Constructing Application Landscape Metrics:

Why & How

Technical Report TB 0701

Josef Lankes, Christian M. Schweda

Software Engineering for Business Information Systems (sebis)
Ernst Denert-Stiftungslehrstuhl
Chair for Informatics 19
Technische Universität München

Boltzmannstraße 3, 85748 Garching b. München, Germany

{lankes, schweda}@in.tum.de

July 2007

Abstract

The report presents metrics as an approach for supporting the management of an application landscape by quantifying specific structural aspects. It derives basic guidelines for the construction of such metrics from a comparison of current application landscape metrics to metrics in software engineering and business administration. Thereby, it puts a special emphasis on integrating the metrics into methodologies for addressing concerns relevant to application landscape management. The report exemplifies these guidelines by describing application landscape metrics for testability aspects, together with a viewpoint for visualizing and a methodology for using the metrics.

Keywords: *application landscape, metrics, testability*

Contents

1 Motivation

Nowadays organizations operate hundreds or even thousands of business applications in order to support their business processes [LMW05]. These applications, when taken together, form the application landscape. The application landscape provides critical support to processes of the respective organization and is in turn evolved and operated by this organization, creating a complex web of interdependencies: Business applications are installed on hardware, are operated by organizational units or modified by projects, which should be in accordance to certain business objectives, just to name a few relevant concepts. In practice, concepts for supporting the management of this environment are often based on information models that deal with these concepts, together with the respective associations. Thereby, "information model" refers, in accordance with [Bu07], "to a model which specifies, which information [...] should be documented, and how the respective information should be structured. This is usually achieved via a model expressed in a language suitable for conceptual modeling, as e.g. UML, enriched with descriptions detailing the exact meaning of the concepts".

The structure of the application landscape with its interdependencies exerts a strong influence on critical success factors [AD05, BT00, Ro00]. As an example, the ability to incorporate new requirements into the application landscape or the ability to adapt to new requirements depend on a structure friendly to modifications.

Decisions that influence the structure of the application landscape, e.g. in projects, can be considered critical, affecting the value of the application landscape as a long-term, important investment of an organization. We hold the opinion, that such decisions should not be totally based on ad-hoc arbitrations made in a state of insufficient information about their potential effects. This sets the stage for explicit evaluation techniques, as e.g. metrics, that are able to assess implications of changes. Quite similar, decision making in the field of business administration is often supported by metrics and measures, as e.g. yield and risk measures for assessing financial investments. Comparable problems in the domain of software engineering have been addressed by software metrics and architectural metrics [Fe91, Ve05, AM96].

In order to leverage research available in the area of software metrics and business science, the work pinpoints differences between use of metrics regarding application landscapes and use of metrics in software engineering and business administration (section 2). The goal is to derive lessons learned, which are used for providing guidelines for metrics based methodologies in the context of application landscape management (section 3), which are exemplified by a metrics system for assessing testability (section 4).

2 Status Quo

Use of metrics is common to a multitude of domains. Many of the foundations of measurement theory originate from natural sciences. Further developments have been introduced in social sciences [Kr71] or economics [Ei87]. This section summarizes, how metrics are used in the management of application landscapes, not by merely listing *what* aspects have been measured in a specific way, but by contrasting this metrics utilization with two fields, which can be seen as bordering to the management of application landscapes: *software engineering* and *business*

administration.

Metrics in software engineering, as e.g. described by [Fe91], target a wide range of aspects, from the software development processes to the different products created therein, as architectures, sourcecode, or the developed software itself. Thereby, this article's focus tends toward product and sourcecode metrics.

Business administration, ranging from strategy to operations research, from human resources to finance, encompasses a multitude of fields where measurement plays a role. To provide a manageable counterpart for the comparison mentioned above, the article focuses on some of these areas as recurring examples: measures in finance and accounting ratios, and the balanced scorecard.

The contrasts between current application landscape metrics and software engineering metrics on the one hand, and measurement in business administration on the other hand, are sketched in respect to different aspects below.

2.1 Maturity of the field

The use of metrics in the context of application landscapes is relatively young. When taking literature as an indicator for the maturity of the field, two characteristics become visible:

Availability of Literature: Relatively few literature is available about metrics utilization for application landscapes. Some works presented on conferences [AD05, SW05] or contained in monographs [Ma05] touch the area. We have also found attempts of practitioners, which have not been made public but are sketched in the respective project documentation, most often in presentation slides.

Product metrics in software engineering, in contrast, date back to e.g. the McCabe [Mc76] metric, which was introduced to measure complexity aspects of software from sourcecode. Since then, the field has brought forward a multitude of different approaches for measuring software, from information theory based architecture metrics [Sh02] to metrics sets specialized in object oriented software, including metrics designed to be applied to architectures or designs instead of sourcecode. Today, periodicals, as e.g. the *Journal of Software Measurement*[1] present metrics as a mature field in software engineering.

Regarding business administration, literature describing aspects of measurement abounds in a multitude of fields. Accounting ratios and basic guidelines for their interpretation are covered by textbooks in the respective field [RWJ02]. The same is true for other measures in finance and investment, as e.g. return measures or risk measures for investments [RWJ02]. Metrics targeting the performance of companies are not solely the subject of textbooks, but also of research, as e.g. documented in dissertations [Sc05]. Regarding the balanced scorecard, a lot of literature has been published since the introduction of this instrument in [KN91], discussing e.g. the application of the balanced scorecard to different fields.

Theoretical Foundations: Even fewer literature is available regarding theoretical foundations of application landscape metrics. Contrary to this, several theoretical treatments of software engineering metrics have been published, from general treatments of the topic [Fe91] to more

[1] See `http://jsm.cs.uni-magdeburg.de`.

specialized elaborations, e.g. on metrics validation [EE00].

Summary: The above comparison presents use of metrics for application landscapes as a rather young field, in need of publications that enable discussion and development of such approaches. In order to avoid theoretical mistakes common to measurement and known from other domains, we deem it advisable to build metrics on a sound theoretical basis. Such theoretical foundations might very well be "borrowed" from the fields introduced above.

Nevertheless, we see works with their focus on specific metrics and their use in *practice* as more fruitful than contributions narrowly focused on "theory of application landscapes metrics", as we see a need for approaches for managing application landscapes, but currently not a community concerned with such a theory in itself.

2.2 Conceptual Models

Basically, the process of measurement can be described as assigning numbers to objects of classes in a way to faithfully represent certain properties [Kr71]. Thus, measurement can be seen as based on a *conceptual model* of the respective domain, defining the classes and their relations. These concepts determine, *what* can be counted, summed up, etc. to derive a measure.

In the context of software engineering, this does not commonly appear as problematic. In many areas, these conceptualizations have been readily available and strictly defined when the respective metrics have been built. An example are control flow graphs, on which McCabes Cyclomatic Complexity [Mc76] is built.

The same is true for many metrics in business administration. Accounting ratios as mentioned above can rely on a substantial body of regulations and guidelines in the field of accounting, which can also serve as definitions for concepts relevant to finance specific metrics. Other fields in business administration can be seen as laying less stable foundations for the definition of metrics. When looking at a balanced scorecard, which might e.g. use goals like "customer satisfaction" or "employees per *", it becomes obvious, that no exact and universally accepted definition of a customer[2] or an employee[3] exists.

This more resembles the situation regarding the management of application landscapes and the conceptual models used therein, which we subsequently call *information models*[4]. Unfortunately, no common information model for application landscapes exists, as described in [Bu07]. Defining such information models is aggravated by ambiguously used terms in this field. This is problematic to metrics definition, as it endangers the advantage of metrics being objective characterizations of the aspect under consideration: a metric is likely to conceal subjectivity by providing a perfectly objective measurement procedure, not taking into account subjective interpretations or ambiguous definitions of the concepts in the underlying model.

Summary: [Bu07] presents a pattern-based approach for creating information models, which we deem applicable also in the context of application landscape metrics. This approach

[2] Consider e.g. how to count institutional customers vs. private customers.
[3] Consider e.g. part time employees or temporary employees.
[4] We have found this to be common terminology in the field of application landscapes.

deviates from the common practice in management of application landscapes to start with the creation of an information model covering all relevant aspects of the application landscape. Instead, the pattern-based approach focuses on a specific problem, and provides the concepts necessary in this respect. Regarding metrics definition, this means that an information model pattern defining the concepts for specific metrics is provided. Such patterns can then be used in constructing comprehensive, organization specific information models.

This can be seen as congruent with the two domains alluded to above. After all, while there are many well-defined metrics in the context of software engineering, there is no single conceptual model of this domain. Object-oriented metrics for example, rely on the concepts introduced in the paradigm of object orientation. Failure modeling, in contrast, might be more concerned with redundant and non-redundant units and their failure probabilities.

The situation is similar in business administration. Accounting ratios rely on the balance sheet as an abstraction of the organization under consideration. Other finance-specific metrics might include stock prices or market capitalization, however abstracting from the field of human resources, which may be of fundamental importance to certain metrics used in a balance scorecard.

When providing precise definitions of the concepts used in the information model, necessary to enable the provision of objective data for metrics calculation, two points should be kept in mind. Firstly, the importance of a shared understanding of the concepts, e.g. of a "part time employee". Secondly, that the understanding should be useful in the context of the metrics and methodologies, in which it is employed.

In order to minimize subjective influence on the models built according to an information model underlying a metrics definition, we propose paying special attention to the incorporation of constructs into the information model, of which it can be easily and objectively verified, whether they actually apply.

2.3 Metrics Validation

In software engineering, the validation of metrics has not been without some misunderstandings and misconceptions, as e.g. detailed in [Fe91]. In order to avoid such problems, we use the terms introduced by [Fe91] to provide more conceptual rigor in metrics validation:

- *Internal Validation/Theoretical Validation* is meant to demonstrate that a metric characterizes the attribute it is meant to measure [EE00]. This can be seen as proofing that the attribute can be measured on a certain level of measurement.
- *External Validation/Empirical Validation* builds on the internal validation and tries to establish that the metric is useful in predicting an important characteristic (also called external attribute). This is relevant as the real benefit of a metric is the information it provides about this external attribute.

A common approach for empirical validation is based on the construction of a model explaining the relationship between the metric under consideration and the respective external attribute. Statistical hypothesis testing is then a possibility to confirm or refute such a model.

This approach is propagated by literature about software engineering metrics, as e.g. in [Fe91] or in the "laws and theories" from [ER03].

Statistical studies for confirming such models are always subject to empirical data, on which the specific statistical techniques can work. Thereby, difficulties can arise in the field of application landscapes, where data collection is known as problematic [LMW05].

Summary: Above elaborations point to the importance of models explaining the relationship between a metric and an external attribute.

Firstly, in the creation of such a model, assumptions regarding the conditions, under which the supposed relationship between the metric and the external attribute holds, become explicit. This enables reflecting how realistic these assumptions are, leading to refinement or refutation of the model, without conducting a statistical survey. Also, an underlying model should, according to [Fe91], be present before such a statistical validation is performed.

Moreover, an useful description of a methodology employing a metric should encompass the assumptions, under which a metric is a valid predictor of the respective external attribute. It has to make sure, that the assumptions are true under the usage conditions. A model as described above simplifies this.

Also theoretical validation should not be neglected when defining metrics and the methodologies using them. It should be verified, whether a metric is consistently defined. Thus, issues in the metric can be found without an empirical validation, or without finding them via potentially costly errors in practical use.

2.4 How metrics are used

Software engineering has brought forward many different ways of utilizing metrics. Using the models linking metrics to external attributes for actually predicting the external attributes may not even be the most prominent one, due to the often extensive data needs for model parameterization. Other ways may include the support of review processes, e.g. via using metrics to determine what should be subjected to an especially thorough review [Ve05]. Such approaches are sometimes supported with metrics visualizations. Management dashboards, giving an overview of specific metrics values, are another example here.

In business administration, visualization of metrics to support decision making is widely used, with Boston Square Diagrams as an example. The Balanced Scorecard [KN91] is a prominent example of metrics in strategy management.

Summary: The multitude of approaches for using metrics in efforts to address specific concerns in software engineering or business administration shows that in devising metrics based methodologies, one should not feel constrained to a specific approach. Thereby, of course, the concern, which is to be addressed by the respective methodology, should always be in the focus of the efforts.

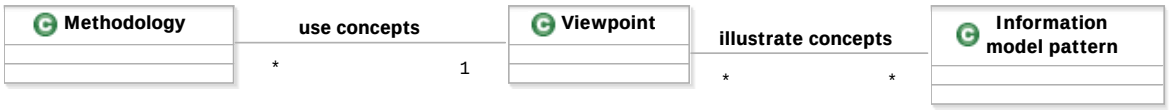

Figure 1: Three kinds of Enterprise Architecture Management patterns [Bu07]

3 Approach to Metrics for Application Landscapes

A metric in itself is only a means to pursue specific ends, and, if used in an engineering-like manner, a methodology describes how it can be employed to pursue such an end, addressing a concern in a planned way. In order to focus on the practical use alluded to in section 2.1, we propose to have the description of application landscape metrics connected to methodologies that employ them in order to address specific concerns.

Thereby, we rely on the pattern-based approach mentioned in section 2.2, detailed in [Bu07] and outlined in Figure 1, with metrics usage concerning the three kinds of patterns as follows:

- the information model pattern contain the conceptual model defining both the classes, on which the metrics are defined, and the metrics characterizing objects of specific classes, e.g. as attributes
- the viewpoint describes, how these metrics can be visualized
- the methodology is a description how to systematically address a concern, here focusing on how the metrics values are derived and how these values should be employed in addressing the respective concern

Subsequently, the metrics-related issues of these three concepts are explored.

3.1 Methodologies and Visualizations for Metrics

As sketched in section 2.4, there are many different ways to benefit from metrics in methodologies for application landscape management. A small subset of these may include:

Support of review processes Metrics may e.g. be used to detect areas of the application landscape where analysis determining potential for improvement can be expected to be especially fruitful.

Support for decision making Metrics may present information in a more suitable form, e.g. by preparing, enriching, or summarizing.

Support for goal operationalization Metrics may be used in operationalizing goals or controlling goal achievement, similar as in a balanced scorecard.

Adequate visualizations can play an important part here, e.g. different Excel-like diagrams, portfolio matrices, or layers[5] on software maps [Bu07, LMW05], as exemplified in section 4.

[5]Layering here can be understood as in e.g. Microsoft Visio.

These possibilities should be fully taken into account, also considering that not every aspect of the respective methodology has to be addressed strictly by using a metric. "Not everything that counts can be counted, and not everything that can be counted counts", can serve as a proverbial summary here.

3.2 Metrics Definition and Information Models

To serve as a solid building block of a methodology, a metric should be defined in an unambiguous way. A metric thus defined can be subjected to internal validation, as outlined in section 2.3. This ensures that the metric is consistently defined. Ruling out, or being able to control unsuitable behavior is essential to avoiding systematic measurement errors, potentially flawing the methodology using the respective metric.

Furthermore, only knowledge of the level of measurement enables efficient use of a metric in a methodology. It establishes, which interpretation of the respective metric values are meaningful, and which not. Assuming a level of measurement too low forfeits valuable information, whereas assuming a level of measurement too high leads to using information, which might actually not be contained in the respective values, yielding unjustified results.

The basis for a metric definition as described above is a suitable conceptual model or information model. As outlined in section 2.2, we propose the conceptual model, on which the metric definition is built, to be based on an information model pattern.

Section 2.2 also alludes to the benefits of objectively enforceable modeling guidelines in information models. When e.g. modeling business objects used by business applications, it is to a certain degree a subjective decision, at which granularity such objects are modeled. However, when considering a deployed business application, and a set of infrastructure service realizations, it can be seen as rather objectively verifiable, on which of the service realizations the deployed business application depends.

3.3 Making Assumptions Explicit: Models and Theories

Underlying a metrics based methodology are assumptions about how a specific metric connects to a specific concern. While different kinds of assumptions can be involved therein, the link between metrics and external attributes, as described in section 2.3, is often essential in metrics based methodologies.

The models or theories giving this link should be explicated, if not in the description of the methodology itself, then in another document where they can be referenced. Such models should also explicitly state, under which conditions they are applicable. It then clearly has to be ensured by the methodology that these usage conditions are met when the methodology is applied. A core problem hereby is tied to the "objectively enforceable modeling guidelines" mentioned in section 3.2. Due to insufficient objectively enforceable modeling guidelines, it may be possible that some precautions have to be taken to know with a specific degree of certainty that the application landscape is modeled in a way leading to usable results for the respective metric. Possibilities here include:

- Reviews could be conducted, possibly on excerpts of the respective part of the application landscape model, to check whether it is a suitable base for metrics calculation, or to discover possible caveats for metrics interpretation.
- Model changes could be tied to review workflows, in order to keep the respective models constantly in a state suitable for metrics calculation.
- Test procedures could be applied, offering more potential for automatization. If a metric is used to make predictions about an external attribute, the prediction accuracy could be checked ex post, if the respective values of this attribute are available.
- Visualizations could be employed, contrasting metric values with ex-post realizations of the respective external attribute. If tests are not deemed appropriate, one may thus get an impression of the predictive qualities of a metric.

Last but not least, it can be noted that a metrics based methodology directly profits from successful validation studies of the underlying models.

4 Exemplifying the Approach

The approach to metrics based evaluation of an application landscape outlined in section 3 is subsequently exemplified via the influence of the application landscape structure on an issue well known from the field of software engineering, *testability of a business application*. While testability is certainly not the only relevant criterion influenced by the structure of an application landscape, it should not be neglected in considerations affecting this structure, as e.g. selecting projects or setting up long term visions of the application landscape.

In the field of application landscapes, testability has e.g. been stated as an important aspect by [Wo06], or by Deutsche Börse Systems, which defined the term in a project conducted together with the chair for Software Engineering for Business Information Systems as follows:

Tests with predefined test coverage should be realizable with reasonable effort. In this context the functional coverage is important as well as the size of the test installation. The functional coverage of the test implicates the tested amount of functionality, e.g. how many test cases of which size are executed? The size of the test installation is influenced by e.g. the number of systems and interfaces taking part in the test, whether batches are part of the test, etc. To minimize the effort, the tests should include as few systems as possible.

We choose testability for exemplifying our approach, as we deem it relatively straightforwardly addressable, compared e.g. to flexibility or changeability. Thus, we try to focus on how to use the guidelines from section 3 in defining metrics, instead of having to focus on an overly complex metrics definition and its underlying theory. Below, an introductory section provides basic ideas of the metrics set to be presented, followed by three sections presenting an excerpt of the metrics in the context of a methodology as described in section 3, while a last section sketches the theoretical foundations of the metrics, as suggested in section 3.3. The descriptions allude to how similar constructs might be used in defining changeability metrics.

4.1 Basic concepts

There are obviously many differing approaches to "testing", originating from different fields and usage contexts. In order to give explicit account for the intended usage context, we subsequently give the definition of testing this article relies on by explicating the testing process assumed therein.

In this testing process, the tests are conducted with knowledge of the test objects' interfaces and functionalities, but without knowing details of its implementation, as in a black box test according to [So04]. Thereby, we see the testing process as composed of the following four activities. This results in corresponding efforts for an installation/deployment of a business application b providing services supporting the business (subsequently called `BusinessServiceProviderInstance`, as defined in section 4.2):

Test case construction Here, the functionalities that should be tested have to be identified. For these functionalities, adequate test cases have to be derived. The respective effort is subsequently called $eTestPlanning(b)$.

Infrastructure setup The deployment to be tested has to be linked to a set of infrastructure elements that actually enable it to operate. Here, an infrastructure functionally identical to the actual operating infrastructure has to be provided. $eInfraSetup(b)$ is the respective effort.

Simulated environment setup This task encompasses simulating the environment of the deployment in an appropriate way. This may include the construction of simulations for systems the actual deployment exchanges data with. $eSetupTestEnv(b)$ constitutes the respective effort.

Test execution This activity, for which the effort is called $eTestExec(b)$, is composed of actions as e.g. invoking operations on interfaces, specifying the used data or recording operation results, in order to execute the planned test cases.

Having defined our testing process as above, it becomes obvious, that on the one hand distinct aspects of the structure of the application landscape may influence the testing effort for specific BusinessServiceProviderInstances. On the other hand, also situative aspects in the actual testing process influence the testing effort. As the later factors do not depend on the structure of the application landscape, we chose not to address them in depth, but to focus on systematic aspects, tied to the following application landscape specific concepts: *The functionalities that should be tested, the instances of BusinessServiceProviders that provide these functionalities, and the infrastructure services these instances rely on.*

Subsequently, we incorporate such concepts in an information model pattern as described in section 3.2, a viewpoint, and a methodology, relying on each other to support testability considerations in planning the evolution of an application landscape. Thereby, we describe four metrics, each acting as an indicator for $eInfraSetup(b)$ respectively. We do not address the other steps of the assumed testing process with their effort components here, due to reasons of brevity.

Similar assumptions could most probably be relevant to changeability metrics. When trying to model the influence of the application landscape structure on e.g. change efforts, basic assumptions about the process in which these efforts occur are most probably relevant.

4.2 Information model pattern

The information model pattern modeling testability related concepts is concerned with application landscape elements providing functionalities supporting the business, so called *BusinessServices*. Furthermore, infrastructure components supportive of the respective service providers are modeled in the information model pattern as shown in Figure 2. The concepts are defined as follows:

BusinessServiceProviderInstance represents the entity providing realizations of business services. Such a provider can be a deployment of a business application.

BusinessServiceProvider represents a type of actual providers for realizations of business services. This may be a specific business application or its version that can be deployed.

InfrastructureServiceRealization represents infrastructure services that are operated and support instances of business service providers. A service realization may be e.g. an actual "http-service". We view the usage of InfrastructureServiceRealizations as *objectively verifyable* according to section 3.2, making it a suitable component in an information model for metrics calculation.

InfrastructureService represents a type of service in the field of infrastructure. This type can be realized by different InfrastructureServiceRealizations.

InfrastructureServiceProviderInstance represents the entity providing realizations of infrastructure services. Such a provider can be a deployment of a system, e.g. an installation of "Oracle 9.2i".

InfrastructureServiceProvider represents a type of providers for realizations of infrastructure services. Such a type could be "Oracle 9.2i".

The metrics are here represented by derived attributes. Subsequently, we provide OCL expressions for calculating the values of the derived attributes, while a detailed discussion of the metrics is provided in section 4.4.1:
context `BusinessServiceProviderInstance::NoISRealizations:Integer`
derive: `reliesOn->size()`
context `BusinessServiceProviderInstance::NoISPInstances:Integer`
derive: `reliesOn->provider->asSet()->size()`
context `InfrastructureService::NoUsages:Integer`
derive: `realizations->supports->size()`
context `InfrastructureServiceProvider::NoInstances:Integer`
derive: `instances->size()`

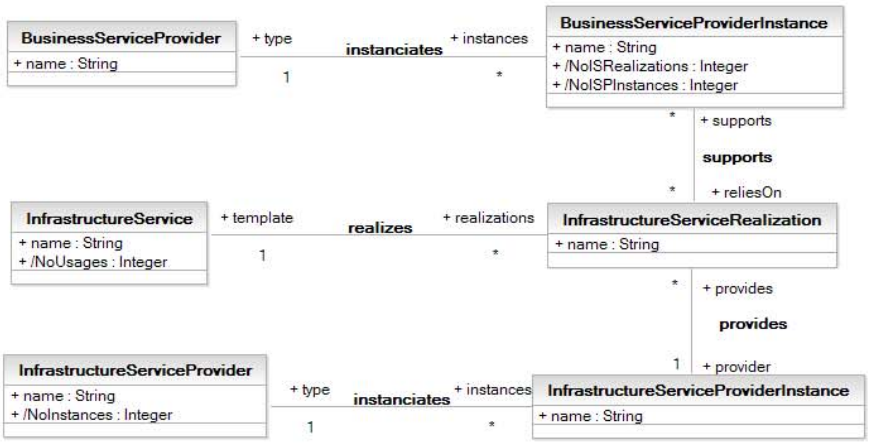

Figure 2: The information model pattern used in the testability metrics

4.3 Viewpoint

For visualizing testability aspects based on the information model pattern outlined above, a viewpoint, exemplified in Figure 3, is given. This viewpoint comprises a software map, as described in [Bu07], consisting of four layers, as detailed below:

The *BusinessService layer* contains rectangular symbols representing the BusinessServiceProviderInstances under consideration. These symbols comprise three compartments, the topmost labeled with the name of the instance and of the corresponding provider. On this layer, the second compartment shows the value of $NoISRealizations$, the third shows $NoISPInstances$.

The *InfrastructureService layer* contains rectangles representing InfrastructureServiceRealizations supportive of the BusinessServiceProviderInstances. The rectangles are labeled with the name of the respective service realization and the name of the service. Further, they are placed in the second compartment of the symbol representing the supported BusinessServiceProviderInstance. Thus, the same InfrastructureServiceRealization may be represented by multiple rectangles, if they are used by different BusinessServiceProviderInstances. The background color of the rectangles indicates $NoUsages$.

The *InfrastructureServiceProvider layer* contains rectangles representing InfrastructureServiceProviderInstances, which provide InfrastructureServiceRealizations and support the BusinessServiceProviderInstances. The rectangles on this layer are labeled with the name of the respective InfrastructureServiceProviderInstance and the name of the respective InfrastructureServiceProvider. Further, they are placed in the third compartment of the symbol representing the associated BusinessServiceProviderInstance. As above, the same InfrastructureServiceProviderInstance may be represented by multiple rectangles, of which the background color is used to indicate $NoInstances$.

The *ProvidesRelationship layer* contains lines attached to the symbols representing *providing* InfrastructureServiceProviderInstances and their *provided* InfrastructureServiceRealizations, as long as they are located in the same BusinessServiceProviderInstance symbol.

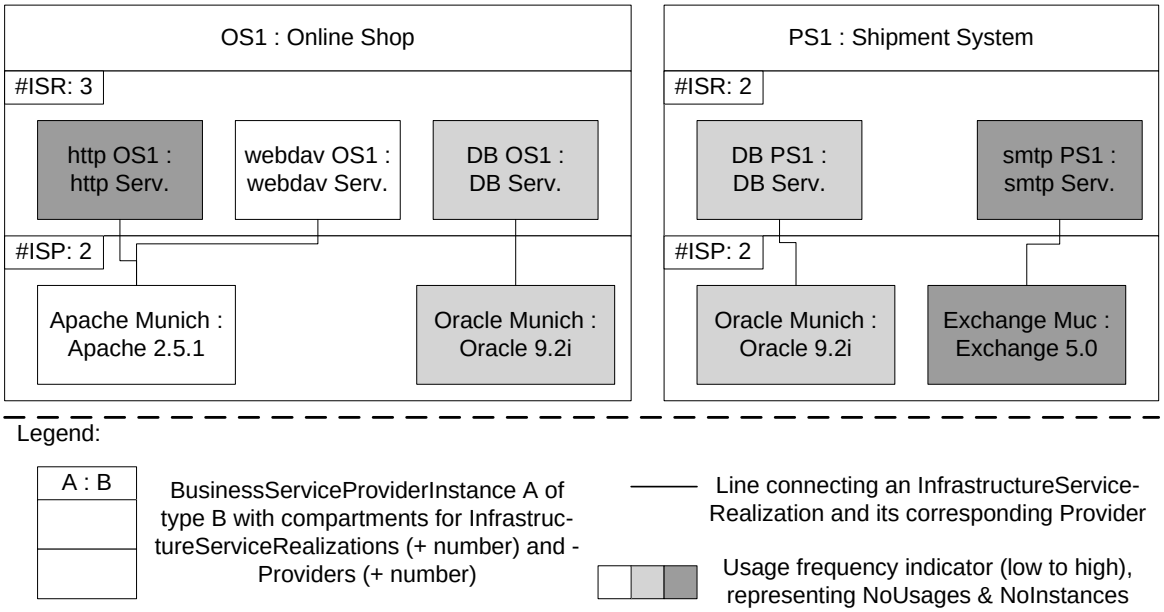

Figure 3: The viewpoint used in the testability metric

4.4 Methodology

4.4.1 Metrics and Metrics Calculation

Subseqently, the four metrics introduced above are detailled, especially giving the assumptions under which they can be used as an indicator of the respective effort.

NoISRealizations(b), as obvious from the OCL definition given in section 4.2, counts the instances of InfrastructureServiceRealizations referenced ($b.reliesOn$). This metric can be used as an indicator of $eInfraSetup(b)$ under the following assumptions:

- For each InfrastructureServiceRealization used by b, a corresponding InfrastructureServiceRealization is used in the test installation.
- The effort necessary for configuring the test installation to use a specific InfrastructureServiceRealization is roughly the same for each InfrastructureServiceRealization. This does not mean that all efforts must be the same. It only means, that the discrepancies must lie within specific limits, and that there should be no concentrations of hard or easy to configure InfrastructureServiceRealizations at specific BusinessServiceProviderInstances, in order to ensure the predictive quality of the metric.

NoISPInstances(b) counts the InfrastructureServiceProviderInstances that together provide the set of InfrastructureServiceRealizations ($b.reliesOn$) used by b, as defined by the OCL expression in section 4.2. This metric is another indicator for $eInfraSetup(b)$, under the following assumptions:

- The test installation is set up to have its InfrastructureServiceRealizations provided by

a set of InfrastructureServiceProviderInstances that has a corresponding InfrastructureServiceProviderInstance for each InfrastructureServiceProviderInstance used by b.

- The effort necessary for setting up an InfrastructureServiceProviderInstance is roughly the same for each InfrastructureServiceProviderInstance. Again, this does not demand identical efforts. It only means, that the differences of the effort must be within certain limits, again also avoiding concentrations of hard or easy to install instances at specific BusinessServiceProviderInstances, as this would endanger the predictive quality of the metric.

NoInstances(isp) represents the number of instances of an InfrastructureServiceProvider *isp* that exist in the used model of the application landscape. If the subsequently stated assumptions hold, the metric can be used as an indicator for the effort necessary for setting up an instance *ispi* of *isp* in the test installation, which might influence $eInfraSetup(b)$ if there is an InfrastructureServiceRealization r for which $r \in b.reliesOn$ and $r.provider = ispi$ holds.

- The model from which the metric is calculated covers the application landscape rather completely, giving an accurate impression of the number of instances of a InfrastructureServiceProvider actually used.
- Due to learning effects, installing a "common" InfrastructureServiceProvider tends to be connected to less effort than installing a rather "exotic" one.

NoUsages(is), counting how often an InfrastructureServiceRealization of a given type supports a BusinessServiceProviderInstance, is also based on learning effects. It can be used as an indicator for the effort necessary for configuring a BusinessServiceProviderInstance to use a InfrastructureServiceProviderInstance, if the following assumptions hold. Then, it is also an indicator of $eInfraSetup(b)$ for a respective BusinessServiceProviderInstance b.

- As with $NoInstances(isp)$, the metric relies on the model being complete enough to reflect the actual number of usages in the application landscape.
- Also here, we assume a tendency towards reduced effort in using InfrastructureServiceRealizations of a commonly employed type (i.e. InfrastructureService).

Although we merely discuss them in the context of testability here, the last two metrics might also be useful in the context of other homogenization or consolidation specific concerns.

Outlook on other possible metrics: Certainly, a complete metrics set would have to consider all activities in the assumed testing process. Thereby, the functionalities of the BusinessServiceProviderInstances would have to be considered. While we do not describe the necessary methodologies and concepts here, for reasons of brevity, we are currently developing an approach for describing these functionalities by linking a colored petri net based process model to the BusinessServiceProviderInstances. From such a model, metrics could be calculated, which measure structural aspects about the functionalities influencing e.g. $eTestPlanning(b)$, $eSetupTestEnv(b)$, or $eTestExec(b)$.

4.4.2 Testability Assessment

Metrics as described above can be used when assessing and discussing different *scenarios* of an application landscape. A *scenario* in this respect may be a planned version of the application landscape, which could be created by one or more projects. Also different planned states, created by alternative solutions via alternative projects, can be seen as *scenarios* here. Due to their ordinal nature (see section 4.5), they should not be added, e.g. in order to get an impression of the total testabiliy of an application landscape.

When making assessments of such scenarios, a multitude of different aspects has to be considered, ranging from the functionality a solution can provide to the business to the knowledge implementing the solution demands from developers. The above metrics can help in addressing testability aspects. Therefore, they rely on the viewpoint from section 4.3, in which the visualized metrics provide additional information to reasoning about testability. This information could e.g. lead to certain aspects being examined more thoroughly, or additional realization strategies, potentially characterized by better metrics values, being considered.

In order to safely use the metrics, the respective assumptions should be checked. Thereby, the user of the methodology firstly has to decide whether his testing process shares similarities with the one on which the metrics are built. Then, the assumptions of the specific metrics and the suitablity of the actual models should be examined, with approaches as mentioned in section 3.3 basically applicable, and knowlege about the theoretical model behind the metrics helpful (see section 4.5). Here,especially $NoInstances(isp)$ and $NoUsages(is)$ should be taken into consideration. If these metrics indicate a concentration of InfrastructureServiceRealzations and InfrastrcutrueServiceProviderInstances that are hard or easy to install and configure, one should check, whether such tendencies might be opposed to the relationship of the efforts indicated by the other metrics. This would certainly have to be considered in an interpretation of the respective metrics.

If data about the actual test effort is available, it may be advisable to use this possiblity to make ex-post checks of the metrics predictive quality.

4.5 Theoretical foundations

On the one hand, the subsequent models clarify the assumptions of the metrics system, which have already been verbally sketched in section 4.4. On the other hand, we view the construction of such models as an important step in constructing such descriptions, as it allows to clearly verify their theoretical plausibility. Handling random variables, as used below, solely by textual arguments might too easily lead to false assumptions.

Before we sketch the models linking the metrics to the respective efforts below, we shortly discuss the level of measurement of the metrics introduced above. Basically, $NoISRealizations$ and $NoISPInstances$ are defined as cardinalities of sets, thus being (at least) ordinally scaled allowing the operations used below. This especially means, that one should be careful in adding $NoISRealizations$ and $NoISPInstances$, as e.g. two BusinessServiceProviderInstances might share e.g. an InfrastructureServiceProviderInstance and the tests might e.g. be carried out in a shared test environment. This ordinal interpretation also has to be applied to the $eInfraSetup$ below.

$NoUsages$ and $NoInstances$ being simple counts of elements, are rationally scaled, allowing the use of covariance as employed below.

Modelling $eInfraSetup(b)$ The relevance of the metrics used as indicators for $eInfraSetup(b)$ is based on modelling this effort as consisting of two basic components: the effort for setting up InfrastructureServiceProviderInstances $eInfIns(b)$ and the effort for configuring the test installation in a way that it uses the InfrastructureServiceRealizations of these InfrastructureServiceProviderInstances $eInfConf(b)$, thus $eInfraSetup(b) = eInfIns(b) + eInfConf(b)$ holds.

The effort for setting up an InfrastructureProviderInstance *ispi* of type *isp* is composed of a systematic effort (random variables $isEffort^{sys}(isp)$ with a shared expectation) and a situative effort (random variables $isEffort^{sit}(ispi)$ with expectation 0). The systematic effort depends on the InfrastructureProvider to be set up itself, the situative one on other factors, as e.g. the tester, possible interruptions, etc. The effort for all InfrastructureServiceProviderInstances for b is derived by summing up these efforts.

The effort for configuring the test installation to use InfrastructureServiceRealizations *isr* of type *is* is derived by summing up the effort for each service realization, which is composed of a systematic (random variables $scEffort^{sys}(is)$ with a shared expectation $scEffort^{sys}$) and a situative part ($scEffort^{sit}(isr)$ with expectation 0).

The model is here detailled in resepct to $eInfConf(b)$, while leaving out a similar explication linking $NoISInstances(b)$ to $eInfIns(b)$ for reasons of brevity: assuming effort distributions as above, $eInfConf(b) = \sum_{r \in b.reliesOn} (scEffort^{sys}(r.type) + scEffort^{sit}(r))$.
Thus, subsequent statements about expectation and variance of $eInfConf(b)$ can be made[6]:

- $E(eInfConf(b)) = NoISRealizations(b)\ scEffort^{sys}$. This[7] shows, that the higher $NoISRealizations(b)$, the higher the $eInfConf(b)$ and thus $eInfraSetup(b)$, if all other factors stay the same, that can be expected.
- $Var(eInfConf(b)) = \sum_{r \in b.reliesOn} (scEffort^{sys}(r.type) + scEffort^{sit}(r))$
  $+2 \sum_{i<j; i,j \in eInfConfSummands(b)} Cov(i,j)$.
  Therein, a set containing all summands adding to $eInfConf(b)$ is defined as:
  $eInfConfSummands(b) = \bigcup_{r \in b.provides} (scEffort^{sys}(r.type) \cup scEffort^{sit}(r))$.

The variance is important in making statements about the predictive quality of the metric. The smaller variances as mentioned above are, the smaller is the probability that an $eInfConf(b1)$ is smaller than $eInfConf(b2)$, although *b1* received a higher metric value than *b2*. Thus, the subsequent properties positively influence the predictive quality of the metric, when comparing two BusinessServiceProviderInstances *b1* and *b2*:

- $Var(eInfConf(b1))$ and $Var(eInfConf(b2))$ should be small, which means that the services should be modelled in a way that the terms of which these two variances are composed, variances of and covariances between the $scEffort^{sys}(r.type)$ and $scEffort^{sit}(r)$

[6] Basically, variance and expectation make only sense for metric variables. While we deem the interpretations made here as applicable, the values should be interpreted carefully.

[7] This does not directly extend into the case where two BusinessServiceProviderInstances are concerned (e.g. due to shared infrastructure), reflecting the ordinal nature of the metric mentioned above. However, the metric is able to address the comparison of different plans or scenarios as described in section 4.4.

(for *b1* and *b2* respectively), should be small. This corresponds to the statements "the effort [...] is roughly equal" and "there should be no concentrations" made in section 4.4.1.

- A big $Cov(eInfConf(b1), eInfConf(b2))$ is advantageous, as it counters tendencies that by chance the two efforts diverge in the opposite directions[8]. While we do not derive this covariance here from covariances between the components of $eInfConf(b1)$ and $eInfConf(b2)$, one might hint that this term might be rather high in case *b1* and *b2* are rather similar, making the metric e.g. especially suitable for comparing subsequent versions of the same BusinessServiceProviderInstance. This corresponds to the statement "predictive quality is improved in case subsequent versions [...] are compared" in section 4.4.1.

Due to learning effects, $Cov(isEffort^{sit}(ispi), NoInstances(isp)) < 0$ and $Cov(scEffort^{sit}(isr), NoRealizations(is))) < 0$, with $isp = ispi.type$ and $is = isr.type$, can be assumed, as they incorporate these effects with commonly used InfrastructureServices or InfrastructureServiceProviders. These influences might negatively affect the above properties on which the predictive quality of the metrics $NoISRealizations(b)$ and $NoISPInstances(b)$ depends. $NoUsages(is)$ and $NoInstances(isp)$ allow capturing these learning effects related influcences, as allued to in the statement about reduced efforts for a "common type" and higher efforts for a "rather exotic InfrastructreServiceProvider" in section 4.4.1.

Statistical modeling, as employed above, might easily be relevant to modeling changes of an application landscape as a foundation for changeability metrics. Certain change efforts might e.g. be modeled as random variables, possibly with covariances to specific attributes of the application landscape, as e.g. interface counts.

5 Outlook

Of course, the exemplary metrics for testability can only be a starting point for supporting management of application systems with metrics based methodologies. On the one hand, we have to admit that we made the simplest possible assumptions in the construction of the metrics. All of the assumptions made can be challenged and improved in any specific situation, further developing and improving the metrics set.

On the other hand, testability is only one, and probably not the most important aspect about an application landscape, which leads to the development of other metrics as an obvious next step. In this respect, we view the aspects "changeability/modifiability" and "support of the functional processes by the application landscape" as promising areas.

Such metrics provide a way to enhance methodologies with a way to use and present information more adequate to the respective stakeholders and concerns. They can process data to present specific information focused on specific stakeholders. They can summarize or selectively present information, to give users of a methodology improved overview. In all this, they can rely on explicit theories, possibly minimizing the influence of implicit assumptions, which may originate from overgeneralizations, trends or hypes.

[8] See the equation $Var(x - y) = Var(x) + Var(y) - 2Cov(x, y)$.

References

[AD05] Aier, S.; Dogan, T.: *Indikatoren zur Bewertung der Nachhaltigkeit von Unternehmensarchitekturen.* In (Ferstl, O. K. et al., Editor): *7. Internationale Tagung Wirtschaftsinformatik 2005*, Bamberg, Germany, 2005. Physica-Verlag.

[AM96] Abreu, F.; Melo, W.: *Evaluating the Impact of Object-Oriented Design on Software Quality.* In (?, Editor): *Proceedings of the 3 rd International Software Metrics Symposium*, Berlin, 1996. IEEE Computer Society.

[BT00] Byrd, T.; Turner, D.: *Measuring the Flexibility of Technology Infrastructure: Exploratory Analysis of a Construct. Journal of Management Information Systems*, 17(1), 2000.

[Bu07] Buckl, S. et al.: *A Pattern based Approach for constructing Enterprise Architecture Management Information Models.* In *8. Internationale Tagung Wirtschaftsinformatik (to be published)*, Karlsruhe, 2007. Karlsruher Universitätsverlag.

[EE00] El Emam, K.: *A Methodology for Validating Software Product Metrics*, 2000.

[Ei87] Eichhorn, W. E.: *Measurement in Economics.* Physica-Verlag, Heidelberg, 1987.

[ER03] Endres, A.; Rombach, D.: *A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories.* Pearson Education, Harlow, England, 2003.

[Fe91] Fenton, N.: *Software Metrics. A rigorous Approach.* Chapman & Hall, London, 1991.

[KN91] Kaplan, R.; Norton, D.: *The Balanced Scorecard - Measures That Drive Performance. Harvard Business Review*, 70(1):S. 71–79, 1991.

[Kr71] Krantz, D. et al.: *Foundations of Measurement*, volume 1, Additive and Polynomial Representation. Academic Press, New York, 1971.

[LMW05] Lankes, J.; Matthes, F.; Wittenburg, A.: *Architekturbeschreibung von Anwendungslandschaften: Softwarekartographie und IEEE Std 1471-2000.* In (Liggesmeyer, P.; Pohl, K.; Goedicke, M., Editor): *Software Engineering 2005*, volume P-64 of *Lecture Notes in Informatics (LNI) - Proceedings*, pages 43–54, Essen, Germany, 2005. Köllen Druck+Verlag.

[Ma05] Masak, D.: *Moderne Enterprise Architekturen.* Springer, Berlin, 2005.

[Mc76] McCabe, T.: *A Complexity Measure. IEEE Transactions on Software Engineering*, 2(4), 1976.

[Ro00] Ross, J.: *Creating a Strategic IT Architecture Competency: Learning in Stages. MIS Quarterly Executive*, 2(1), 2000.

[RWJ02] Ross, S.; Westerfield, R.; Jaffe, J.: *Corporate Finance.* McGraw-Hill, Boston, 6th, edition, 2002.

[Sc05] Schedler, B.: *Leistungsmessung in multinationalen Unternehmen.* Dissertation, Universität St. Gallen, 2005.

[Sh02] Shereshevsky, M. et al.: *Information Theoretic Metrics for Software Architectures.* In *Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC 2001)*, Chicago, 2002. IEEE Computer Society.

[So04] Sommerville, I.: *Software Engineering.* Pearson Education, Essex, England, 7th edition, 2004.

[SW05] Schwinn, A.; Winter, R.: *Entwicklung von Zielen und Messgrößen zur Steuerung der Applikationsintegration.* In (Ferstl, O. K. et al., Editor): *7. Internationale Tagung Wirtschaftsinformatik 2005*, Bamberg, Germany, 2005. Physica-Verlag.

[Ve05] Verlage, M. et al.: *Überwachung der Qualität der Architektur einer Software-Produktlinie am Beispiel eines web-basierten Wertpapierinformationssystems.* In (Liggesmeyer, P.; Pohl, K.; Goedicke, M., Editor): *Software Engineering 2005*, volume P-64 of *Lecture Notes in Informatics (LNI) - Proceedings*, Essen, Germany, 2005. Köllen Druck+Verlag.

[Wo06] Wolff, H.: *Transparenz schaffen, Strukturen erkennen, Landschaften gestalten. Eine Fallstudie zur Einführung von EAM bei der SBB.* Presentation, 2006.

