Evaluating Usefulness of Software Metrics - an Industrial Experience Report -

Eric Bouwers*‡, Arie van Deursen‡ and Joost Visser*§

* Software Improvement Group, Amsterdam, The Netherlands
E-mail {e.bouwers,j.visser}@sig.eu
‡ Delft University of Technology, Delft, The Netherlands
E-mail {Arie.vanDeursen,E.M.Bouwers}@tudelft.nl
§ Radboud University, Nijmegen, The Netherlands

***Abstract*—A wide range of software metrics targeting various abstraction levels and quality attributes have been proposed by the research community. For many of these metrics the evaluation consists of verifying the mathematical properties of the metric, investigating the behavior of the metric for a number of open-source systems or comparing the value of the metric against other metrics quantifying related quality attributes.**

Unfortunately, a structural analysis of the usefulness of metrics in a real-world evaluation setting is often missing. Such an evaluation is important to understand the situations in which a metric can be applied, to identify areas of possible improvements, to explore general problems detected by the metrics and to define generally applicable solution strategies.

In this paper we execute such an analysis for two architecture level metrics, Component Balance and Dependency Profiles, by analyzing the challenges involved in applying these metrics in an industrial setting. In addition, we explore the usefulness of the metrics by conducting semi-structured interviews with experienced assessors. We document the lessons learned both for the application of these specific metrics, as well as for the method of evaluating metrics in practice.

I. Introduction

Software metrics continue to be of interest for researchers and practitioners. Metrics such as volume [3], McCabe Complexity [26], the C&K metric suite [12] and a wide range of architecture metrics (see Koziolek [23] for an overview) are well-known and used in practice. Moreover, new software metrics continue to be defined by the research community.

The evaluation of a new metric typically consists of correlating the (change in) value of the metric with other quality indicators such as likelihood of change [25] or its ability to predict the presence of bugs [28]. In other cases, the evaluation consists of an analysis of the values of a metric for a set of open-source systems, either on one single snapshot or over a period of time [30], [29]. More theoretical approaches of metric evaluation inspect mathematical properties of metrics (see, for example, Briand et al. [11], [10] and Fenton et al. [15]) or focus on metrological properties of metrics (see, for example, Abran [1]).

The focus of these types of evaluation is to determine whether the designed metric is related to the quality property it has been designed to quantify, a property known as "construct validity" [21]. Although this is an important part of the evaluation of a metric, these types of evaluations cannot be used to determine whether a metric is *useful*. For a metric to be considered useful its value should correspond to the intuition of an measurer [15] and should be actively used in a decision-making process [16].

In this paper we evaluate the usefulness of two architecture level metrics, Component Balance [7] and Dependency Profiles [8], which are designed to quantify the analyzability and encapsulation within a software system. Evidence of the construct validity of these metrics has been previously gathered in small-scale experiments [7], [8]. The large-scale study presented here aims to gain an understanding of the usefulness of these two metrics in practice.

The context of this research is the Software Improvement Group (SIG), a consultancy firm specialized in providing strategic advice to IT management based on technical findings. As a first step both metrics are embedded in the measurement model used to monitor and assess the technical quality of a large set (500+) of systems developed by (or for) clients of SIG. The metrics are interpreted by consultants working at SIG, who fulfill the role of external quality assessors.

Data about the usefulness of the metrics is collected using two different methods. First, data about the challenges involved in actually applying the metrics is collected by observing the quality assessors and documented in the form of memos. Secondly, semi-structured interviews are conducted with the quality assessors who interpreted the metrics when assessing their customers' software systems.

Our analysis of the collected data leads to two types of findings. First, we identify in which situations and under which conditions the metrics are useful. Second, we discover how to improve the metrics themselves and ways to apply them better.

In addition to reporting on the evaluation of these specific metrics in this particular context, we reflect upon a general method for evaluating software metrics in a practical setting. The challenges involved in designing and executing such a study are outlined and the generalizability of the results is discussed. We conclude that despite the inherent limitations of this type of studies, the execution of such a study is crucial for the proper evaluation of any software metric.

II. Evaluation Goal

The goal of this study is to gain an understanding of the usefulness of two software metrics. Before we can reason about the usefulness of a metric the term "usefulness" needs to be characterized. Different characteristics of usefulness are available, e.g. a metric is considered useful if the metric:

- corresponds to the intuition of the measurer [15]
- is actively used in a decision making process [16]

In the first definition, a crucial role is played by the person using the metric. Apart from the experience of the particular person, the role in which he/she uses the metric, e.g., a developer, a quality assessor inside a company or an external quality assessor, has a significant impact on the outcome of the evaluation. In the second definition, the context in which the metric is used, e.g., assessing the quality of a system, analyzing the properties of an architecture or assessing the performance of developers, has a large influence on the outcome of the evaluation.

The subjects of this evaluation are the Component Balance [7] and Dependency Profiles [8] metrics. In our previous work have evaluated the properties, correlations and statistical behavior of the metrics [7], [9], but not their usefulness.

Both of these metrics have been designed to quantify specific properties of the implemented architecture of a software system [7], [8], i.e., analyzability and encapsulation. Such a quality assessment can either be done by an internal assessor (e.g., working inside a single company) or an external assessor (employed by, for example, a consultancy firm). In this research, we choose the viewpoint of external assessors.

Thus we define our evaluation goal in the template from the GQM approach of Basili et al. [6] as follows:

> The objective of the study is to understand the usefulness of the "Component Balance" and "Dependency Profiles" metrics, from the point of view of external quality assessors, in the context of external assessments of implemented architectures.

III. Evaluation Method

To answer our research question we use the four step methodology as outlined in Figure 1. To start, the metrics are included in the standard operating procedure of the external assessors. Details about this embedding and the context of the external assessors are given in Section IV.

In line with recommendations about collecting data in qualitative research [2], data about the challenges involved in applying the metrics are gathered using two methods. First, an observer records real-word experiences of using the metrics in the form of memos. Secondly, interviews with the assessors are conducted to determine the perceived usefulness of the metric as seen by the assessors. Details about this data-gathering process are given in Section V.

Lastly, the data extracted by both methods is analyzed and condensed separately, the results of which are given in Section VI and Section VII. Based on this data, the most

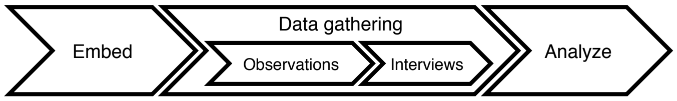

Fig. 1. A four-step process for evaluating software metrics in practice

common observations are discussed and possible solution areas are explored in Section VIII.

Apart from evaluating the metrics we also reflect upon the benefits and limitations of this evaluation process in Section IX, discuss related work in Section X, after which the paper concludes in Section XI.

IV. Evaluation Setting

The evaluation took place within the Software Improvement Group (SIG), a consultancy firm which "...translates detailed technical findings about software-intensive systems into actionable information for top management".[1]

The length of the investigation was six months, from the start of February 2012 until end of August 2012. At the start of this period, two system properties based on the Component Balance and the Dependency Profiles metrics were added to the company's software measurement model. Details about this measurement model and the embedding of the metrics are given in Section IV-A and Section IV-B.

The measurement model is applied by consultants employed by SIG on a wide range of customer systems throughout various services. Details about the consultants interpreting the metrics are discussed in Section IV-C, while the context of this research is described in two parts focussing on the services in which the metrics are used (Section IV-D) and the type of systems assessed in the services (Section IV-E).

A. Software Measurement Model

A measurement model based on the maintainability aspect of the ISO/IEC 9126 [19] standard for software quality was used throughout the services of SIG [18]. This model operationalizes the standard by decomposing its sub-characteristics into a set of six system properties, which are quantified by code-level metrics such as the duplication percentage and length of units.

These code-level metrics are in turn used to derive a rating for each system property using a benchmarking-based methodology [4]. More details about the exact construction of the model and its application can be found in Baggen et al. [5].

Since the introduction of this model, the ISO/IEC 9126 standard has been replaced by the new ISO/IEC 25010 [20] standard for software quality. One of the changes of the new standard is the introduction of the sub-characteristic of "Modularization", which is defined as:

> [The] degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

[1]http://www.sig.eu

To capture this new sub-characteristic, the measurement model was extended with two system properties: *Component Balance* and *Component Independence*. Apart from upgrading to the latest quality standard, the introduction of these system properties was expected to stimulate discussions about the architecture of systems and to incorporate a common viewpoint in the assessment of implemented architectures.

B. Component System Properties

The metric used to quantify the system property of Component Balance is the metric with the same name as was introduced in our previous work [7]. The metric used to quantify the Component Independence system property is the combination of two categories of the Dependency Profiles metric [8]. Both metrics were chosen based on the results of our earlier experiments, which showed that these metrics outperform other metrics when quantifying the quality characteristics of analyzability [7] and encapsulation [9].

Component Balance: The Component Balance metric consists of two factors, System Breakdown Optimality (SBO) and Component Size Uniformity (CSU). The SBO provides a value in the range of $[0,1]$ based on the number of components. The "optimal" number of components receives a high score which gradually decreases when a system contains a higher or a lower number of components. The "optimal" number of components is currently defined as the median number of components in a representative benchmark of systems.

The CSU metric provides a value in the range of $[0,1]$ based on whether the volume of the system is distributed (roughly) equally amongst the components of the system. More details about the design of these two metrics and their aggregation can be found in Bouwers et al. [7].

Component Independence: To quantify Component Independence two categories of the Dependency Profiles are used. A dependency profile categorizes all modules within a component into one of four categories based on the dependencies the module has on modules in other components, as illustrated in Figure 2. The percentage of code within the system in the *internal* category and the *outgoing* category is combined into a single percentage that quantifies the volume percentage of code in each component that is not called from or otherwise directly depended upon by code in another component. The higher this percentage the higher the rating for Component Independence.

Implementation choices: Similar to the existing system properties, the raw software metric is translated to a rating on a scale of $[0.5, 5.5]$ via a benchmarking based methodology [4], and these ratings are combined with the ratings for the other six system properties to calculate an overall rating [5].

To ensure that both metrics can be applied to systems that are written in multiple programming languages the following algorithms are used. For Component Balance, the volume of the components is measured as the sum of Lines of Code for all programming languages used within a component.

To calculate the categories of the Dependency Profiles, language-specific call- and hierarchy-dependencies are statically resolved to calculate a percentage per programming language. The rating obtained per programming language is aggregated to the system-level using a weighted average, taking into account the relative volume of each language.

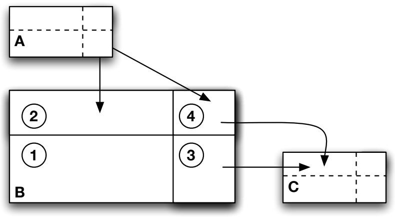

Fig. 2. The four categories of modules as defined within Dependency Profiles [8]. 1 = hidden, 2 = inbound, 3 = outbound and 4 = transit modules. Arrows denote dependencies from/to modules within other components.

C. Consultants

In this research we observe the consultants working for SIG. These consultants provide recommendations to clients to improve the quality of a system in order to mitigate risks, but are not involved in the execution of these recommendations. Thus they fulfill the role of external quality assessors.

Two different types of consultants are distinguished, *technical* consultants trained in identifying and assessing technical risks within a software system and *general* consultants responsible for translating technical findings into project- and business risks. By observing both types of consultants we aim to gain more insight in both the technical usefulness as well as the usefulness of the metrics on a business level.

D. Services

Four different services are offered by SIG. Two of them, the Software Risk Assessment (SRA) and the Software Risk Monitor (SRM), are the main subjects of this research. The goal of an SRA is to answer a specific research question related to risks involved in the technical quality of a software system. A standard investigation lasts 6-8 weeks and is executed by a team consisting of both general and technical consultants [14]. In an SRM, the identified risks and technical quality of a system are tracked over a period of time to ensure timely identification and mitigation of problems [24].

E. Software subjects

In the six-month study period the measurement model has been used to monitor the technical quality of over 500 systems, and applied to over 50 systems in the setting of an SRA. The size of the systems varies from three thousand to several million lines of code written in a wide range of programming languages ranging from Object-Oriented and related languages (e.g., Java, C#, JSP, ASP, JavaScript and various SQL-dialects) to languages typically deployed in mainframes (e.g., Cobol, Tandem). The systems originate from different domains including banking, insurance, government and logistics.

To gain the most benefits from transferring to the new model a system needs to have its component defined. These definitions were made based on information retrieved from the development teams using interviews and design documents.

V. Data Gathering

After the metrics were embedded into the measurement model, data about the application and usage of the metrics was gathered using two different methods:

1) Experiences of using the metrics were collected through observations and documented in the form of memos
2) The opinions of the external assessors about the usefulness of the metrics were collected by conducting semi-structured interviews

Combining these two methods of data-gathering does not only allow us to triangulate findings, but also reduces the known limitations of either method. A reflection on this design decision is given in Section IX.

A. Observations

The objective of using this method is to gain an understanding of the challenges involved in applying the metrics, to gain insight in the situations in which the metrics can be used and to identify situations in which the values of the metrics do not directly correspond to the intuition of the assessor.

During the six month period the first author, who works as a technical consultants at SIG, collects experiences about the metrics in the form of written memos. All questions and remarks about the metrics which are either publicly stated or directly asked are documented on a daily basis.

Each memo contains a description of the problem/question, the answer provided/action defined and possible follow-up actions. After the six month period all memos are manually analyzed to identify recurring questions and general observations.

B. Interviews

The objective of using this method is to get an overview of the usefulness of the metrics as perceived by the external quality assessors. Eleven software quality assessors with at least two years of experience with performing metric-based quality assessments were interviewed to construct this overview.

The interviews are time-boxed to a period of 30 minutes and are conducted by the second author of the paper. This author, who is not involved in the daily operations of SIG, has previous experience in using interviews as a basis for qualitative research [17] and started each interview with the following question:

> How do you use Component Balance and Component Independence?

The discussion based on this question is used to get a qualitative insight into the usefulness of the metrics. Each discussion is documented in a report which is validated by the interviewee.

In order to get a more quantitative insight into usefulness of the metrics each assessor was asked to answer the following two questions at the end of the interview:

1) On a scale of 1 to 5 (higher is better), how useful do you find Component Balance / Component Independence in your job?
2) On a scale of 1 to 5 (higher is better), does the use of Component Balance Component / Independence make it easier to do your job?

The above questions are based on the questions of Davis [13] and are designed to get an insight into the perceived usefulness and ease of use of the metrics.

VI. Observation Findings

Over the period of six months a total of 48 memos were collected. The memos describe interactions of the first author with 17 quality assessors (over one third of the available quality assessors), decomposed into 10 general consultants, six technical consultants and one internal researcher.

Twenty memos discuss specific systems, 14 different systems (spanning four different business contexts) where subject to these discussions. All memos combined involved 11 different customers and suppliers.

Note that even though the specific discussions only cover a fraction of all analyzed systems and clients, the remaining memos contain discussions about general trends observed by consultants, which are based on their experience with all systems as discussed in Section IV-E.

All memos were analyzed together at the end of the six months period. In this analysis 20 different concepts are extracted from the memos, which in turn are grouped into five different categories. Figure 3 shows an overview of the collected categories and the related concepts, each of which is discussed in the following sections.

A. Decision making

Indications for the actual use of the two architecture metrics in a decision making process was found in 16 different memos and grouped together in three different concepts.

Targeted Improvements: The addition of the metrics to the measurement model resulted in targeted improvement efforts being made by development teams, including the development team employed by the company. Shortly after the introduction of the architecture metrics one of the internal developers posted the following message on the internal communications-channel:

> In "eating-your-own-dogfood-news", the new component independence metric helped us find a remnant of old design in [system-name] that was subsequently refactored, resulting in a +0.1 maintainability and a +0.85 component independence

Start of discussions: Five memos describe direct questions posed by development teams on how to improve the rating for the metrics. In all cases, findings related to the specific system were discussed and recommendations were defined and communicated back to the development team.

Communication device: One memo describes a discussion with a development team of a system which is being monitored. At first the monitoring was focussed on specific technical issues which required a technical componentization. However, these technical components did not correspond to the components used by the developers to reason about the

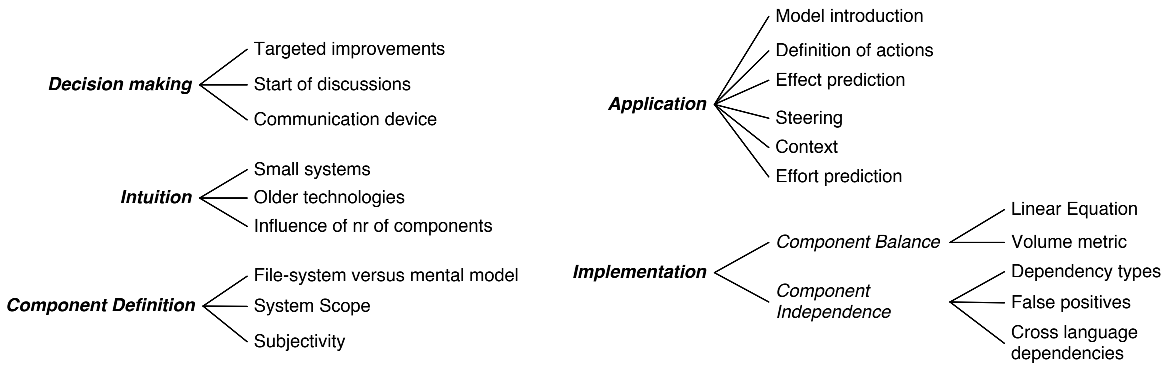

Fig. 3. An overview of the five categories (displayed in bold/italic font), two sub-categories (displayed in italic font) and their related concepts as collected via observations.

system. In order to decrease the effort needed to transfer the system to a new maintenance team the current development team decided to re-structure the source-code to reflect their mental components. In this situation, both architecture metrics were used to communicate the progress of this re-structuring to project-management.

In summary, the metrics formed a basis for a discussion about the components, led to the definition of a roadmap to make the transition of the system easier and provided a way to track the progress for non-technical personnel.

B. Intuition

As explained by Fenton et al. [15] a metric can be considered useful if it corresponds to the intuition of the person using the metric. On three occasions assessors specifically mention that in some cases the value of the metrics, in particular the Component Balance metric, does not immediately correspond to their intuition about the state of the system. One assessor states that in about half of the cases the ratings are as expected, while in the other half of the cases the definition of the components needs to be re-assessed.

More detailed examples of the situations in which the (change in) the value of the metrics does not correspond to the intuition of the assessors are described in seven different memos and grouped into three distinct categories.

Small systems: Four different memos (involving five different assessors) describe that smaller systems seem to receive lower ratings for the architecture metrics faster than larger systems. One of the assessors hypothesized that this is due to smaller systems with components related to technical topics (typically only a few such as database, front-end, services) because the size of the system does not require a functional decomposition.

For one assessor this size-related issue was important enough to sit down together with an internal researcher to inspect the distribution of the ratings for all eight system properties to determine whether the distribution of ratings was indeed different. The result of this inspection was that, apart from a relatively large spike caused by systems which did not yet have component definition, the distribution of ratings for architecture metrics was not different from that of the other metrics.

Older technologies: On two occasions assessors mentioned that systems written in older technologies (e.g., Cobol, Tandem, Pascal) seem to receive higher ratings for the component based metrics more easily than systems written in modern technologies (e.g., Java and C#).

For Component Balance, one assessor hypothesizes that this trend could be caused by the way in which components are solicited from the developers. Because the technologies themselves do not have a "component"-concept these types of systems normally do not have any components defined. During the transfer to the new quality model the sources are grouped together in components according to functionality *after* the metrics are explained, which could lead to a specific steering towards the "optimal" number of components and thus higher ratings.

Influence of number of components: For Component Balance one assessor observes that the number of components seems to influence the rating for Component Balance more than the size-distribution, which confirms the observations in the initial validation of this metric [7].

C. Component Definition

As discussed in Section IV, components needed to be defined for a system in order to gain the most from the transfer to the new model. Three concepts are related to this category.

File-system versus mental model: The components of a system were defined based on either the structure of the file-system (e.g., the top-level directories are used as components) or based on interviews with developers about how they view the system. In the latter case it might well be that files from different directories are combined into a single component.

To illustrate, one assessor outlined a case in which the system contained a top-level directory structure depicting technical components, while the second-level directories contained a functional decomposition. Depending on the view-point of the developers either the functional or the technical components can be used to calculate the metrics. However, it was unclear to the assessor which one of the two is the best representation of the "real" components of the system and should thus be used for the current assessment.

Because there may be different components under various view-points, the value of the rating can diverge, which in turn can have political consequences (for example if there exists a contractual agreement to reach a certain rating for each system property). This type of situation calls for a more clear definition of what constitutes a component.

On the other hand, one assessor stated that within a SRA setting it can be helpful to use different component-definitions (representing different views on the system) to determine risks with respect to different view-points.

Subjectivity: The lack of a very precise guideline of what constitutes a component is a reason to view a measurement based on components as subjective in two memos. In particular, by involving the developer in the definition of the components there exists a feeling that the value of the measurement can be easily influenced by using a different definition of component instead of a change in the code.

System Scope: The question described in one memo was whether libraries developed inside the company (but in this case only included as a binary dependency) should be included as separate components. Even though this issue relates to the determination of system boundaries as opposed to the definition of component, it represents an important issue as it influences the number of components and thus the discussions based on the metrics. An additional challenge is that some technologies, e.g., JavaScript, enforce the source-code to be part of a code-base and thus influence the definition of the components (and other metrics).

D. Application

Challenges involved in the introduction and application of the metric are grouped together in six different concepts.

Model Introduction: The transition to the new quality model has not been without challenges. After the initial introduction there was an additional need for an elevator pitch for the new metrics (requested in one memo). In addition, the investment needed to define components for a large number of systems written in older technologies, combined with viewing the definition of components as subjective, made one client decide not to upgrade yet to the new quality model for their portfolio.

Definition of Actions: On three occasions different assessors indicate that defining actions based on the architecture metrics was more involved than providing advice for code-level metrics. On the code-level it is relatively straight-forward to define general actions (e.g., remove this type of duplication or refactor these long methods), but constructing this type of advice for the architecture metrics requires more effort because these recommendations are more context-dependent.

Effect prediction: Related to the definition of actions, four memos describe that assessors are not always certain about the effect the implementation of a recommendation has on the value of the metrics. Especially because the addition of a component potentially affects both metrics, the results of implementing a recommendation is seen as harder to predict and could be smaller or bigger than desired.

For example, according to one assessor the effects of adding a component to a small system can significantly influence the overall rating, but one memo describes an example in which this influence was neglectable. Because of this uncertainty, assessors ask for tool-support to simulate the implementation of a recommendation in three different memos.

Steering: One memo describes that the metrics are relatively stable for systems which are in maintenance mode. Any change in the metrics is normally the result of a targeted effort and thus expected. This makes it harder to use the metrics to steer development on a weekly basis, which is seen as disappointing.

Context: As indicated in Section VI-A suppliers directly ask for recommendations to increase the rating for a metric. In these situations it is tempting to focus on either one of the metrics to increase only that value. However, five memos describe discussions which point out that it is important to keep in mind that the eventual goal is not to have perfect ratings, but to have an architecture which fits the current needs.

For example, for one system the Component Independence received a low rating, while the rating for all other system properties are high. Combining this with the observation that the system has only one open issue at this time any effort spend to increase the rating for the this particular aspect is unneeded.

Effort prediction: In line with the finding about providing recommendations, the effort prediction related to improving architectural issues is seen as difficult. Two memos describe that the effort needed to group components together is deemed to be lower than the effort required to split up components in separate chunks of functionality. In some situations a low rating is related to only a few violations, while in other situations a large refactoring is needed. Because of this the assessors experience difficulties in applying a general effort prediction model.

E. Implementation

Eight memos describe implementation issues regarding Component Balance (three) or Component Independence (five).

Component Balance: Two memos (involving one assessor) describe a discussion about the implementation of the function to determine the rating for the number of components of a system. The conclusion of this discussion was that even though the function could be improved, the impact on the way in which the metric can be applied would be small. Secondly, the use of Lines of Code to depict the size of components is, according to one memo, not always applicable to XML-based languages.

Component Independence: A first observation is that only using inter-language dependencies has a high impact on systems for which one component is implemented in a different technology. For example, in two systems one of the components was dedicated to an SQL-type language, while the other components used an Object-Oriented language. In these situations the rating for Component Independence was considered to be too high.

Secondly, two memos describe two different systems in which incorrect call resolving caused false positives, which

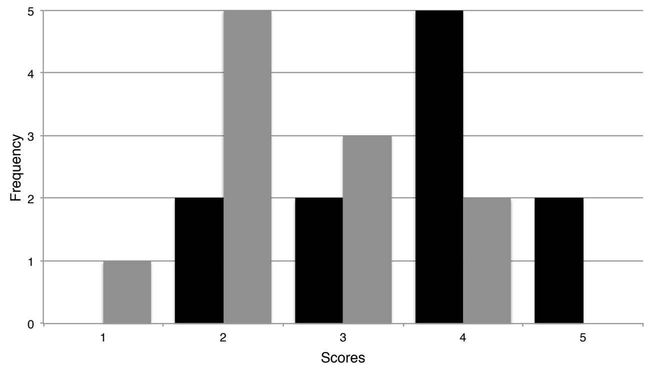

Fig. 4. Histogram displaying the distribution of scores related to the usefulness (black) and ease of use (gray) of the architecture metrics as given by the eleven interviewees.

in turn results in a rating which is too low (because modules are put into the wrong category).

Lastly, one memo outlines a conversation between an assessor and a supplier regarding the topic of dependency injection. As argued by the supplier, the difference in constructing a class directly or using a framework to construct and inject a class is small, but using the first approach significantly impacts the rating for Component Independence in a negative way.

VII. Interview Findings

A total of eleven quality assessors were interviewed on three days over a period of one week. During the interviews notes were taken by the interviewer which formed the basis for a report of each interview. These reports have been validated by the interviewees.

The results of the interviews consists of two parts: a quantitative part based on the scores given by the interviewees and a qualitative part in which the reports of the interviews are analyzed. The analyses of the reports has first been done by the authors on an individual basis, after which the results were discussed and combined. These results are presented in the remainder of this section based on the categories and concepts as displayed in Figure 3. Whenever a new finding could not be related to an existing concept, but was mentioned by at least two interviewees, a new concept was introduced.

A. *Interviewee scores*

An overview of the scores given by the interviewees is displayed in Figure 4. Scores related to the second question, whether the metrics are considered to be useful, are displayed in black, scores related to the third question, whether the metrics make their job easier, are displayed in gray.

From the distribution of scores we can deduce that overall the metrics are perceived to be useful, but that the application of the metrics does not make the job of assessments easier.

Related to the usefulness, nine interviewees indicate that the metrics provide a starting point for discussions about the components of a system. In three interviews specific examples of how the metrics identified problems within a system were discussed, while in two interviews examples of the use of the metrics as a communication device were given. Lastly, three different interviewees provide examples of targeted improvement efforts.

In relation to making the job of the consultant easier different types of challenges were brought forward, these challenges are discussed below.

B. *Intuition*

The concepts associated with this category were only discussed briefly. For example, only one interviewee mentions that smaller systems tend to score lower on the component metrics. And even though systems implemented in older technologies were discussed in four interviews, only one interviewee mentioned that they tend to score lower. Additional insights for this concept are discussed below. The seemingly large influence of the number of components on the rating was not discussed in any interview.

Older technologies: One interviewee explained that older technologies tend to receive a lower rating, which was in-line with his expectation. Interestingly, another interviewee mentions that systems with older technologies normally receive a slightly better rating (an observation also made in the memos). From this we conclude that the exact influence of the architecture metrics on systems written in older technologies varies. Lastly, another interviewee mentions that for older technologies no meaningful componentization exists, thus the component metrics should play a less significant role in the overall assessment.

C. *Component definition*

The exact definition of components was a substantial topic in ten of the eleven interviews. In particular, each of the three concepts as described in the memos were mentioned by at least three interviewees. Additional insights regarding these existing concepts are discussed below. In addition, the concept of *Technologies without components* is added to cover new findings in the interviews.

File-system versus mental model: Seven interviewees mention the challenge of choosing the "right" view on the components of a system. Apart from the mental model and the file-system, views related to the deployment of the code or functional decompositions can be chosen. A decision about which view should be leading in the calculation of the metrics is requested.

Subjectivity: The fact that multiple view-points can be used to calculate the metrics leads to a feeling that the component definition is subjective. Although this flexibility is considered to be a good thing in the context of an SRA, where different view-points lead to different insights, the added flexibility sometimes leads to unwanted discussions about what constitutes a component (especially when the rating for the component metrics is low).

Technologies without components: In three interviews it was mentioned that for some technologies the concept of a "component" does not exists. This does not only include older technologies such as Cobol and Tandem, but also systems implemented in newer technologies such as SAP. Additionally, visual programming languages, typically used to model business processes, also do not have an inherent concept of a component. It is yet unclear what the best way is to apply the component metrics to systems written in these technologies.

D. Application

With respect to the application of the metrics all concepts were mentioned in at least one interview. Several new insights were obtained for two different concepts. In addition, the concept of *Responsibility* is introduced to capture new findings.

Definition of actions: Nine interviewees mention that the definition of actions is harder for the architectural metrics than for the code-level metrics in the model. This is one of the main reasons why the new metrics do not make it easier to perform assessments. This increased effort needed to perform assessments is not necessarily seen as problematic. Definition of the actions might be harder, but is also seen as more interesting, challenging and valuable. However, to deal with common situations more efficiently one assessor suggests to collect experiences of applying the metrics to derive common recommendations.

Steering: What is problematic is that in some situations the advice related to the components metrics can involve a substantial amount of refactoring. In some situations this leads to the identification of problems that cannot be solved due to resource constraints, which limits the usability of the advice in the setting of an SRA.

In a monitor setting the metrics can initially fluctuate, which is seen as problematic by one interviewee. However, two other interviewees do not find this problematic as long as there is a roadmap towards a stable set of components. One interviewee mentions that within a monitoring setting it is easier to do something to improve the underlying architecture, although this is not necessarily done on a weekly basis.

Responsibility: An additional challenge in the application of the metrics is that there exist situations in which the development team does not feel responsible for the components of the system. In some cases the components cannot be changed by the development team because the technology or framework dictates the component-structure; in other cases the components are mandated by a person outside the development team. In these situations a discussion about the value of the component metrics is considered to be useless by the development team, which hinders the application of the metrics.

E. Implementation

Ten interviews mentioned issues related to the implementation category. Issues related to the definition of Component Balance mainly revolved around a new concept of *Optimal number of components*; issues related to the other two concepts were not mentioned. With respect to Component Independence all three concepts were discussed at least once.

Component Balance - Optimal number of components: As explained in Section IV-B the "optimal" number of components is currently defined as the median number of components in a benchmark, which is currently 7. In five interviews this number is discussed, in all cases it is questioned whether the highest rating for SBO should be attached to this number only.

VIII. Discussion of Findings

The data extracted from the 48 memos and the eleven interviews illustrate the usefulness of the Component Balance and Dependency Profiles metrics in the assessment of implemented architectures performed by external assessors. Examples show that the metrics can trigger targeted improvement efforts, start meaningful discussions and can be used as a communication device in a number of different situations.

The evaluation also illustrates situations in which the metrics perform less than optimal. For example, because the metrics are relatively stable for systems in maintenance mode they cannot be used to steer development on a weekly basis. Moreover, in some situations the recommendations following from the metrics require a significant amount of effort, which is not always available due to resource constraints.

In these situations the metrics illustrate a problem that is not subject to improvement, which reflects upon the perceived usefulness of the metrics.

Apart from these limitations, several areas of improvements are identified. We discuss those areas related to the three most discussed topics: Component Definition, Application and Intuition.

Component Definition: The need for a strict definition of what entails a component across technologies is an important topic of discussion. Even though the ability to use different view-points is seen as both a positive and a negative aspect, a strict definition of component is asked for on several occasions. As illustrated in Section VI-A there is a notion that the components of a system should be in line with the structure on the file-system, but the interviews indicate that such a definition is not applicable to all technologies.

In order to improve on this situation we plan to follow-up on the advice of one interviewee and collect a representative set of component definitions for different technologies. Such a set can be used as a basis for determining the components of existing systems, and provides an opportunity to derive general as well as technology-specific guidelines.

Application: With respect to the application of the metrics both the observations and the interviews indicate that defining actions based on the value of the metrics is not always straightforward. Moreover, estimating the amount of effort involved in implementing recommendations is considered challenging.

To deal with this problem we plan to build up a body of knowledge containing common value-patterns and associated recommendations for the architecture metrics. Having these common patterns and the effects of implementing the recommendations on the metrics available makes it easier for the assessors to gain a better feeling for the interpretation and application of the architecture metrics. The data collected in this evaluation should be considered a first step towards this body of knowledge.

Intuition: According to the assessors, the value of the metric for smaller systems and systems written in specific technologies are not always as expected. Specific reasons for the mismatch between value and intuition seem to be the current determination of the "optimal" number of components for Component Balance and missing cross-language dependencies for Component Independence.

The overall intuition of the assessors could indicate that specific groups of systems should be treated differently by the metrics. In our future work we will perform statistical analyses on the values of the metrics in different groups in order to validate this hypothesis.

To address the specific issues we plan to investigate ways to implement support for cross-language dependencies in a cost-efficient manner. In addition, findings ways to make the resolving of dependencies more precise is deemed to be an important part of our future work. For Component Balance we plan to investigate ways to better define the "optimal" number of components.

IX. Reflections on Evaluation Methodology

In this section we reflect upon the benefits of evaluating the usefulness of metrics in practice and the used process.

First of all, the main benefit of performing this type of evaluation is a better understanding of the usefulness of the software metrics. Moreover, the effective identification of possible improvement areas illustrates the benefits of performing such an evaluation.

However, an important question here is whether the application of the metrics brings new insights, or whether all findings could have been defined before the evaluation. For example, one could argue that the need for a strict component definition or the questions related to systems written in different technologies could have been defined before the metrics were applied. Even if this is true, we believe that the relative importance of the different areas of future work could not have been determined in a purely academic context.

A second question related to the benefits of this type of evaluation is whether the results can be generalized to different contexts. In principle all findings are limited by the context as defined within Section IV. However, given the depth of this evaluation we believe that the benefits of the architecture metrics also apply to external assessors in different settings.

Note that it is important to be able to place the value of the metrics in a context, for example by the use of a benchmark. Because of this, the usefulness of the metrics for developers working on a single system is considered to be limited.

In relation to the followed methodology we make three important observations. First, the use of two different types of data-gathering is important to ensure a balanced evaluation. Secondly, the confidentiality constraints inherent to the evaluation of metrics within an industry setting limits reproducibility of the results. Lastly, the embedding of metrics within a standard operating procedure can be challenging.

Balanced Evaluation: Every data-gathering technique has known limitations. For example, the interviews provide an indication of the usefulness of the metrics as perceived by the assessors. As pointed out by Davis [13], perceived usefulness is not necessarily the same as objective usefulness. A limitation of the gathering of observations is the inherent confirmation bias of the observer.

By combining the data from both methods we believe that these limitations are partially countered. In addition, the two methods are executed by two different persons to increase the possibility of finding new information in both methods. Furthermore, the interviews are conducted by the one author that does not have daily interactions with the interviewees to reduce interviewer bias.

Based on the new findings in the interviews and the discovered areas of future work we believe that combining these two types of data-gathering leads to a balanced and critical evaluation of the usefulness of software metrics in practice.

Reproducibility: Due to reasons of confidentiality, the data collected within the memos and the interviews cannot be made publicly available. However, we believe that the description of the data as given in Section VI and Section VII is detailed enough to support the conclusions drawn from the data.

Metric Embedding: The biggest challenge in executing the proposed methodology is the embedding of new metrics in a standard operating procedure on a large scale, a topic that is out of scope of the current research. However, the benefits of evaluating metrics in practice as described above and in Section VIII is intended to assist researchers in acquiring the needed commitment from industry partners.

X. Related Work

Empirical evaluations of software metrics typically consist of evaluating the value of a metric against one of three external properties: quality in terms of faults, effort (either development or maintenance) or volume [22].

By contrast, theoretical approached of metric evaluation inspect mathematical properties of metrics [15] or focus on metrological properties of metrics [1].

These types of evaluation aim to determine whether a metric is indeed measuring the attribute it was designed for in a theoretical manner. Kaner et al. [21] stress that this type of evaluation should also be done using a more practitioners oriented view-point and defines a framework for evaluating the validity and risk of a metric in the form of 10 questions.

All of the above evaluation strategies are meant to be done before a metric is used. Although useful, this pre-deployment validation covers only part of the 47 different validation criteria for metrics recently summarized in a literature review [27]. Our evaluation of the usefulness of software metrics bests fits the *actionability* criteria, which is defined as:

> A metric has actionability if it allows a software manager to make an empirically informed decision based on the software product's status [27]

To the best of our knowledge, no empirical evaluation of this validation attribute has been done for specific metrics.

XI. Conclusion

This paper describes a large-scale industrial evaluation of the usefulness of the Component Balance and Dependency Profile metrics, in the context of the assessment of implemented architectures, from the view-point of external quality assessors. Using two different methods for gathering data, a detailed overview of the benefits and challenges of the two specific metrics is constructed and discussed.

For SIG, this evaluation identified different areas for improving the application of the metrics which have led to the definition of concrete improvement actions. For other practitioners, this evaluation can be used to decide whether or not the two architecture metrics can be used in their assessment processes. For the research community, the overview of areas for future work in Section VIII, and the detailed overview of the data as discussed in Section VI and Section VII, can be used as a starting point for conducting new research.

In addition, a methodology for evaluating software metrics in practice was introduced and the benefits and limitations of this approach are discussed. For practitioners, the overview of the insights gained from this type of evaluation is intended to inspire practitioners to collaborate with researchers to perform similar types of evaluations. For researchers, the methodology can serve as a starting point for evaluating the usefulness of (new) software metrics in practice, and can be reflected upon to improve the methodology itself.

To summarize, this paper makes the following contributions:

- It introduces a methodology for evaluating the usefulness of software metric in industry
- It describes the execution of this methodology in an empirical study towards understanding the usefulness of two specific software metrics
- It provides an overview of challenges involved in the application of the two specific software metrics and lists concrete areas for improvement
- It reflects upon the usefulness of the evaluation methodology, concluding that the relative importance of challenges involved in applying specific metrics cannot be determined in a purely academic setting.

References

[1] A. Abran. *Software Metrics and Software Metrology*. Wiley-IEEE Computer Society Pr, 2010.

[2] S. Adolph, W. Hall, and P. Kruchten. Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16(4):487–513, Aug. 2011.

[3] A. Albrecht and J. Gaffney, J.E. Software function, source lines of code, and development effort prediction: A software science validation. *Software Engineering, IEEE Transactions on*, SE-9(6):639 – 648, 1983.

[4] T. L. Alves, J. P. Correia, and J. Visser. Benchmark-based aggregation of metrics to ratings. In *IWSM/Mensura*, pages 20–29, 2011.

[5] R. Baggen, K. Schill, and J. Visser. Standardized code quality benchmarking for improving software maintainability. In *4th International Workshop on Software Quality and Maintainability (SQM 2010)*, 2010.

[6] V. R. Basili, G. Caldiera, and H. D. Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[7] E. Bouwers, J. Correia, A. van Deursen, and J. Visser. Quantifying the analyzability of software architectures. In *Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA 2011)*. IEEE Computer Society, 2011.

[8] E. Bouwers, A. van Deursen, and J. Visser. Dependency profiles for software architecture evaluations. In *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM 2011)*. IEEE Computer Society, 2011.

[9] E. Bouwers, A. van Deursen, and J. Visser. Quantifying the encapsulation of implemented software architectures. Technical Report TUD-SERG-2011-031, Delft Software Engineering Research Group, Delft University of Technology, 2011.

[10] L. C. Briand, J. W. Daly, and J. Wüst. A unified framework for cohesion measurement in object-orientedsystems. *Empirical Software Engineering*, 3(1):65–117, July 1998.

[11] L. C. Briand, S. Morasca, and V. R. Basili. Defining and validating measures for object-based high-level design. *IEEE Transactions on Software Engineering*, 25(5):722–743, 1999.

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

[13] F. D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q.*, 13(3):319–340, 1989.

[14] A. v. Deursen and T. Kuipers. Source-based software risk assessment. In *ICSM '03: Proceedings of the International Conference on Software Maintenance*. IEEE Computer Society, 2003.

[15] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., Boston, MA, USA, 2nd edition, 1998.

[16] A. Gopal, T. Mukhopadhyay, and M. Krishnan. The impact of institutional forces on software metrics programs. *Software Engineering, IEEE Transactions on*, 31(8):679 – 694, aug. 2005.

[17] M. Greiler, A. van Deursen, and M. Storey. Test confessions: a study of testing practices for plug-in systems. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 244–254, Piscataway, NJ, USA, 2012. IEEE Press.

[18] I. Heitlager, T. Kuipers, and J. Visser. A practical model for measuring maintainability. In *QUATIC '07: Proc. 6th Int. Conf. on Quality of Information and Communications Technology*, pages 30–39. IEEE Computer Society, 2007.

[19] International Organization for Standardization. ISO/IEC 9126-1: Software engineering - product quality - part 1: Quality model, 2001.

[20] International Organization for Standardization. ISO/IEC 25010: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011.

[21] C. Kaner and W. P. Bond. Software engineering metrics: What do they measure and how do we know? In *10TH International Software Metrics Symposium - Metrics 2004*, 2004.

[22] B. Kitchenham. Whats up with software metrics? A preliminary mapping study. *Journal of Systems and Software*, 83(1):37 – 51, 2010.

[23] H. Koziolek. Sustainability evaluation of software architectures: a systematic review. 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*, QoSA-ISARCS '11, pages 3–12, New York, NY, USA, 2011. ACM.

[24] T. Kuipers and J. Visser. A tool-based methodology for software portfolio monitoring. In *Software Audit and Metrics, Proceedings of the 1st International Workshop on Software Audit and Metrics*, pages 118–128. INSTICC Press., 2004.

[25] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen. The ability of object-oriented metrics to predict change-proneness: a meta-analysis. *Empirical Software Engineering*, 17:200–242, 2012. 10.1007/s10664-011-9170-z.

[26] T. J. McCabe. A complexity measure. In *ICSE '76: Proceedings of the 2nd international conference on Software engineering*. IEEE Computer Society Press, 1976.

[27] A. Meneely, B. Smith, and L. Williams. Validating software metrics: A spectrum of philosophies. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 21, 2012.

[28] N. Nagappan and T. Ball. Static analysis tools as early indicators of pre-release defect density. In *Proceedings of the 27th international conference on Software engineering*, ICSE '05, pages 580–586, New York, NY, USA, 2005. ACM.

[29] 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*, 34:700–720, 2008.

[30] S. Sarkar, G. M. Rama, and A. C. Kak. API-based and information-theoretic metrics for measuring the quality of software modularization. *IEEE Transactions of Software Engineering*, 33(1):14–32, 2007.