dwheeler.com/sloccount/
[5]http://javancss.codehaus.org/
[6]https://javacc.dev.java.net/

# 4 Results

We now turn to our analysis of the gathered metrics data. We first estimate the parameters of our DOC model (Section 4.1), then in Section 4.2 and 4.3 we construct and discuss several linear regression models involving the metrics. Finally in Section 4.4, we discuss some limitations of our analysis.

## 4.1 Modeling of Coupling

Using (2.1) with the data for all the 1,141 projects used in the study, maximum likelihood estimation gives $a = 0.614$, $b = 0.136$, $c$=0.804 and $\sigma^2 = 0.0185$ giving the model

$$k = 1 + \frac{0.614}{1 + 0.136 n^{0.804}} + \frac{1}{\log n} \cdot \varepsilon \tag{4.1}$$

where $\varepsilon$ is $N(0, \sigma^2 = 0.0185)$.

We have already shown this model in Figure 2.1 in Section 2.2.3. It is also instructive to see directly how $\log E$ depends on $\log n$. This relationship is depicted in Figure 4.1.

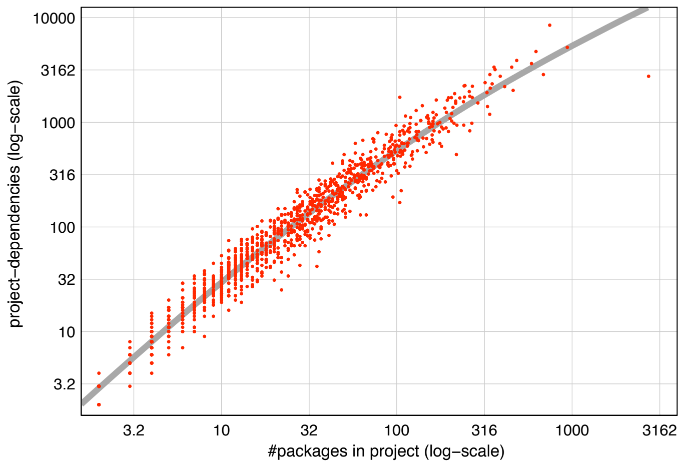

Figure 4.1: Scatter plot of the relationship between number of packages in project, $n$, and number of edges, $E$, in the package dependency graph for 1,141 studied projects. The gray line is the model (4.1) times $\log n$

## 4.2 Pairwise Regression Models

Turning to the distribution of calculated metrics, Figure 4.2 shows histograms of of the product and architecture measures. The raw values of four of the seven metrics have highly positively skewed distributions. For two of these, AMC and ACP, it sufficed to take logarithms to produce approximately normal distributions (meaning that AMC and ACP are approximately log-normally distributed), but for ACD and ROU the distribution was still quite skewed even after taking logarithms. For these we removed the skew using a Box-Cox power-transformation [15]. The transformation is

$$y = \frac{(x + \alpha)^{\lambda}}{\lambda}$$

where x is the raw variable and y is the transformed variable. The parameters $\alpha$ and $\lambda$ were estimated by maximizing the normal likelihood over all 1,141 projects, giving $\alpha = -0.038$, $\lambda = -1.94$ for ROU and $\alpha = -2.34$, $\lambda = 2.52$ for ACD. Furthermore, it makes sense that ODR is not easily normalized since this is a metric that depends highly on each project's culture of bug reporting.

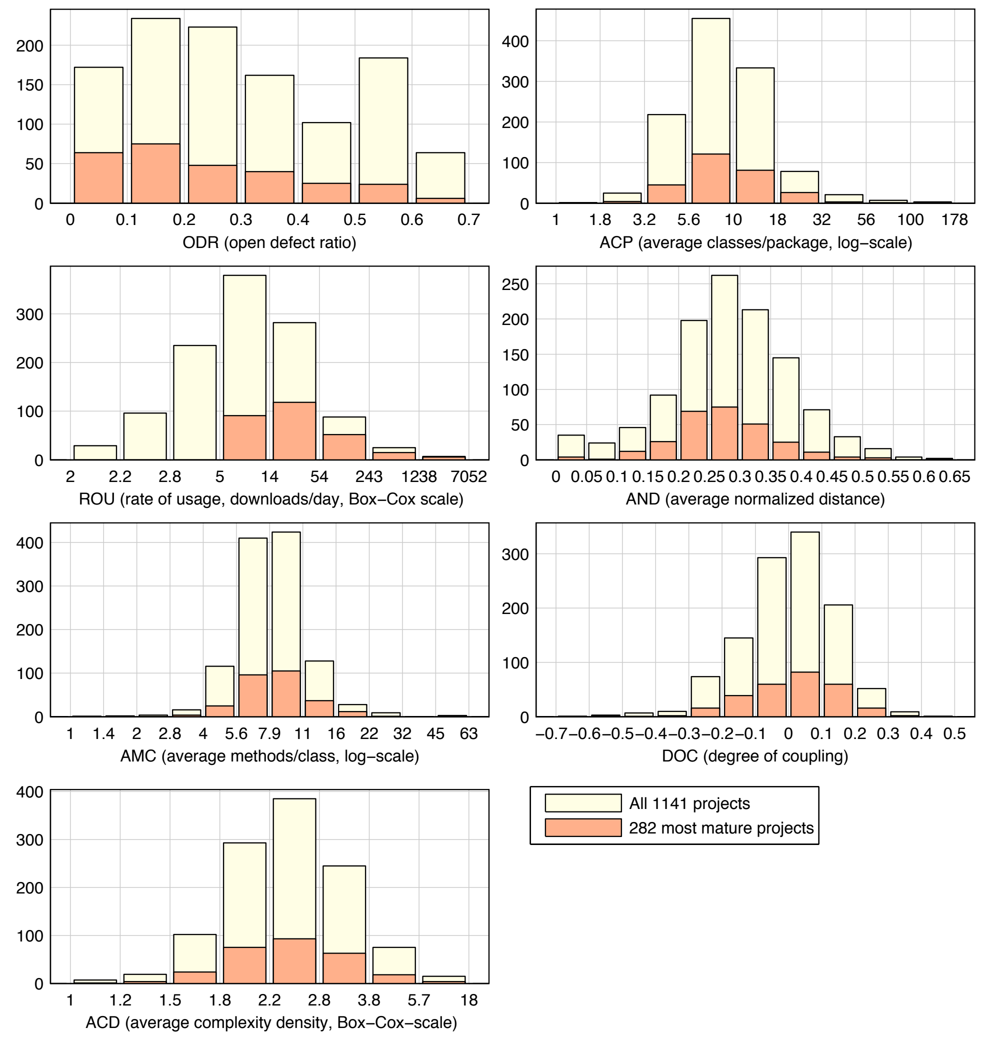

Figure 4.2: Distribution of measurements for all and mature projects

The histograms in Figure 4.2 indicate that after transformation all the variables are essentially skew-free, and all except ODR are approximately normally distributed. To investigate normality further, the number of projects out of the total of 1,141 with a variable that is more extreme than 2 and 3 standard deviations from the mean have been counted. These counts are shown in Table 4.1 together with the counts that a pure normal distribution would give.

Note that ACP has a little heavy right tail, and DOC has a heavy left tail, but apart from that the normality assumption holds reasonably well. This indicates (approximately) that AND and DOC are normally distributed, that AMC and ACP are log-normally distributed, and that ROU and ACD have a truncated power-normal distribution.

For each of the 12 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.

| | $< -3\sigma$ | $< -2\sigma$ | $> 2\sigma$ | $> 3\sigma$ |
|---|---|---|---|---|
| *Normal* | *1.5* | *26* | *26* | *1.5* |
| box-cox(ROU) | 0 | 32 | 30 | 4 |
| log(AMC) | 5 | 19 | 32 | 8 |
| box-cox(ACD) | 4 | 23 | 26 | 5 |
| log(ACP) | 0 | 12 | 35 | 11 |
| AND | 0 | 48 | 27 | 2 |
| DOC | 10 | 30 | 15 | 1 |

Table 4.1: Counts of Extreme Values of Metrics, All Projects

Using a 5% significance level, all pairs gave a model significantly different from a constant model. Figure 4.3 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 4.3 we observe that projects with ACP around 15–20 (minimum is 14.7 for all projects, 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 (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, that good architecture should have normalized 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 0.08 for mature).

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.

The remaining two dependent variables are the internal product metrics AMC and ACD. For high quality their values should be low. For mature projects the effect of ACP on these metrics matches the effect of ACP on defect ratio: average ACP is beneficial (minima at 6.7 and 7.7 respectively). For all projects the effect is less pronounced. Regarding the effect of AND and DOC on ACD, we again find the effect to be as expected, and similar to the effect of these metrics on ODR (minima at 0.35 and −0.04 for all projects; 0.42 and 0.01 for mature ones). The dependence of AMC on AND appears minimal, but its dependence on DOC is however as one might have expected: both low AND and low DOC should be favorable.

## 4.3 Multiple Regression Models

In addition to the models shown in Figure 4.3, we have constructed multiple regression models using step-wise regression (see e.g. [40]). The form of the resulting models is given in Table 4.2, which lists the model variables in the order that they enter the regression. For example, when constructing the ODR model, DOC turned out to be the most significant explanatory variable. When a (squared) variable is not present (for example DOC$^2$ in the ROU model) this is because the corresponding model term turned out to be non-significant. As an example, the complete ODR model for all projects is

$$ODR = 0.572 - 0.029 \cdot DOC - 0.15 \cdot DOC^2 - 0.27 \cdot \log(ACP) + 0.11 \cdot \log(ACP)^2 - 1.02 \cdot AND + 1.87 \cdot AND^2$$

The columns headed $R^2$ show the proportion of the total variance of the dependent variable which is explained by the models.

| Dependent var. | Model var. | All, $R^2$ | Mature, $R^2$ |
|---|---|---|---|
| $ODR$ | $DOC, DOC^2, \log(ACP), \log(ACP)^2, AND, AND^2$ | 4.3% | 7.4% |
| box-cox($ROU$) | $AND, AND^2, DOC, \log(ACP)$ | 1.5% | 5.7% |
| $\log(AMC)$ | $\log(ACP), \log(ACP)^2, DOC, AND$ | 3.3% | 12.9% |
| box-cox($ACD$) | $AND, AND^2, \log(ACP), \log(ACP)^2$ | 2.8% | 5.6% |

Table 4.2: Regression Models

It is interesting to note that except for the ACD-model, all the architecture metrics are significant components of the models. This means that their combined effect on the corresponding product metric is larger than their effect in the simple regression models shown in Figure 4.3. However it must be admitted that the $R^2$-values are not very impressive. Even though the $p$-values are highly significant (since many projects have been analyzed), there is a large amount of spread in our data. The models can be used to predict average product metrics with confidence, but they would not be very useful to predict metrics for single projects.

## 4.4 Limitations

There are a number of important limitations to our study, including

- The projects that we surveyed are all open source projects and moreover open source projects that are hosted on SourceForge. While the results may not be generalizable to closed source projects, this points to an area of further research
- Our analysis across projects is based on average values. Concas et al. [21] argue that system properties (such as WMC) often follow a power law or a log-normal distribution. Thus it may be problematic to work with the mean (or standard deviation) of these properties to characterize whole systems or projects. While working with means (or standard deviations) may thus be representative of a known distribution, we did not assume specific distributions. Further research could look also at specific distributions of these metrics for the projects investigated
- The analysis is automated. This pertains to the former point and means that we did not check, e.g., if the downloaded source code could compile or if bug reporting was consistent across projects. The large set of projects is meant to counter the effects of this. Again, this points to further research: with added resources, a large set of projects could be checked for consistency (in terms of source code, bug reporting etc.) and be used as a repository for this type of research
- The DOC metric is dependent on the set of changeable parameters, $a$, $b$, and $c$ that are estimated based on a particular set of projects. It would be worthwhile to try to simplify the DOC model to obtain some sort of measure that would be reasonably size-independent, but simpler than the current DOC definition, and hopefully more likely to apply to other project sets.
- We have analyzed a limited number of metrics. In particular, the range of available architecture metrics appears limited and further research would be needed in that area. In relation to this, there is a current interest in software architecture research in non-product aspects of software architecture design, e.g., in design decisions [35] and organizations [19]

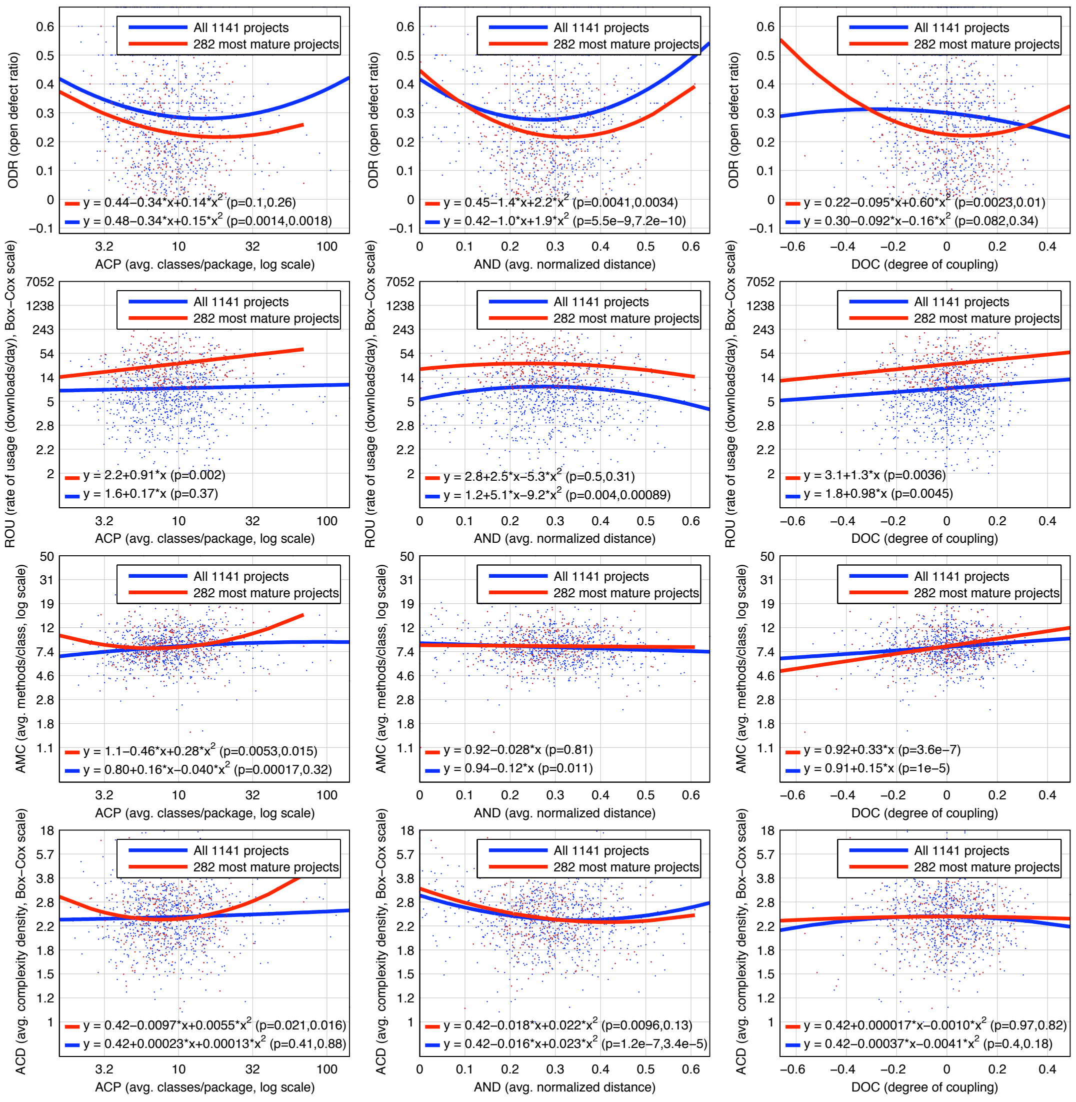

Figure 4.3: Analysis of relationships between product and architecture metrics (When two $p$-values are given the first applies to the $x$ coefficient and the second applies to the $x^2$ coefficient)

# 5 Discussion

We carried out a fairly comprehensive review of metrics on software design and quality that have been proposed in the scientific literature, especially as applied to (large) Java projects. We classified these metrics into architecture metrics, which try to measure the high-level design of software, and product metrics, which try to measure software implementation. Seven 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.

An important issue in software architecture is that of package coupling, i.e. the degree to which the packages of a project depend on one another. We hypothesize that the dependency graph becomes sparser and sparser with project size. We have modelled the effect as $E = n^k$, where $n$ is the number of packages and $E$ is the number of edges in the graph, and find that for small projects $k$ is around 1.5 and for the largest projects that we analyzed it is around 1.25. Our model then assumes that k tends to 1 with increasing $n$. One of our architecture metrics, DOC, is based on this model, but the others (classes per package, ACP, and normalized distance, AND) are based on previously proposed metrics. As product metrics we computed open defect ratio, ODR, rate of usage, ROU, methods per class, AMC, and cyclomatic complexity, ACD, but all of these are (or have been proposed to be) measures of software quality. For six of these metrics (all but ODR) we established an approximate probability distribution, valid for our data set.

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 12 pairs of product-architecture metrics as well as multiple regression models for all three architecture metrics. In all cases statistically significant relationships were discovered. The relationships are in general stronger for the mature set. For this set and ODR 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.

There is also a significant relationship between the architecture metrics and the other product metrics ROU, AMC and ACD. For the mature set, medium values of ACP go together with low values of AMC and ACD (both pointing to high quality), and for both project sets medium values of AND give high ROU and low ACD (again pointing to high quality). In other cases the effect is not as conclusive, and in a few cases it is even counterintuitive (in particular for the pairs DOC-ROU and DOC-AMC).

In general, the effect of the architecture metrics on the product 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 10 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.

# A Appendix

## A.1 List of All Relevant Projects

a2j, aceoperator, acmus, acqlite, adempiere, adito, aft, aglets, akrogen, amtu, ant-contrib, ant4eclipse, antforms, antichess, antlreclipse, antrunner, apelon-dts, aperture, apg, applecommander, arbaro, arch4j, architecturware, archive-crawler, argunet, arianne, arsenal-1, art, ashkelon, asmplugin, asterfax, astroinfo, astyleclipse, atla, atrisframework, avignon, avis, avr-eclipse, azsmrc, aztsearch, babeldoc, balie, bananasplitter, barbecue, barcode4j, basegen, basex, basicportal, bazaar, beanlib, beauty-hair-mng, beepcore-java, benerator, beyondcvs, bie-gpl, bigzip, bioweka, blax, blogbridge, bloof, bluemusic, blueskytime, bodington, bootchart, borg-calendar, botsnscouts, bplusdotnet, bpmspace, bracket-tracker, bsheet, bt-thud, bt747, buddi-plugins, butler, butterflyxml, buttress, byronstar-sl, camprocessor, carbonado, cardwizard, care2002, castordoclet, cb2java, cc-config, ccdtovcd, cctools, cdk, cdox, centraview, cese, cewolf, cglib, cgupnpjava, chaperon, charliebot, chateverywhere, checkclipse, checkstyle, chi, cilib, cmakeed, cml, cmpcs, cobalt, cobertura, cocodonkey, codecover, codesugar, coefficient, cofax, coffee-soft, cogengine, cogroo, colladarefinery, collections, colorer, complat, components4oaw, comsuite, concierge, conduitmanager, conexp, controlremote, controltier, coopnet, coras, coreasm, corewar, cosi-nms, coverlipse, csbi, csql, cssparser, csvjdbc, csvtosql, ctl-dispatch, cuecreator, cvsgrab, d3web, daffodilcrm, daffodildb, dashboard, databionic-esom, datavision, davenport, dbfit, dbmonster, dbmt, dbprism, dbunit, dbxml-core, dctm, dddsample, dhcp4java, dictionarymaker, digir, digisimulator, dimdim, dimensionex, discarchiver, distributor, djproject, docsearcher, dom4j, domingo, donnerlaparole, dooble, dotplot, dozer, dr-cube, dragmath, dragon-char, dragonchess, dresden-ocl, drjava, droid, druid, dsol, dspace, dubman, dump3, dynamicjasper, e-p-i-c, eastwood, easydesigner, easyos, easysql, easystruts, ebeanorm, ebjava, ebookmanager, ebookme, ebxmlrr, echo, echopoint, eclemma, eclibatis, eclipse-javacc, eclipse-rbe, eclipse-tools, eclipsejdo, eclipseproject, eclipsesql, eclipsetail, eclipsetidy, eclipsewiki, eclipsexslt, ecut, egonet, egothor, ehcache, eiffel-mas, eimp, ejb3unit, ejbca, ejbgen, el4j, elevatorsim, elexis, elips, elml, eln, elvyx, emma, emonic, emulinker, entagged, entrainer, epp-rtk, epp-ver-04, eps, eressea, erlide, escher, etex, eug, eulersharp, eurobudget, evaristo, evetrader, eworld, exist, exo, eyedb, facecart, fedora-commons, filebunker, fina, findbugs, finj, fipa-os, firemox, fitnesse, fitpro, flashrecruit, flatpack, flesh, flexjson, flexstor, flickrbackup, floggy, fluid, fnr, foa, fobs, follow, formproc, foucault, fourever, foxtrot, freecs, freedom-erp, freehost3270, freelims, freemarker, freemarker-ide, freemercator, freemind, freesudoku, freetts, freewrl, frinika, ftp4che, funambol-gmail, furthurnet, fuzzymath, galleon, gametable, ganymede, gate, gateway, gatormail, gdapi, gdbi, gebora, geekblog, gems, gendiapo, genj, genoviz, georaptor, gestdb, getittogether, gface, gfd, gham, gild, gistoolkit, gjtapi, glassbox, gml4j, gmod, gngeofrontend, gnuaccounting, gogui, gomule, gpe4gtk, gpsmap, graysky, greasyspoon, greatcow, greenmail, gridsam, gridsim, grinder, groimp, grok, gruntspud, gsa-japi, gsbase, gsn, gtads, gted, gui4j, gvf, gwanted, h3270, hammurapi, hansel, hartmath, henplus, hibernate4gwt, hibernatesample, high-scale-lib, hipergate, hl7api, holongate, hsqldb, htmlparser, htmlunit, httpunit, humaitrader, hypercontent, hypergraph, ical4j, id3v2-chap-tool, idea-arch, identitymngr, idiet, ihm, ij-plugins, ikvm, impact, importscrubber, incanto, inforama, informa, ingenias, insenvim, ion-cms, ipdr, ireport, iscreen, iseries-toolkit, isis, isql, iteraplan, itext, itracker, ivalidator, ivc, j-algo, j-ftp, j-twat, j2cstranslator, j2ep, j2meunit, j2s, j4fry, j4sign, jaaslounge, jabaserver, jabberapplet, jabook, jabref, jacareto, jace, jackcess, jacob-project, jadclipse, jaffa, jag, jaimbot, jaimlib, jake2, jaligner, jalopy, jameleon, jamm, jamod, jamon, jamos, japs, japt-proxy, jarp, jasmin, jason, jasperreports, jasperserver, jastor, jatha, jato, java-jml, java3dsloader, javabdd, javacpc, javacurses, javadc3, javahmo, javahtmlparser, javaisdoomed, javalogging, javamail-crypto, javamath, javapathfinder, javaswf, javavis, javawebparts, javax-usb, jaxb-builder, jaxe, jaxme, jaxodraw, jaxor, jazzy, jbarcodebean, jbcheckstyle, jbilling, jboss-opentool, jcache, jcae, jcaif, jcctray, jchatirc, jchempaint, jchessboard, jclasslib, jcomm, jcommander, jconfig, jcr-webexplorer, jcrbrowser, jcrontab, jcryptool, jcsc, jct, jcv, jdai, jdbcmanager, jdbforms, jdbm, jdee, jdesigner, jdip, jdivelog, jdochelper, jduplicate, jedit, jedit-syntax, jena, jenia4faces, jeocaching, jep, jeplite, jeppers, jester, jetrix, jeuclid, jext, jfcunit, jfern, jffmpeg, jfig, jfilecrypt, jfilesync, jfin, jfire, jflac, jflex, jfreechart, jfuzzylogic,

jgap, jgen, jgenea, jggapi, jgloss, jgnash, jgossipforum, jgrapht, jgrib, jgrinder, jguard, jhotdraw, jical,
jikes, jikesrvm, jipe, jiprof, jitterbit, jkaiui, jkiwi, jlibrary, jlogic, jmailsrv, jmakeztxt, jmanage, jmath-
tools, jmatlink, jmax, jmdns, jmemorize, jmk, jmlspecs, jmol, jmoney, jmp3renamer, jmri, jmt, jmusic,
jmxplorer, jnetstream, jnlp, jobscheduler, joda-time, jode, joesnmp, jooda, joone, joost, jorgan, jortho,
josgui, josso, jox, jpackit, jpcap, jped, jpen, jpetrinet, jpf, jpicedt, jpilotexam, jpivot, jpl, jpodder, jpos,
jposloader, jppf-project, jprogect, jpublish, jpws, jpydbg, jquantum, jrat, jrdf, jreepad, jrefactory, jrf,
jrobin, jsap, jscheme, jsci, jscicalc, jseditor, jsettlers, jsf-spring, jsh, jskat, jslp, jsmsirl, json-lib, json-
taglib, jspeex, jstock, jsxe, jsyncmanager, jsynoptic, jtcfrost, jtds, jteg, jtestcase, jtidy, jtreeview, jtri,
jttslite, juddi, juggleanim, jugglinglab, juk, jump, jump-pilot, junitbook, junitdoclet, junitee, jupload,
juploadr, jvftp, jvi, jwamtoolconstr, jwap, jwbf, jwebmail, jwebunit, jwic, jwma, jwordnet, jworksheet,
jxquick, jxtaim, jydt, jzjkit, kafenio, kaon, karto, kasai, keepassj2me, kneobase, knowtator, kobjects,
kontor, kowari, krysalis, ksoap2, ktable, kurzfiler, kw-cdt, kxml, lazy8ledger, lectcomm, lemur, lgl,
libusbjava, lily4jedit, liquibase, liquidlnf, llamachat, llrp-toolkit, loadsim, locallucene, loggingsele-
nium, lopica, loro, lpdspooler, lpg, lsid, ltsa, lucene, lumbermill, lunar-eclipse, lunareclipse, lusid,
luxor-xul, mactor, maexplorer, magiccollection, majix, mandarax, mantaray, mapletree, marf, mars-
sim, martyr, matheclipse, mathlib, matrex, mav, maven-plugins, maxent, mc4j, mdr, mecat, mediachest,
medialibrary, meetingpoint, memoranda, meta-extractor, metis-rs, metrics, mfradio, mged, mgo, mh-
ptester, microemulator, microlog, middlegen, midishare, midp-calc, millebv, mime-util, mindraider,
mm8leveleditor, mmbox, mmssuite, moagg, mobilekaraoke, mobilenews, mobilezx, mobup, mock-
maker, mockobjects, modelj, mogwai, moreunit, morph, mov, mp3elf, mpeg7audioenc, mpegparser,
mrpostman, muffin, mule, mx4j, mxeclipse, mycore, mydoggy, myster, mytelly, mytourbook, nake-
dobjects, nanovm, napkinlaf, nekohtml, netsitemais, nettool, networkagent, neuroscholar, neurosdbm,
newsml-toolkit, nfcchat, nice, nitro-nitf, nlpfarm, nmrshiftdb, nntprss, notmac, nsuml, numberrace,
nxtcommand, obe, objectlabkit, obpm, observation, octv, oggcarton, ogre4j, ohioedge, ojax, olap4j,
omegat, omnigene, one-jar, ooimlib, ooweb, open-chord, open-dis, openbasemovil, openbravopos,
opencyc, openharmonise, openhms, openi, openjean, openjms, openjnlp, openkm, openmailarchiva,
openmed, openmi, opennlp, openorb, openp2m, openproj, openrods, openrpg, openshore, open-
sourcecrm, opensubsystems, opensvgviewer, openuss, openwfe, openxava, openxml4j, opproject,
opsi, opt4j, optalgtoolkit, orderlycalls, oreka, osdldbt, osmius, osrmt, osseo, outliner, oval, ovanttasks,
owasp, owlapi, owlvision, owx, oxerp, oxyus, ozone, p-unit, p6spy, palooca, pamguard, pandoras-
jar, paperharbour, paros, pauker, pavlov, pbeans, pdfbox, pdfdoclet, pdfsam, pdune, personalblog,
phex, phosphor, php-java-bridge, phpwebedit, pi4soa, piccolo, picturemetadata, pipe2, piqle, pixory,
pkb, pklite, planeta, planetgenesis, plog4u, pmd, poesia, pojosoft-lms, pokersource, polepos, pollo,
pooka, posterita, ppgp, praya, prefuse, processdash, project-x, properjavardp, proxool, pscs, pushlets,
qalab, qform, qftp, qtitools, quantum, raccoon, rachota, radical, ratool, rcfaces, rcosjava, readmaniac,
recoder, red-piranha, redmin-mylyncon, rem1, rendezvous, reprap, reteppdf, retroweaver, rivernorth,
rmock, robocode, roborescue, rodin-b-sharp, roller, romaframework, routeruler, rptools, rsslibj, rssowl,
rssview, rubyeclipse, runawfe, rvsn00p, sahi, salto-db, salto-framework, sannotations, sax, sblim, sbw,
schemeway, scope, scrabbledict, scrinch, sdljava, sdm, securityfilter, seda, sequalite, serverwatcher,
seven-mock, sharptools, shastahub, shelled, shoddybattle, shop, sisc, skunkdav, smallsql, smartqvt,
smartweb, smc, smoothmetal, smstools, snap, soapui, soccer, softsqueeze, solitairecatan, sonogram,
sourcejammer, sourcetapcrm, sparql, sparta-xml, spindle, sportstracker, spring-beandoc, springlay-
out, sql2java, sqladmin, sqldeveloper, sqltools, sqlunit, squirrel-sql, sslext, starpound, statcvs, statsvn,
stoplicht, storybook2, storytestiq, strangebrew, stringtree, strutsgenerator, strutstestcase, studianalyse,
stxx, sunshade, sunxacml, supercsv, superwaba, sweethome3d, swingosc, swingset, swixat, sword-
app, swt-composer, swtbot, swtfox, symmetricds, sync4j, syncdocs, synclast, systray, ta-lib, tableview,
tacos, tagtraum-jo, taylor, tcljava, telnetd, terppaint, tersus, thing, thingamablog, thinwire, thout, thun-
dergraph, tikal, timecult, timedoctor, timeslottracker, timetrack, timmon, tinysql, tinyvm, tipcon1, tjdo,
tjger, tn5250j, tockit, tolven, torqueide, toscanaj, trackit, trackplus, transferware, transmogrify, travian-
shell, treebeard, treeform, treemap, triplea, triptracker, tudomais, tudu, turquaz, tuxguitar, twinkle,
txtfl, tyrex, u-p2p, ubermq, uced, uddi4j, uengine, uic, uilib-oai, uitags, umldot, umleditor, unbbayes,
unicore, upnp-portmapper, vainstall, valuelist, vassalengine, vcb, veditor, versiontree, veryquickwiki,
vigilog, vijava, villonanny, visualstruts, vnc-tight, vssplugin, vtd-xml, wapmon, watij, wbemser-
vices, web-cat, webadmin, webcamstudio, webcompmath, webcurator, webdav-servlet, webforum,

webmacro, webman-cms, webonswing, webswell, webunitproj, wfmopen, wh2fo, wicketwebbeans, wife, wiki-flcelloguy, wiki2xhtml, wintvcap-gui, wisupdecode, wodka, wonder, woped, wordfreak, ws4d-javame, wsdl4j, wsmostudio, wurfl, wvtool, wx4j, x4l-reload, xbrlapi, xbrowser, xcoder, xdoclet, xdoclet-plugins, xebece, xebra, xena, xerlin, xflow, xframe, xgql, xholon, xilize, xinco, xinity, xins, xml2java, xmldb-org, xmlpipedb, xmm, xmoon, xmsf, xnap, xnap-commons, xnapster, xpairtise, xparam, xpetstore, xpusp, xradar, xslt-process, xtf, xui, xweb, yajp, yale, yawiki, z390, zk1, zkdesktop, zkforge, zkstudio, zmpp, zocalo, zplet, zss, zvtm

## A.2 List of Mature Projects

adempiere, adito, antrunner, apelon-dts, argunet, arianne, barbecue, barcode4j, basicportal, bluemusic, bpmspace, buttress, centraview, cewolf, cgupnpjava, checkstyle, cmpcs, cobertura, codecover, cofax, colladarefinery, controlremote, csvjdbc, cuecreator, daffodildb, dbfit, dbunit, dddsample, dimensionex, dom4j, domingo, dozer, dragmath, drjava, druid, dspace, dynamicjasper, e-p-i-c, ebookme, ebxmlrr, eclipse-rbe, eclipsesql, eclipsewiki, ehcache, eimp, ejb3unit, ejbca, emma, eressea, evetrader, exist, exo, fedora-commons, findbugs, fitnesse, fluid, fourever, freedom-erp, freelims, freemarker, freemind, freetts, galleon, gate, gems, genj, gestdb, gfd, gistoolkit, gjtapi, glassbox, gmod, grinder, groimp, hammurapi, hipergate, hl7api, hsqldb, htmlparser, htmlunit, ireport, iseries-toolkit, iteraplan, itracker, j2meunit, j2s, jabref, jackcess, jacob-project, jaffa, jag, jameleon, japs, jarp, jason, jasperreports, jasperserver, javacpc, javahmo, javaisdoomed, javax-usb, jaxe, jbilling, jboss-opentool, jchempaint, jconfig, jcrontab, jdbforms, jdee, jdesigner, jdip, jedit, jena, jenia4faces, jext, jfcunit, jfreechart, jgen, jgnash, jgossipforum, jgrapht, jikes, jikesrvm, jiprof, jlibrary, jmanage, jmathtools, jmdns, jmol, jmri, jmt, jmusic, jobscheduler, joda-time, jode, joone, josso, jpcap, jpivot, jpodder, jpos, jrefactory, jrobin, jsf-spring, jsyncmanager, jtcfrost, jtds, jtidy, jtreeview, jugglinglab, juk, junitbook, junitdoclet, junitee, jupload, juploadr, jwebunit, kasai, kobjects, kontor, krysalis, lemur, llrp-toolkit, lpg, mandarax, mantaray, mars-sim, mav, mc4j, medialibrary, meta-extractor, metrics, microemulator, middlegen, midishare, millebv, mindraider, mrpostman, muffin, mx4j, mytourbook, nakedobjects, nanovm, napkinlaf, nettool, neuroscholar, neurosdbm, nfcchat, notmac, omegat, opencyc, openi, openorb, openproj, openrpg, opensourcecrm, openuss, openwfe, openxava, opproject, opsi, ozone, p6spy, palooca, pauker, pdfdoclet, pdune, phex, php-java-bridge, pmd, pokersource, processdash, proxool, qform, qftp, quantum, rachota, robocode, romaframework, rssowl, rssview, runawfe, sahi, sblim, scope, seda, shoddybattle, sisc, smstools, softsqueeze, sql2java, squirrel-sql, starpound, storybook2, strutsgenerator, strutstestcase, sweethome3d, swtbot, symmetricds, sync4j, taylor, tcljava, tersus, thout, trackit, trackplus, triplea, tudu, twinkle, tyrex, uddi4j, uengine, uitags, unicore, vainstall, valuelist, vassalengine, versiontree, veryquickwiki, villonanny, vnc-tight, vtd-xml, watij, wbemservices, webadmin, webmacro, webman-cms, wfmopen, wife, wiki2xhtml, wonder, woped, wsdl4j, xbrowser, xdoclet, xdoclet-plugins, xins, xmm, xpetstore, xui, yale, zk1

# Bibliography

[1] *8th IEEE International Software Metrics Symposium (METRICS 2002), 4-7 June 2002, Ottawa, Canada*. IEEE Computer Society, 2002.

[2] *9th IEEE International Software Metrics Symposium (METRICS 2003), 3-5 September 2003, Sydney, Australia*. IEEE Computer Society, 2003.

[3] *11th IEEE International Symposium on Software Metrics (METRICS 2005), 19-22 September 2005, Como Italy*. IEEE Computer Society, 2005.

[4] W. Abdelmoez, M. Shereshevsky, R. Gunnalan, H.H. Ammar, Bo Yu, S. Bogazzi, M. Korkmaz, and A. Mili. Quantifying software architectures: an analysis of change propagation probabilities. In *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on*, pages 124–, 2005.

[5] Walid Abdelmoez, Diaa Eldin M. Nassar, Mark Shereshevsky, Nicholay Gradetsky, Rajesh Gunnalan, Hany H. Ammar, Bo Yu, and Ali Mili. Error propagation in software architectures. In *IEEE METRICS*, pages 384–393. IEEE Computer Society, 2004.

[6] C. Alexander. *The timeless way of building*. Oxford University Press, USA, 1979.

[7] Alberto Avritzer and Elaine J. Weyuker. Investigating metrics for architectural assessment. In *IEEE METRICS*, pages 4–10. IEEE Computer Society, 1998.

[8] Victor R. Basili. Software modeling and measurement: the goal/question/metric paradigm. Technical report, University of Maryland at College Park, College Park, MD, USA, 1992.

[9] V.R. Basili, L.C. Briand, and W.L. Melo. A validation of object-oriented design metrics as quality indicators. *Software Engineering, IEEE Transactions on*, 22(10):751–761, Oct 1996.

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

[11] Benoit Baudry, Yves Le Traon, and Gerson Sunyé. Testability analysis of a uml class diagram. In *IEEE METRICS* [1], pages 54–.

[12] Benoit Baudry, Yves Le Traon, Gerson Sunyé, and Jean-Marc Jézéquel. Measuring and improving design patterns testability. In *IEEE METRICS* [2], pages 50–.

[13] Karl Beecher, Cornelia Boldyreff, Andrea Capiluppi, and Stephen 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.

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

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

[16] Silvia Breu, Thomas Zimmermann, and Christian Lindig. Mining eclipse for cross-cutting concerns. In Diehl et al. [23], pages 94–97.

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

[18] P. Clements, R. Kazman, and M. Klein. *Evaluating software architectures: methods and case studies*. Addison-Wesley Professional, 2002.

[19] P. Clements, R. Kazman, M. Klein, D. Devesh, S. Reddy, and P. Verma. The duties, skills, and knowledge of software architects. In *Software Architecture, 2007. WICSA '07. The Working IEEE/IFIP Conference on*, pages 20–23, Jan. 2007.

[20] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion. For Subversion 1.4*. Red-Bean online version. Compiled from r2866, 2009.

[21] Giulio Concas, Michele Marchesi, Sandro Pinna, and Nicola Serra. Power-laws in a large object-oriented software system. *IEEE Transactions on Software Engineering*, 33(10):687–708, 2007.

[22] Eduardo Santana de Almeida, Alexandre Alvaro, Vinicius Cardoso Garcia, Leandro Marques Nascimento, Silvio Romero de Lemos Meira, and Daniel Lucrédio. Designing domain-specific software architecture (dssa): Towards a new approach. In Gorton et al. [30], page 30.

[23] Stephan Diehl, Harald Gall, and Ahmed E. Hassan, editors. *Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006, Shanghai, China, May 22-23, 2006*. ACM, 2006.

[24] L. Dobrica and E. Niemela. A survey on software architecture analysis methods. *IEEE Transactions on software Engineering*, 28(7):638–653, 2002.

[25] William M. Evanco. Architectural tradeoffs at the object. In *IEEE METRICS* [1], pages 43–53.

[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-wesley Reading, MA, 1995.

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

[28] Simon Giesecke, Johannes Bornhold, and Wilhelm Hasselbring. Middleware-induced architectural style modelling for architecture exploration. In Gorton et al. [30], page 21.

[29] G.K. Gill and C.F. Kemerer. Cyclomatic complexity density and software maintenance productivity. *Software Engineering, IEEE Transactions on*, 17(12):1284–1288, Dec 1991.

[30] Ian Gorton, Jeff Tyree, and Dilip Soni, editors. *Sixth Working IEEE / IFIP Conference on Software Architecture (WICSA 2007), 6-9 January 2005, Mumbai, Maharashtra, India*. IEEE Computer Society, 2007.

[31] Ahmed E. Hassan, Michele Lanza, and Michael W. Godfrey, editors. *Fith International Workshop on Mining Software Repositories, MSR 2008 (ICSE Workshop), Leipzig, Germany, May 10-11, 2008, Proceedings*. ACM, 2008.

[32] Israel Herraiz, Jesús M. González-Barahona, and Gregorio Robles. Determinism and evolution. In Hassan et al. [31], pages 1–10.

[33] ISO/IEC. *Software engineering – Product quality – Part 1: Quality model*, 2001. ISO/IEC 9126-1:2001.

[34] ISO/IEC. *Information technology – Process assessment – Part 1: Concepts and vocabulary*, 2004. ISO/IEC 15504-1:2004.

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

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

[37] Rick Kazman, Mark Klein, and Paul Clements. Atam: Method for architecture evaluation. Technical report, CMU/SEI, August 2000.

[38] Barbara Kitchenham and Shari Lawrence Pfleeger. Software quality: The elusive target. *IEEE Software*, pages 12–21, January 1996.

[39] Jingyue Li, Reidar Conradi, Odd Petter N. Slyngstad, Christian Bunse, Muhammad Umair Ahmed Khan, Marco Torchiano, and Maurizio Morisio. Validation of new theses on off-the-shelf component based development. In *IEEE METRICS* [3], page 26.

[40] Bernard W. Lindgren. *Statistcal Theory*. McMillan, New York, 3 edition, 1976.

[41] T. J. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[42] John Moses. A consideration of the impact of interactions with module effects on the direct measurement of subjective software attributes. In *IEEE METRICS*, pages 112–123. IEEE Computer Society, 2001.

[43] Taiga Nakamura and Victor R. Basili. Metrics of software architecture changes based on structural distance. In *IEEE METRICS* [3], page 8.

[44] I. Shaik, Walid Abdelmoez, Rajesh Gunnalan, Mark Shereshevsky, A. Zeid, Hany H. Ammar, Ali Mili, and Christopher P. Fuhrman. Change propagation for assessing design quality of software architectures. In *WICSA*, pages 205–208. IEEE Computer Society, 2005.

[45] M. Shepperd. A critique of cyclomatic complexity as a software metric. *Software Engineering Journal*, 3(2):30–36, Mar 1988.

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

[47] CMMI Product Team. CMMI for Development, Version 1.2. Technical Report CMU/SEI-2006-TR-008, Software Engineering Institute, Carnegie Mellon University, 2006.

[48] André van der Hoek, Ebru Dincel, and Nenad Medvidovic. Using service utilization metrics to assess the structure of product line architectures. In *IEEE METRICS* [2], pages 298–308.

[49] Michel Wermelinger and Yijun Yu. Analyzing the evolution of eclipse plugins. In Hassan et al. [31], pages 133–136.

[50] Elaine J. Weyuker. Predicting project risk from architecture reviews. In *IEEE METRICS*, pages 82–90. IEEE Computer Society, 1999.

[51] Kenny 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`.

[52] Yaojin Yang and Claudio Riva. Scenarios for mining the software architecture evolution. In Diehl et al. [23], pages 10–13.

[53] Hongyu Zhang, Hee Beng Kuan Tan, and Michele Marchesi. The distribution of program sizes and its implications: An eclipse case study. *CoRR*, abs/0905.2288, 2009.