

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/330700405>

Software Architecture Metrics: a literature review

Preprint · January 2019

CITATIONS

0

READS

4,697

4 authors, including:



Fabio Petrillo

University of Québec in Chicoutimi

80 PUBLICATIONS 431 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



DR-Tools Suite [View project](#)



Software Processes for Video Games Development [View project](#)

SOFTWARE ARCHITECTURE METRICS: A LITERATURE REVIEW

A PREPRINT

Théo Coulin¹, Maxence Detante¹, William Mouchère¹, Fabio Petrillo²
 Département de génie informatique et génie logiciel
 Polytechnique Montréal¹
 Montreal, QC, Canada
 {theo.coulin,maxence.detante,william.mouchere}@polymtl.ca

Département de Mathématique et Informatique
 Université du Québec à Chicoutimi²
 Chicoutimi, QC, Canada
 fabio@petrillo.com

January 29, 2019

ABSTRACT

In Software Engineering, **early detection of architectural issues is key. It helps mitigate the risk of poor performance, and lowers the cost of repairing these issues.** Metrics give a quick overview of the project which helps designers with the detection of flaws or degradation in their architecture. **Even though studies unveiled architectural metrics more than 25 years ago, they have not yet been embraced by the industry nor the open source community.** In this study, we aim at conducting a review of existing metrics focused on the software architecture for evaluating quality, early in the design flow and throughout the project's lifetime. We also give guidelines of their usage and study their relevance in different contexts.

1 Introduction

Building a system in the software engineering field has proven to be a difficult task, especially in large projects. The need to think ahead and to reflect on what the system could look like before starting to implement it is vital. Therefore, the importance of software architecture is critical in any software engineering project. This is the reason why we wanted to explore the quantitative measurements that could be driven on software architectures to check if it has good properties.

The quality of a software's architecture is essential, yet very difficult to apprehend and measure. The quality features of an architecture are not obvious as relations and dependencies can extend very far away. Moreover the complexity and the amount of relations also often make it difficult to read. Evaluating an architecture by hand is a tedious task. This is why we think it is very interesting to have metrics that sum up and quantify the features of an architecture. They help the designer because they hint the flaws to look for.

Having such metrics would make architecture checking much faster and less costly. This would make it possible for designers to run checks from the start of the project [1] and throughout the life of the software. For example, in a scrum context, the architecture could be evaluated at each sprint to make sure it's not drifting to become something impossible to maintain. It would also allow for comparison between architectures to pick the one that fits best to the project's requirements.

The goal of this paper is to provide a systematic review of architecture metrics and to link them with some of the quality features that they represent. Most metrics are not focused on one quality feature, but by combining them, one can end up with a quantitative measure of the quality features.

To reach our goal we will focus on the two following research questions :

- *What are the metrics available for a software designer to evaluate the quality of a software architecture?*
- *To what extent are these metrics representative of an architecture's quality?*

The contributions of this study are twofold. First we will build a reference list of the existing metrics applicable on software architecture. Then we will give guidelines for the usage of these metrics and show what quality features they express and how relevant they are.

The rest of the paper is structured as follows. First, in section 2, we are going to shortly present the context of software engineering and give definitions for the words we will often use in the paper. Then, we will talk about some related studies in section 3. In section 4, we will explain our study design and the methodology we followed. Section section 5 will outline the publication trends of the papers we gathered. Section 6 will present the metrics we identified during our review. For each metric we will give guidelines for usage and check whether the metric can be considered relevant or not. Threats to the validity of our work are considered in section 7. Finally we will close the study by summing up the conclusions and thinking about the future work ahead in section 8.

2 Background

2.1 Architecture Qualities

When designing a software, there are many different ways to come up with the solution to the original problem. In fact, the solution is often found through trial-and-error because in software engineering problems are wicked. The solution to the first problem might unveil something new that will affect your original solution. If it is possible to end up with multiple different design, it is also very likely that only a few of these design are quality design. This is where metrics might first be useful, to quickly determine if a solution is of sufficient quality to be used. Another problem is that the quality of a design can not be quantitatively expressed, as it is a qualitative estimation of a combination of factors whose importance depends on the project's requirements. Here is a list of the quality features of a software architecture that we found during our review :

- **Maintainability** [2, 3, 4, 5] : It is very important that a software is easy to maintain. It must be easy to make changes in a software, either for the addition of a new feature, or for a bug fix. This means that changes must be possible locally, without impacting all of the software. **Maintainability also means that it is possible for multiple developers to work on separate parts of the software without impacting each other**, which enables parallel work. Maintainability is often synonym with sustainability, as it makes it possible for the software project to be used and updated for many years. For all these reasons, making a software maintainable is key for the future of the project.
- **Extensibility** [5] : The **ability for a software architecture to handle addition of new functionalities and components**. It is very valuable in agile development as **features are added throughout the life of the project**.
- **Simplicity, Understandability** [6, 7, 8, 2] : Making a software architecture as simple as possible is key to making it most understandable for everyone. Another way to represent this is the complexity of the architecture. Of course the software architecture will always be complex, but the goal is to make it as simple as possible. A simpler architecture is more maintainable, easier to communicate about and possibly more reliable.
- **Re-usability** [9] : In the industry, it is often very valuable to re-use previous projects to make development and design quicker. Making a software architecture re-usable is good to save on future projects.
- **Performance** [10, 11] : Having a metric capable of estimating performance early in the design process is very interesting as performance is often an issue in software architecture while it is also often a part of the requirements of the client. This would help with choosing the best architecture for performance before even starting to develop anything, as no implementation trick can compensate for the poor performances of a bad design.

2.2 Relevance

We will try to estimate the **relevance** of the metrics we show in this paper. The term relevance has a rather large meaning, so we will try here to explain what criteria we used to assess it. We are aware of the subjective aspect of such a definition, but we tried to highlight what we believe to be key components in the quality of a metric :

- **Relation with architectural qualities** : The metric has to depict efficiently one or more quality features of a software architecture. It seems obvious, but the metric has to provide insight on a given aspect of the architecture.

- **Theoretical validation** : If the metric is complex and uses mathematical concepts, such as statistics for example, its validity has to be demonstrated theoretically.
- **Empirical validation** : The more the metric has been tested in real conditions, the better. A metric, even crafted with the greatest care and proven theoretically, needs to be tested to see if it really provides meaningful information on the architecture.
- **Ease of utilization** : a metric, or a given set of metrics, must to a certain extent be easy to compute. As taking too much time on the architectural phase of a project is prohibitive in business contexts, the time that is necessary to compute metrics has to be reduced to the minimum.
- **Understandability** : A metric should be easy to understand by anyone, as it can be a tool for presenting the advantages and drawbacks of an architecture in a convincing way.

3 Related Work

One of the first papers on the subject [12], wrote in 1994 by Chidamber et al., proposed a first set of metrics to apply on a software architecture. Since then, many combinations of their metrics have been studied to produce more relevant indicators. The focus has also been on making possible automatic evaluation of UML designs because a software architecture is not trivial to translate into something understandable by a computer.

We found three Systematic Literature Reviews which deal with software architecture quality metrics. The first review [13] found about 40 metrics distributed over 11 *metrics-based* evaluation methods. But this review includes in its definition of architecture the system implementation. This means that not all the metrics found are relevant in an *architecture-only* context, which makes it a broader review than ours. Moreover, the paper focuses on the sustainability of the software rather than on the quality in general, which makes it look on a different angle than we do. Given that it dates back from 2011 we think that there might have been new metrics or interpretations in the literature.

The second review [14], which is focused on the *architecture-only* context, found 25 metrics. It builds a mapping of these metrics in the literature, and uses their frequency of appearance to evaluate their maturity. Some metrics we found will overlap with these two other reviews, but we think that we may find different or even new metrics ([14] dates back to 2015). Moreover, our study puts an incentive on the usage of these metrics to improve the quality of an architecture when the others were focused on other aspects.

The third review [15] is a very recent review on software architecture metrics. It focuses on giving software designers tools to evaluate large architectures for industrial applications. They end up building three major categories of metrics : *Architecture Measures*, *Design Stability* and *Technical Debt*. Even though they share some common base material with us, their study is different from ours in that they don't show the metrics that they do not endorse as relevant. We, on the counterpart of that, will showcase those as metrics to be tested, as they could lead to an improvement of what is available.

4 Study Design

We conducted this review applying a methodology and following guidelines from [16] and [17]. We put a lot of effort in making it transparent and reproducible. We will explain in the following section the methodology we followed.

4.1 Research Questions

Our goal is to answer the following research questions:

- *What are the metrics available for a software designer to evaluate the quality of a software architecture?*
- *To what extent are these metrics representative of an architecture's quality?*

4.2 Information gathering and interest checking

As a first approach to find content in the literature we used Google to find some papers about metrics, as using the databases wasn't very effective in the first place. This was our way to get a few papers to verify there was an interest in software architecture metrics and to help us identify the scope of our work. While doing this, we found two SLR, [13, 14] and [18, 12] which are founding papers for the subject.

4.3 Tools

For this review, we used the website parsif.al to keep a trace of everything we did regarding the selection of the papers. These tools helped us having an overview of our paper set and made the selection process easier. We used Zotero to export our bibliography. We also used a lot of shared spreadsheets to divide our work.

4.4 Search and selection process

We designed our search and selection process following steps as described in [16, 17]. It helped us having a broad scope of research but efficiently sorting among the amount of papers available. The steps we followed are depicted in Figure 1.

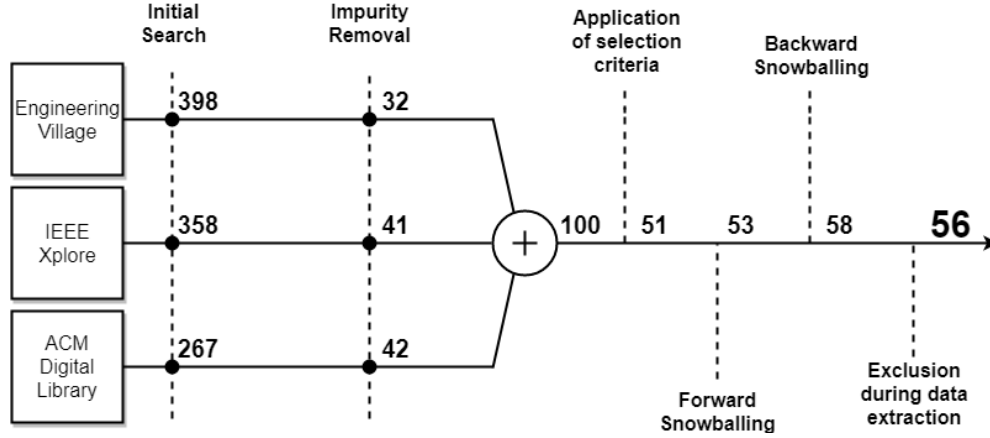


Figure 1: Papers selection methodology and results

Initial search First, we built search queries to dig in known databases, such as IEEE, ACM and Engineering Village. We selected these databases because they have a very good reputation, and they are very large and complete which makes this review very exhaustive. They also are very well supported by parsif.al and Zotero which made the process of exporting easier.

Our queries were the following :

- IEEE : ("software architecture" OR "software architectures") AND metric* AND (evaluation OR quality)
- ACM : ("software architecture" OR "software architectures" +metrics +metric evaluation quality)
- Engineering Village : (("software architecture" OR "SA") AND metric* AND (qualit* or eval* or pertinenc* or impact or relevanc*) wn TI)

We applied this query on all metadata for IEEE and ACM, but only on the papers titles for Engineering Village.

Impurity removal Third, we performed an impurity removal per database. Given the content of our research, there are many side papers that match the query but are not relevant, ie. papers on code metrics or papers on network architecture. We removed all of these papers to make a coherent set of papers around our subject.

Merging and duplicates removal Fourth, we merged the result of each database into a single set and removed the duplicates.

Application of the selection criteria Fifth, we performed a three reviewers voting on our dataset to choose the papers that were the most meaningful for our work. Our selection criteria were the following :

- S1 - Are there any metrics proposed to evaluate the quality of an architecture ?
- S2 - Is the relevance of the metric assessed ?

As shown in Picture 2, we split our dataset in three and assigned two reviewers for each sub-dataset. They went over every paper deciding whether or not it matched our selection criteria. If both reviewers are positive, the paper is in, if

both are negative, the paper is out. Finally, if the reviewers did not agree, the third reviewer chose whether to keep or not the paper.

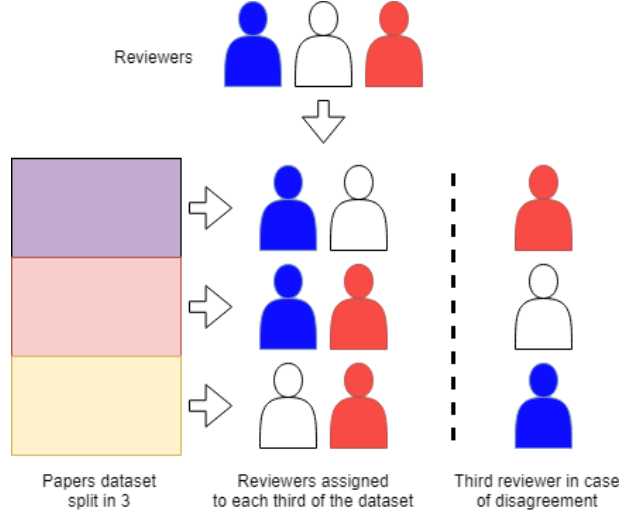


Figure 2: Three reviewers voting for application of the selection criteria

Forward and backward snowballing Sixth, we did a forward snowballing using the papers that matches the inclusion criteria, in order to discover more meaningful articles. Seventh, we applied a backward snowballing to make sure that we did not miss any important and founding paper. We performed the snowballing with the following policy:

- First we selected only papers which had a title that seemed interesting;
- We checked if the paper matched our selection criteria;
- We only kept the paper if it was very interesting to us, as we already had a significant database of papers.

This means that our snowballing is not exhaustive, but rather focused on adding significant knowledge to our already big set of papers.

Exclusion during data extraction While performing our data extraction, there were some papers that we believed were not as relevant as expected. This is why we chose to remove those papers from our final set.

4.5 Data extraction

The papers that we gathered contain different metrics that we tried to summarize and explain. We also checked whether or not they were relevant. For every paper, we focused on extracting a few key components that appeared to be of high importance to us:

- Which architectural quality is treated. The extracted data can be a subset of *{Maintainability, Extensibility, Simplicity and Understandability, Re-usability, Performance}*;
- The specific domain or topic associated with the paper. A good example of a domain is Service Oriented Design, but it can also cover a broad topic like change propagation or performance;
- The fact that the metric(s) proposed has been theoretically and/or experimentally validated by the researchers.

After the data extraction, we synthesized our findings in different tables in order to have an overview of the metrics and then to categorize them by looking at what aspect they were measuring. It allowed us to think about the structure of our paper.

5 Publication trends

We thought that it was interesting to examine a bit the publication trends in our papers dataset, especially the publication years. Figure 3 shows the distribution of publication years in our dataset.

Table 1: Underlying architecture measurement

Metric	#Papers	Papers
Coupling	28/56	[21, 22, 3, 23, 24, 25, 6, 1, 26, 27, 8, 28, 29, 30, 31, 32, 33, 15, 5, 2, 20, 34, 14, 35, 13, 36, 37, 38, 39]
Cohesion	19/56	[21, 22, 4, 24, 1, 27, 28, 29, 30, 31, 32, 15, 5, 14, 35, 13, 40, 39, 37]
Complexity	18/56	[24, 6, 1, 23, 27, 8, 30, 7, 41, 32, 19, 15, 5, 14, 35, 13, 37, 39]
Size	14/56	[22, 24, 6, 42, 43, 44, 31, 19, 15, 5, 14, 35, 13, 36]

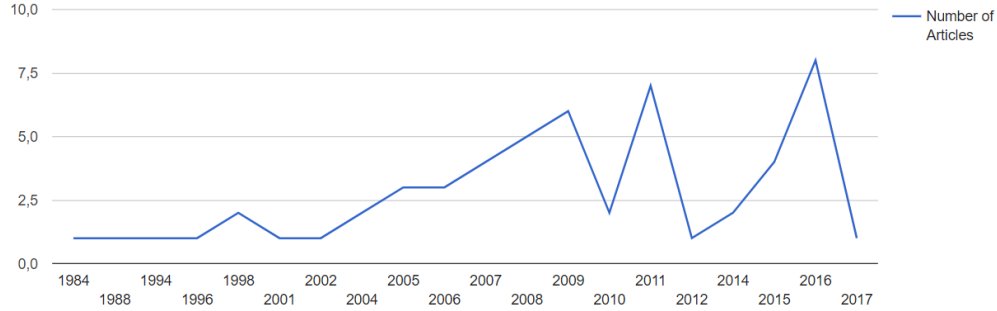


Figure 3: Distribution of publication years in our dataset

We can see that software architecture metrics have been a concern for quite some time, as the first paper dates back to 1984. Since then, the scientific interest in this subject has grown a lot to become very trendy in the last couple of years. The industrial interest is surely growing as more and more articles about the subject show a cooperation with the industry [19, 2, 20].

6 Results

Our research showed that there are groups of metrics which derive from the same underlying measurement. Often papers make little changes in the measurement in order to tweak it and come up with something more meaningful. Sometimes they also make combinations of these measurements. We decided to group the metrics we found by the underlying measurement to which they belong. Then we will present metrics that we found that do not enter in these categories. Finally we will focus on metrics applied to Service Oriented Architecture as they differ a bit from Object Oriented metrics but are very relevant in today's industrial context. As shown by Table 1 there are many papers which deal with coupling, cohesion, complexity and size of the software's architecture. We can see a focus on coupling and cohesion, as it is known that it is very important to keep low coupling and high cohesion in an architecture.

We also sorted the papers by the quality features they tackle to show what is focused by the literature. This is summed up in Table 2. We believe that the quality feature the most talked about are very probably features that can now be measured with a metric reliably. This means that these feature are ready to be evaluated automatically on large projects. In the Table 2 we can also see that maintainability is the main focus, probably because it is the most important quality features, as all others depend on it.

Table 2: Quality features tackled by the papers

Feature	#Papers	Papers
Maintainability	31/56	[4, 3, 27, 44, 45, 29, 46, 30, 31, 47, 48, 32, 19, 49, 33, 15, 5, 50, 51, 52, 2, 20, 34, 14, 35, 13, 40, 53, 36, 38, 39]
Performance	6/56	[54, 55, 11, 46, 10, 14]
Extensibility	6/56	[29, 32, 5, 14, 13, 36]
Simplicity	9/56	[24, 44, 46, 15, 5, 51, 14, 13, 36]
Re-usability	12/56	[24, 9, 29, 31, 47, 48, 32, 15, 5, 50, 20, 34, 14]

6.1 The goal/question/metric paradigm

First, we want to introduce the work of Basili et al. [56]. This work is fundamental, as it proposes a framework for creating new metrics. It is based on the goal/question/metric paradigm, which is used in a considerable amount of papers we found.

The aim of this method is to derive metrics that will help assess the quality of a given parameter. In order to be sure that the metric is well-related to the parameter, the framework proposes a three-levels systematic procedure to finally derive software metrics : the goal level, questions level and ultimately metrics level. The contribution of this method is to propose templates for setting goals, and guidelines for deriving quantifiable questions and, finally, metrics.

On the first level, one has to set project goals, which are split into different subcategories : *purpose*, *perspective*, and *environment*.

On the second level, questions must be derived from the goals set. There is also three categories here : *definition of the product*, where questions are related to quality attributes; *quality perspectives of interest*; and *feedback*.

On the last level, deriving metrics from the questions and goals mentioned above, a few guidelines are provided. The difference between subjective and objective metrics is emphasized, where objective metrics are seen as countable characteristics of a product (complexity, for example), and subjective metrics are related to aspects that cannot be characterized objectively. In that case, a ordinal scale may be used to categorize and create a metric.

6.2 Coupling

On an architecture level, the metric which is the most used is coupling, and its complementary, cohesion. The first occurrence of coupling as a metric we found was in Chidamber et al. [12]. It has then been derived into many versions which sharpen the precision of the metric. Generally speaking, what comes from these papers is that it is important to have a high cohesion in modules, and a low coupling throughout the architecture. Low coupling is very important because it diminishes the risk of ripple effect when making changes in the program. Thus, low coupling is very important to keep the architecture maintainable. Maintainability is the most important quality that is displayed by a low coupling.

In [28], Shereshevsky presents a metric for coupling that is applicable on different levels of architecture design. Either at early steps of the design or at code level. They are based on data and information flows. It has not been validated in the paper but seems promising.

In [33], Choi extends the static coupling defined by Chidamber to dynamic coupling which is a measurement at object level. The goal is to be even more accurate. It has been compared with static coupling in the paper and seems to give more result, but at the cost of simplicity of use.

Another very interesting approach is that of Mo et al. in [2]. Instead of measuring coupling, they decided to measure the decoupling, that is the modularity of the architecture. A good modularity means easy maintenance and re-usability. Their metric has been validated in the paper and seems very up to date and usable.

6.3 Complexity

Another very important metric is complexity. It affects the understandability of the architecture and possibly the performance. It has also been first defined by Chidamber et al. in [12].

It can be expressed by the number of classes in the architecture, or the number of links between classes in the architecture. In [7], the author actually makes a mix of these two values. He calculates the dependence graph of the architecture and uses common graph calculus to assess the complexity of the architecture. This metric hasn't been verified yet.

In [6], the author uses Full Function Point, a functional size measure, to calculate the complexity of a software architecture. There is an example of application in the paper, but it has not been widely tested.

6.4 Change and error Propagation

Change Propagation evaluates the maintainability of an architecture based on the probability that a change in a class will have an impact on other classes. Shereshevsky et al. [28] proposed a set of metrics for software architectures in 2001, and introduced change propagation and requirements propagation between components. The difference between the two is that change propagation can help measuring the cost of corrective maintenance, whereas requirements propagation deals with adaption to a change in requirements.

In 2005, Abdelmoez et al. [47] dedicated a whole paper to change propagation metrics, in component-based architectures. They proposed to build a matrix of change propagation probabilities between components, in order for the architect to be able, at a glance, to assess the difficulty and cost of maintenance operations. Moreover, it can highlight key components that have a high probability to change, or that imply changing other components when they change. Those quick insights allows architects to take counter-measures easily, for example making a component easily adaptable.

Abdelmoez et al. also worked on metrics related to error propagation in [53]. It is also based on component-based architecture, introducing metrics to compute the error propagation in a system, using a stochastic approach. They define the error propagation probability between two components, A and B. If there exists a connector between the aforementioned components, we can compute the probability that an error occurring in A is transmitted to B, rather than masked by B.

They also consider the fact that even if an error occurs in A, there is no guarantee that A will transmit a message to B. Therefore, they introduce a transmission probability matrix to assess if an error in A will effectively be transmitted to B, considering not only the probability of the error occurring in A but also the probability of A transmitting a message to B.

Those metrics seem pretty solid on the theoretical part. Moreover, the authors developed a framework to empirically test their metrics by using a fault injection tool. Also, they rigorously compare those results with the analytic ones they got from computing the metrics with only architecture-level information. One can use those metrics to assess if a bug in a given component can impact other component, and therefore gain insight on the future maintainability of a system.

6.5 Design Pattern Density

We found a very promising metric, proposed by Dirk Riehle in [57], which measures the percentage of class in the architecture that are part of a design pattern. It helps the designer to evaluate the maturity of an architecture. The more mature an architecture is, the more design pattern are put into it, and the higher the design pattern density. It is very good when applied on frameworks, which should be very densely filled with design patterns. A framework with high pattern density is more understandable and likely more performing. This metric seems to be quite hard to use as it does not express on a fixed scale the maturity of the design, but rather on a scale which depends on the problem the software deals with. It has been tested with success on open source frameworks, but it has not been broadly validated yet and would benefit from being. We think that this different metric might be a very good way to express maintainability and understandability of the design.

6.6 Performance Evaluation through architecture

We also found a few papers related to performance, more particularly Software performance Engineering (SPE), and how to assess the possible performance of a system from its architecture only. It builds on the hypothesis that architecture

plays a critical role in the performance of a system. Liu et al. [54] describe their method to assess the performance of a component-based and container-hosted solution. It requires prior modeling of several critical part of the future system, such as the tool that receives all requests and redirects them to the corresponding service, or the database activity. Therefore, it requires additional efforts, but seem to provide an accurate profile of the platform performance through response time prediction. Distefano et al. [11] use UML diagrams and the *OMG Profile for Schedulability, Performance, and Time Specification*. From this point, they construct a performance model, from which they extract several metrics such as utilization and throughput.

The drawbacks of these methods are the same: the cost of implementing them to compute metrics and solve the models seems to be high and the process time-consuming. Moreover, since performance evaluation is highly correlated with the physical characteristics of the system, the measurements depend a lot on the architect capability to estimate correctly several performance-related factors. Thus, there is a need to have a good knowledge of the performance field to construct the performance model, which will be solved to derive performance metrics. to assess the compliance of the architecture with performance requirements.

Gheeta et al. [55] try to implement a lightweight approach to performance evaluation, by only considering *Use Cases* diagrams, and annotating them with performance goals such as a time, or a size, to respect. Then, from technical and environmental factors, and from the complexity of use cases and actors, they estimate a representative workload. From the diagrams made, they generate a performance model, which is then solved to show response times of different components of the system.

6.7 Modularization

In [36] Sarkar et al. define a set of metrics that is suited to perform an evaluation of large projects' modularity. A project that is very modular is easier to maintain in the long term and can easily be extended. The metrics defined by the author are made to encourage the usage of proper APIs between the modules of the project. Those APIs can be APIs providing service, or extension APIs. The later define a taxonomy of the functionality to provide for the plugins that will extend the module. Their set includes :

- metrics to measure the coupling between modules;
- metrics to count the number of inter-module calls that are not made through the defined API;
- metrics to detect inheritance between classes of different modules;
- metrics to assess that the higher level of abstraction is the one used by classes outside the module (Liskov Substitution Principle);
- metrics to assess that interfaces are actually holding a single responsibility.

Though dating back from 2008, this metrics set is very modern as it checks some of the SOLID principles. All the metrics have been very thoroughly validated in the paper and are very well described.

6.8 UML diagrams evaluation

Li et al. [44] proposed three metrics to evaluate different aspects of UML diagrams. The first metric proposed is *Information Content* (IC). It is based on a hierarchy and weight of the different elements in UML diagrams. It defines the quantity of information a diagram or the architecture passes. The higher the IC, the higher the amount of information delivered.

The second is more interesting as it is original, and helps assessing the quality of the architecture in terms of understandability. It is called *Visual Effect*. The higher the visual effect, the more complex it is for a human being to comprehend the diagram at a glance.

The third metric is close to being a coupling metric : the *Connectivity degree* measures the number of associations w.r.t. the numbers of entities in the diagram. Different types of associations have different weights.

According to their authors, these metrics can be used to assess the scale, complexity and stability of a given architecture, but these metrics have not been validated experimentally (except a case study).

6.9 Methodologies for architectural evaluation in agile environment

We found two papers dealing with controlling software architecture through the whole development process, in the context of agile methodologies.

Agile methodologies have the particularity to work in iterations. At the start of each iteration, some requirements are chosen by the development team and the client. They become the objectives for the iteration. Those aspects are implemented, and at the end of the sprint, the product is presented to the client. Then, a reflection on the iteration is made to highlights ways to improve, and the process goes on with a new iteration. The advantages of such methodologies are that the product is more resilient in the event of changing requirements. But these methodologies also put emphasis on not having too much documentation.

The problem that can arise is to have a robust software architecture in those conditions. We found two papers that dealt with this paradox of having to keep a robust architecture, while using a methodology that is heavily feature-oriented : Ahuja et al. [43] and Christensen et al.[45].

The work of Christensen et al. [45] is very interesting : it starts with the finding that most of the techniques used to assess the quality of software architectures are heavyweight and costly to perform, thus not useful in an agile context. They developed the *architectural Software Quality Assurance* technique (aSQA) to provide a lightweight technique, whose goal is to assess the quality of software architecture as well as prioritizing things to work on. It also enables to balance quality attributes of the architecture. Also, this technique needs to have a component based, or service-based architecture to be easy to use.

The most important part of the technique lies in the beginning of its implementation. At the start of each project, every stakeholder has to agree on the qualities the product should have. This allows to derive architecture qualities. The next step is to define metrics to quantify the quality attributes of the architecture.

The next step is to define a mapping of quality measurements to aSQA levels. Since the aSQA technique puts emphasis on balance and prioritization between qualities, it is necessary to use the same scale for all metrics. Therefore, they decided to use ordinal values between 1 and 5 to allow comparison between quality attributes. 1 defines an unacceptable level of the quality attribute, 3 an acceptable one, and 5 an excellent level of quality.

From those metrics, a target level and the current level of the quality attributes are assessed. It enables to compute the health of the quality attribute, and along with its importance (defined by the stakeholders), we can derive the focus to put on this particular quality.

The advantage of this technique is also that it has been tested as the reference technique in a danish company for multiple projects, and also in two other companies. Therefore, its validity is acceptable.

We can conclude that using metrics to assess the quality of software architecture can be interesting even when using processes that do not normally put an emphasis on architecture, as it allows to balance qualities of the product, and also to prioritize and focus attributes for the next iteration.

6.10 Service Oriented Architecture

A designer working on a Service Oriented Architecture (SOA) could think that using Object Oriented (OO) metrics is a good way to evaluate his design. However, it as been proven that OO metrics are particularly irrelevant for assessing the quality of a SO design, as concepts of classes, methods or objects are totally nonexistent in SOA[24]. New metrics have to be derived from the OO ones to adapt their measures to the SOA concepts, like procedures for instance. This is the reason why we decided to dedicate a whole sub-section to the SOA oriented metrics that we reviewed during our work.

Considering Web services in particular, Qian et al. proposed in 2006 a (de)coupling metric[9] for evaluation in service composition of service oriented components. They proposed four metrics giving assumptions of how their evolution impact different quality attributes such as maintainability and understandability. However, no validation of the metrics are provided.

Pereplechikov et al. introduced cohesion metrics especially for SOA in [4]. They define some characteristics and identify types of cohesion with a ranking to determine which one is stronger than the other. They present a set of five metrics, evaluating the different types of coupling with a sixth one for assessing the total cohesiveness of the software. Their work is particularly interesting as the metrics are calculated at design time and are thus technology independent. On the other hand, there is no empirical validation of their set in this paper, but they plan it as future work.

In a second work the same year, the same authors created a second set of metrics for evaluating the coupling of a SO design[3]. They made eight assumptions about different types of coupling to help them define a set of metrics covering these assumptions. They introduce eight primary metrics and six aggregation metrics based on computations on the primary ones. Note that we decided to consider only their purely static metrics as the other are dynamic metrics, evaluated during run-time, which is out of our scope. Despite not having a real empirical validation, they used the *property-based software engineering measurement framework* referenced in their paper to validate the presented metrics. As their previous work, those are design-time metrics, allowing for maintainability prediction from high level of

abstraction in the architectural model. The importance of their metrics is not yet established and they leave it for future work.

Finally, the year after, Pereplechikov et al. conducted a similar study[38] aimed at the validation of the SOA coupling metrics that they introduced in their previous work[3], through the study of the impact of coupling on the maintainability of an architecture. The paper is organized like the one we previously presented ([40]). The experimental protocol used for the validation is very similar to the first one, and we suppose that the participants of this study are the same. The results show that there is a need of more empirical validation, with more case studies and participants, as the weights of the metrics are not well established. In addition, the validation has shown that some types of coupling (e.g. intraservice coupling) have a lesser impact on maintainability than other ones (e.g. indirect and direct, incoming and outgoing, extraservice coupling). Sadly, they do not propose the same guidelines as in their previous work, but this study is a very interesting complement of the early effort they put in coupling metrics [3].

Hofmeister and Wirtz introduced a complexity[8] metric from the coupling point of view. They define six base measures, then three metrics for measuring the coupling, based on the previous ones and finally three metrics for measuring the complexity. For each of them, they give explanation on how they computed it and discuss the exact signification of what is measured. Before testing their model on a case study, they state that their metrics are relevant only if they give a bad rating of a software. In this case, a redesign could be considered, but if the metrics state that there is no need for a redesign, it does not mean that the model is optimal. In addition they do not consider the categorization of a design between "easy to modify" and "hard to modify" feasible. That's why they only introduce these metrics and specify that there is a need of studying their behavior on more "enterprise-scale case studies".

In 2008, Shim et al. came with a very robust paper[24] dealing with SOA metrics in general. First, all of their metrics are evaluated at design time, before any implementation work begin. Second, they extend QMOOD, a hierarchical quality assessment model for object-oriented design quality evaluation, to adapt it to SOA evaluation. The resulting model is a four-level meta-model with 1) High-Level Design Quality Attributes, which were also defined by them, 2) SOA Design Properties, 3) SOA Design Metrics and 4) SOA Design Components. In addition they clearly explain the mappings between each level in the model. Moreover, they define twenty-one basic metrics and seven derived metrics, computed on the previous ones and they establish relationships between these derived metrics and the SOA Design Properties. Finally, they link the properties with the High Quality Attributes by assessing the impact of each property on the quality attributes. At the end of their study, they validate their model by an empirical study on two versions of a project, in order to value the quality of the changes. This work is a real milestone in the SOA architecture metrics field. We think that any SO architect should have a look at their model if he wants to evaluate his design and truly understand the results of the analysis.

Ma et al. proposed an interesting study [21] on evaluating the identification of SOA services. As this identification is one of the most important tasks in defining an SOA, it could be a good thing to have a way to quantitatively measure the quality of an identified services portfolio, in order to further be able to compare multiple set of services and eventually choose the best one. Again, as this study focus on service identification, it focuses on the design level of the development process. As other SOA studies, they first define a set of four specific SO metrics to fulfill the specific needs of such a design. Each metric evaluate respectively : Service granularity, Service coupling, Service cohesion and Business entity convergence. The proposed model is conducted in three steps: 1) Modeling, identify services in the business process and model the structure of the identified portfolio; 2) Measuring, use the model to measure quality features of services in the portfolio with the help of corresponding design metrics; 3) Evaluation : overall evaluation of the services set by normalizing the metrics and adding some weights, which any designer can customize to adapt the model to his needs. Finally, the model is validated by a case study on five different portfolios of identified services for a same problem. The evaluation revealed that the model seems quite accurate as the most balanced portfolio was selected as the best in this case.

Sindhgatta et al. defined a rather large set of metrics[29] for evaluating many different quality aspect of a SOA, namely service cohesion, coupling, re-usability, composability and granularity. Their study is particularly interesting because they put a real effort in evaluating and measuring the validity of their metrics, for each category, through two industry projects. In addition, these metrics are calculated and based on the design level only, allowing for early detection of design flaws. This paper is a very good way for an architecture designer to get into SOA evaluation as the authors present many different quality aspect, study the correlations and links between their metrics and finally confront their theory to two real case study, giving range of values for each metric which helps for comparison.

A second paper that deals with service granularity as been proposed by Alahmari et al.[39] in 2011. In this study, they propose a framework based on eight metrics. In addition of presenting each metric, they also explain what they exactly evaluate and proceed on a theoretical validation using Briand et al. framework (widely used in the metrics sector). They end their paper with a test of their model on a case study. This test show that "the granularity of service operations always affects complexity, cohesion and coupling" but the exact extent to what it affects those aspects is not

yet established. We see two main issues with this paper : first, there is not real empirical validation of this set of metrics, and second, the metrics are based on the code syntax, thus it is difficult to really use them to make decision during the design process. However, they can be useful in a refactoring process for flaws identification.

7 Threats to Validity

External validity.

Even though we selected the papers for this literature review with the greatest care, there might be articles that are biased, or for which the experimentation process is not rigorous, as the metrics are tested on a few projects only. This might lead to some metrics that are misleading or not that relevant.

Internal validity. There is a great threat on this particular aspect. Since we conducted this review in less than three months, and worked on it as a project while attending multiple courses, there is a threat on all levels of our analysis. From the methodology to the results, the lack of proper time might induce some flaws in the design or in the thought process. For instance, during the forward and backward snowballing, we selected the papers mainly with their title, which are sometimes misleading. Consequently, we may have missed important publications, as we dug down only if the title seemed relevant. Nevertheless, we tried to follow the best practices in SLR, as described in [16, 17].

Also, by focusing on the architecture quality, we are missing a lot of quality properties of a software product. We are also letting many of these properties to the implementation phase and are exposed to deviation from the original architectural design. However, we deliberately reduced our scope to the architecture level as we wanted to focus on the design flaws rather than the implementation flaws. Less than a real threat, it is something that one have to have in mind when considering evaluating his software following the guidelines and metrics we reviewed in this paper.

8 Conclusions and Future work

We conducted a Systematic Literature Review to find which metrics were available to assess the quality of an architecture, early in the design process and throughout the software's lifetime. It has been known for a while that architecture is a key component in software development. Therefore, we listed a wide range of metrics, sorting them by the quality aspect of architecture they tackle. We aimed at giving architects an overview of the tools available, in order for them to have a better understanding and another viewpoint when choosing between different architectures during the early stages of a project. In addition, the cost-effectiveness of metrics allows designers to quickly and quantitatively evaluate their architecture, leaving an open window to comparison. This is a real improvement considering the slow and qualitative evaluation that can be given by expert only, which can not really be integrated in a fast development process like scrum for instance.

Future work may focus on assessing the robustness of the metrics mentioned in this paper. For example, trying to get more experimental data to confront the validity of the metrics might be interesting, as they are not always tested on real projects. This could help the adoption of those metrics by the industry, as they seem to be not that relevant in their point of view[26] for now.

Most evaluation tools, like SonarQube¹ do not implement any metrics that are design based. These tools put the incentive on the codebase and we believe that it would be beneficial to add metrics that are at design-level rather than at code-level.

Finally, an interesting subject to put effort in is to use some of those metrics and try to employ them together in order to detect other flaws, like security weakness or components identification[26]

References

- [1] M. Galster, A. Eberlein, and M. Moussavi, "Early assessment of software architecture qualities," in *2008 Second International Conference on Research Challenges in Information Science*, Jun. 2008, pp. 81–86.
- [2] R. Mo, Y. Cai, R. Kazman, L. Xiao, and Q. Feng, "Decoupling Level: A New Metric for Architectural Maintenance Complexity," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 499–510. [Online]. Available: <http://doi.acm.org/10.1145/2884781.2884825>

¹<http://docs.sonarqube.org/display/SONAR/Metric+definitions>

- [3] M. Pereplechikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling Metrics for Predicting Maintainability in Service-Oriented Designs," in *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*, Apr. 2007, pp. 329–340.
- [4] M. Pereplechikov, C. Ryan, and K. Frampton, "Cohesion Metrics for Predicting Maintainability of Service-Oriented Software," in *Seventh International Conference on Quality Software (QSIC 2007)*, Oct. 2007, pp. 328–335.
- [5] U. Dayanandan and K. Vivekanandan, "An Empirical Evaluation Model for Software Architecture Maintainability for Object Oriented Design," in *Proceedings of the International Conference on Informatics and Analytics*, ser. ICIA-16. New York, NY, USA: ACM, 2016, pp. 98:1–98:4. [Online]. Available: <http://doi.acm.org/10.1145/2980258.2980459>
- [6] M. AlSharif, W. P. Bond, and T. Al-Otaiby, "Assessing the Complexity of Software Architecture," in *Proceedings of the 42Nd Annual Southeast Regional Conference*, ser. ACM-SE 42. New York, NY, USA: ACM, 2004, pp. 98–103. [Online]. Available: <http://doi.acm.org/10.1145/986537.986562>
- [7] J. Zhao, "On Assessing the Complexity of Software Architectures," in *Proceedings of the Third International Workshop on Software Architecture*, ser. ISAW '98. New York, NY, USA: ACM, 1998, pp. 163–166. [Online]. Available: <http://doi.acm.org/10.1145/288408.288450>
- [8] H. Hofmeister and G. Wirtz, "Supporting Service-Oriented Design with Metrics," in *2008 12th International IEEE Enterprise Distributed Object Computing Conference*, Sep. 2008, pp. 191–200.
- [9] K. Qian, J. Liu, and F. Tsui, "Decoupling Metrics for Services Composition," in *5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COM SAR'06)*, Jul. 2006, pp. 44–47.
- [10] L. G. Williams and C. U. Smith, "Performance Evaluation of Software Architectures," in *Proceedings of the 1st International Workshop on Software and Performance*, ser. WOSP '98. New York, NY, USA: ACM, 1998, pp. 164–177. [Online]. Available: <http://doi.acm.org/10.1145/287318.287353>
- [11] S. Distefano, M. Scarpa, and A. Puliafito, "From UML to Petri Nets: The PCM-Based Methodology," *IEEE Transactions on Software Engineering*, vol. 37, no. 1, pp. 65–79, Jan. 2011.
- [12] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994.
- [13] H. Koziolok, "Sustainability evaluation of software architectures," in *in: ACM Sigsoft Int'l Conf. on the Quality of Softw. Architectures*, 2011, pp. 3–12.
- [14] S. Stevanetic and U. Zdun, "Software Metrics for Measuring the Understandability of Architectural Structures: A Systematic Mapping Study," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '15. New York, NY, USA: ACM, 2015, pp. 21:1–21:14. [Online]. Available: <http://doi.acm.org/10.1145/2745802.2745822>
- [15] M. Staron and W. Meding, "A portfolio of internal quality metrics for software architects," in *Lecture Notes in Business Information Processing*, vol. 269, Vienna, Austria, 2017, pp. 57 – 69. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-49421-0_5
- [16] B. Kitchenham and S. Charters, *Guidelines for performing Systematic Literature Reviews in Software Engineering*, 2007.
- [17] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Information and Software Technology*, vol. 55, no. 12, pp. 2049–2075, Dec. 2013. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0950584913001560>
- [18] S. R. Chidamber and C. F. Kemerer, "Towards a Metrics Suite for Object Oriented Design," in *Conference Proceedings on Object-oriented Programming Systems, Languages, and Applications*, ser. OOPSLA '91. New York, NY, USA: ACM, 1991, pp. 197–211. [Online]. Available: <http://doi.acm.org/10.1145/117954.117970>
- [19] H. Koziolok, D. Domis, T. Goldschmidt, P. Vorst, and R. J. Weiss, "MORPHOSIS: A Lightweight Method Facilitating Sustainable Software Architectures," in *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, Aug. 2012, pp. 253–257.
- [20] S. Roubtsov, A. Serebrenik, and M. D. Van Brand, "Dn-based design quality comparison of industrial java applications," in *2009 5th Central and Eastern European Software Engineering Conference in Russia, CEE-SECR 2009*, Moscow, Russia, 2009, pp. 95 – 101. [Online]. Available: <http://dx.doi.org/10.1109/CEE-SECR.2009.5501182>

- [21] Q. Ma, N. Zhou, Y. Zhu, and H. Wang, “Evaluating Service Identification with Design Metrics on Business Process Decomposition,” in *2009 IEEE International Conference on Services Computing*, Sep. 2009, pp. 160–167.
- [22] S. Peldszus, G. Kulcsár, M. Lochau, and S. Schulze, “Continuous Detection of Design Flaws in Evolving Object-oriented Programs Using Incremental Multi-pattern Matching,” in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2016. New York, NY, USA: ACM, 2016, pp. 578–589. [Online]. Available: <http://doi.acm.org/10.1145/2970276.2970338>
- [23] P. Banerjee and A. Sarkar, “Quality Evaluation Framework for Component Based Software,” in *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, ser. ICTCS '16. New York, NY, USA: ACM, 2016, pp. 17:1–17:6. [Online]. Available: <http://doi.acm.org/10.1145/2905055.2905223>
- [24] B. Shim, S. Choue, S. Kim, and S. Park, “A design quality model for service-oriented architecture,” in *2008 15th Asia-Pacific Software Engineering Conference*, Piscataway, NJ, USA, 2008, pp. 403 – 10. [Online]. Available: <http://dx.doi.org/10.1109/APSEC.2008.32>
- [25] R. T. Tvedt, M. Lindvall, and P. Costa, “A process for software architecture evaluation using metrics,” in *27th Annual NASA Goddard/IEEE Software Engineering Workshop, 2002. Proceedings.*, Dec. 2002, pp. 191–196.
- [26] A. Nicolaescu, H. Lichter, and Y. Xu, “Evolution of Object Oriented Coupling Metrics: A Sampling of 25 Years of Research,” in *Proceedings of the Second International Workshop on Software Architecture and Metrics*, ser. SAM '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 48–54. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2821327.2821337>
- [27] D. M. Le, C. Carrillo, R. Capilla, and N. Medvidovic, “Relating Architectural Decay and Sustainability of Software Systems,” in *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Apr. 2016, pp. 178–181.
- [28] M. Shereshevsky, H. Ammari, N. Gradetsky, A. Mili, and H. H. Ammar, “Information theoretic metrics for software architectures,” in *25th Annual International Computer Software and Applications Conference. COMPSAC 2001*, 2001, pp. 151–157.
- [29] R. Sindhgatta, B. Sengupta, and K. Ponnalagu, “Measuring the Quality of Service Oriented Design,” in *Service-Oriented Computing. Proceedings 7th International Joint Conference, ICSOC-ServiceWave 2009*, Berlin, Germany, 2009, pp. 485 – 99. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10383-4_36
- [30] H. Venkitachalam, J. Richenhagen, A. Schlosser, and T. Tasky, “Metrics for Verification and Validation of Architecture in Powertrain Software Development,” in *Proceedings of the First International Workshop on Automotive Software Architecture*, ser. WASA '15. New York, NY, USA: ACM, 2015, pp. 27–33. [Online]. Available: <http://doi.acm.org/10.1145/2752489.2752496>
- [31] K. K. Chahal and H. Singh, “Metrics to Study Symptoms of Bad Software Designs,” *SIGSOFT Softw. Eng. Notes*, vol. 34, no. 1, pp. 1–4, Jan. 2009. [Online]. Available: <http://doi.acm.org/10.1145/1457516.1457522>
- [32] G. Zayaraz and P. Thambidurai, “Quantitative Measurement of Software Architectural Qualities through COSMIC FFP,” in *2006 Annual IEEE India Conference*, Sep. 2006, pp. 1–4.
- [33] M. Choi and J. Lee, “A Dynamic Coupling for Reusable and Efficient Software System,” in *5th ACIS International Conference on Software Engineering Research, Management Applications (SERA 2007)*, Aug. 2007, pp. 720–726.
- [34] A. Serebrenik, S. Roubtsov, and M. v. d. Brand, “Dn-based architecture assessment of Java Open Source software systems,” in *2009 IEEE 17th International Conference on Program Comprehension*, May 2009, pp. 198–207.
- [35] M. Alenezi, “Software Architecture Quality Measurement Stability and Understandability,” *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 7, no. 7, 2016.
- [36] S. Sarkar, A. C. Kak, and G. M. Rama, “Metrics for Measuring the Quality of Modularization of Large-Scale Object-Oriented Software,” *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 700–720, Sep. 2008.
- [37] V. R. Basili, L. C. Briand, and W. L. Melo, “A validation of object-oriented design metrics as quality indicators,” *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, Oct. 1996.
- [38] M. Pereplechikov and C. Ryan, “A Controlled Experiment for Evaluating the Impact of Coupling on the Maintainability of Service-Oriented Software,” *IEEE Transactions on Software Engineering*, vol. 37, no. 4, pp. 449–465, Jul. 2011.
- [39] S. Alahmari, E. Zaluska, and D. C. D. Roure, “A Metrics Framework for Evaluating SOA Service Granularity,” in *2011 IEEE International Conference on Services Computing*, Jul. 2011, pp. 512–519.

- [40] M. Pereplechikov, C. Ryan, and Z. Tari, “The impact of service cohesion on the analyzability of service-oriented software,” *IEEE Transactions on Services Computing*, vol. 3, no. 2, pp. 89 – 103, 2010. [Online]. Available: <http://dx.doi.org/10.1109/TSC.2010.23>
- [41] J. Beane, N. Giddings, and J. Silverman, “Quantifying Software Designs,” in *Proceedings of the 7th International Conference on Software Engineering*, ser. ICSE ’84. Piscataway, NJ, USA: IEEE Press, 1984, pp. 314–322. [Online]. Available: <http://dl.acm.org/citation.cfm?id=800054.801986>
- [42] E. Bouwers, J. P. Correia, A. v. Deursen, and J. Visser, “Quantifying the Analyzability of Software Architectures,” in *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, Jun. 2011, pp. 83–92.
- [43] C. Ahuja, P. Kaur, and H. Singh, “Quantitative evaluation of software architecture,” in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, Mar. 2016, pp. 1000–1006.
- [44] J. Li, Z. Guo, Y. Zhao, Z. Zhang, and R. Pang, “Towards quantitative evaluation of UML based software architecture,” in *2007 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, Piscataway, NJ, USA, 2007, pp. 663 – 9.
- [45] H. B. Christensen, K. M. Hansen, and B. Lindstrom, “Lightweight and continuous architectural software quality assurance using the aSQa technique,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6285 LNCS, 2010, pp. 118 – 132. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15114-9_11
- [46] O. Zimmermann, “Metrics for Architectural Synthesis and Evaluation – Requirements and Compilation by Viewpoint. An Industrial Experience Report,” in *2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics*, May 2015, pp. 8–14.
- [47] W. Abdelmoez, M. Shereshevsky, R. Gunalan, H. H. Ammar, B. Yu, S. Bogazzi, M. Korkmaz, and A. Mili, “Quantifying software architectures: an analysis of change propagation probabilities,” in *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, 2005., Jan. 2005, pp. 124–.
- [48] E. Bouwers, A. v. Deursen, and J. Visser, “Quantifying the Encapsulation of Implemented Software Architectures,” in *2014 IEEE International Conference on Software Maintenance and Evolution*, Sep. 2014, pp. 211–220.
- [49] D. Perez-Palacin, R. Mirandola, and J. Merseguer, “Software Architecture Adaptability Metrics for QoS-based Self-adaptation,” in *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*, ser. QoSA-ISARCS ’11. New York, NY, USA: ACM, 2011, pp. 171–176. [Online]. Available: <http://doi.acm.org/10.1145/2000259.2000288>
- [50] H. Changjun, J. Feng, and Z. Chongchong, “An architectural quality assessment for domain-specific software,” in *Proceedings - International Conference on Computer Science and Software Engineering, CSSE 2008*, vol. 2, Wuhan, Hubei, China, 2008, pp. 143 – 146. [Online]. Available: <http://dx.doi.org/10.1109/CSSE.2008.671>
- [51] F. A. Fontana, I. Pigazzini, R. Roveda, and M. Zanoni, “Automatic Detection of Instability Architectural Smells,” in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Oct. 2016, pp. 433–437.
- [52] I. Shaik, W. Abdelmoez, R. Gunalan, M. Shereshevsky, A. Zeid, H. H. Ammar, A. Mili, and C. Fuhrman, “Change Propagation for Assessing Design Quality of Software Architectures,” in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA’05)*, 2005, pp. 205–208.
- [53] W. Abdelmoez, D. M. Nassar, M. Shereshevsky, N. Gradetsky, R. Gunalan, H. H. Ammar, B. Yu, and A. Mili, “Error propagation in software architectures,” in *10th International Symposium on Software Metrics, 2004. Proceedings.*, Sep. 2004, pp. 384–393.
- [54] Y. Liu, I. Gorton, and A. Fekete, “Design-level performance prediction of component-based applications,” *IEEE Transactions on Software Engineering*, vol. 31, no. 11, pp. 928–941, Nov. 2005.
- [55] D. E. Geetha, T. S. Kumar, and K. R. Kanth, “Framework for Hybrid Performance Prediction Process Model: Use Case Performance Engineering Approach,” *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 3, pp. 1–15, May 2011. [Online]. Available: <http://doi.acm.org/10.1145/1968587.1968607>
- [56] V. R. Basili and H. D. Rombach, “The TAME project: towards improvement-oriented software environments,” *IEEE Transactions on Software Engineering*, vol. 14, no. 6, pp. 758–773, Jun. 1988.
- [57] D. Riehle, “Design Pattern Density Defined,” in *Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA ’09. New York, NY, USA: ACM, 2009, pp. 469–480. [Online]. Available: <http://doi.acm.org/10.1145/1640089.1640125>