Using Metrics to Evaluate Failure Propagation in Application Landscapes

Josef Lankes, Christian M. Schweda

Chair Software Engineering for Business Information Systems
Technische Universität München
Boltzmannstr. 3
85748 Garching
{lankes, schweda}@in.tum.de

Abstract: The number of business applications operated by enterprises to provide support for the business has increased over the last years. With increasing number of applications and connections between them, also the overall complexity of the application landscape has grown. A main task in managing an application landscape is maintaining quality levels. This article contributes a set of metrics for measuring a specific attribute of an application landscape – *failure propagation.*

1 Motivation

Today, enterprises operate a large number of applications providing critical support to the business. These business applications form, when taken together, the application landscape of an enterprise, which can be seen from a managerial point of view as an important asset, providing essential support to business processes, but also acting as a sometimes costly bottleneck. Together with the growing complexity, among others created by an increasing number of business applications and dependencies between them, this results in the need to manage the application landscape in a structured and continuous way. This has lead to approaches as Enterprise Architecture (EA) management [LW04] or Enterprise Modeling [Fr02] gaining increasing importance.

However, not all approaches available to management activities are common in managing application landscapes. Metrics, while readily employed in other domains as Software Engineering [Fe91] or Business Administration (see e.g. the balanced scorecard [KN91], or financial metrics [RWJ02]), cannot yet be considered widely used in managing application landscapes (as one of few examples, [La05] presents a quantitative approach for assessing performance and workloads for applications). However, we view metrics, an approach to quantify properties of objects [Kr71], as a powerful aid for management activities in coping with an application landscape's complexity. Especially, visualizing properties by displaying metrics values on diagrams as e.g. software maps [Bu07b] can enable users to focus on aspects otherwise not easily visible in the often large application landscape models, which range into thousands of business applications.

One of the aspects, which are of concern regarding an application landscape, is the propagation of failures through the often dense web of interdependencies between the business applications, with potentially critical and unforeseen effects on the supported business processes. It is an aspect especially of importance concerning systems being directly visible to customers, with the kind of business being conducted by the organization certainly affecting the criticality (consider e.g. an insurance vs. a stock exchange). Thus, we provide a metrics based approach to assist in evaluating structures in an application landscape which are prone to promoting failure propagation. We do not view this as a standalone approach, but as one to be integrated with existing efforts concerned with managing an application landscape, also ones described in literature (as e.g. [BW05]). Thereby, the approach can complement evaluations of other quality attributes.

The remainder of this article is structured as follows: Section 2 details our approach to assess the effect of the application landscape structure on failure propagation, in the context of an EA management pattern as introduced in [Bu07a]. Therefore, firstly the concerns to be addressed by measuring failure propagation are detailed and secondly an appropriate information model fragment containing metrics is given. Thirdly, viewpoints utilizing the metrics are presented, which are then in the last part of the approach used by methodologies to address the initial concerns. Section 3 shows the applicability of the approach in practice and thereby exemplifies adaptations, which have to be made to tailor the approach to an actual use case. The example is based on a case study executed in cooperation with a large Swiss bank. Section 4 points to further directions of research.

2 Measuring failure propagation

With the basic makeup of an application landscape as a complex web of interdependent applications, which communicate a plethora of information objects over a multitude of communication channels, interfaces, and interconnections, it is not surprising, that the failure of such an interconnected application is likely to cause other applications not to perform correctly or to perform at all. This behavior exerts influence on the support for business processes provided by the application landscape, and therefore on the business services provided by the enterprise itself. Various approaches for evaluating failure-specific aspects exist. At the level of business processes, e.g. quality control is concerned with the subject, employing approaches as control charts, cause-and-effect diagrams, or QC process charts for process analysis [Is90]. Subsequently, we provide a metrics-based approach, which, while not neglecting process aspects, is more focused on the application landscape.

As a metric is only a means in pursuing a specific end, it must be related to a methodology that gives an indication on how the metric can be employed in addressing the specific concern. A methodology itself can be seen as based on specific viewpoints, which visualize the underlying information. Therefore, we structure our presentation of the failure propagation metrics along three elements: information model, viewpoint (which can together be seen as a modeling language), and methodology (here describing some aspects to be considered in an associated process model). Preceding detailed information on these three parts, the basics underlying our approach are introduced.

2.1 Basic concepts

The basic idea of failure propagation here is that a business application is dependent on services rendered by other business applications in order to perform specific tasks. A service can either be necessary for other business applications or directly related to the provision of a business service and therefore to supporting a business process.

The model subsequently introduced assumes that services are exposed via interfaces, so that it can be indicated for interfaces, whether they can provide the respective service (operational) or not (not operational, failure). The idea behind this model is to focus on inter-application aspects. Figure 1 exemplifies this, showing a situation where the execution of an Order Management process employs an “Account Info” (user)-interface and an “Execute Transaction” (user)-interface. These interfaces are offered by an employee portal, which in turn uses other interfaces offered by two more business applications. However, considering that not all used interfaces might be necessary for keeping a specific offered interface of an application operational, the model is also designed to be able to use specific intra-application details. The Employee Portal e.g. relies on “Archiving” and “Accept Transaction” for providing “Execute Transaction”, while its “Account Info” relies on an Account Info – interface from the Order System, as shown in Figure 2.

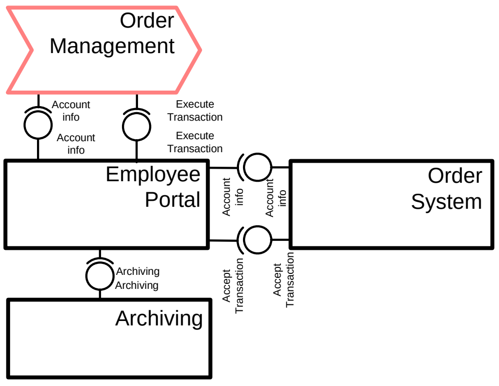

Figure 1: Landscape view on failure propagation

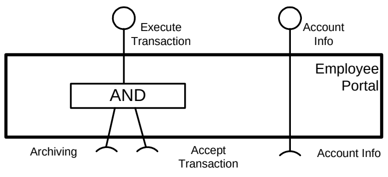

Figure 2: Intra-application view on failure propagation

In modeling failure propagation, the concept of failure trees, as e.g. in [Ge89] is used as a well-understood theoretic foundation. In a failure tree, e.g. an offered service (interface in our model) is represented by the root, while the interfaces used in performing the service make up the leaves of the tree. In between, different nodes representing different kinds of dependencies exist, e.g. AND-nodes meaning that both of the child nodes must be operational (see Figure 2). Both the inter-application and the intra-application aspect allow making statements about and thus measuring, how prone the application landscape structure is to failure propagation, and what risks this might pose for the business processes. The necessary concepts therefore are introduced in detail in the subsequent section.

2.2 Information model pattern

To introduce the concepts of interest when the propagation of failures between business applications is concerned, we provide an information model containing the respective classes, attributes and associations. For the *inter-application* aspect of failure propagation, the concepts as introduced in the model in Figure 3 are important[1]. These concepts, i.e. classes and associations are explained below – a description of the derived attributes (the metrics) is given at the end of this section.

`BusinessApplication` A business application in this context is a system, which is implemented in software, deployed at a specific location and which provides support for at least one business process. In performing the business support, a business application may be dependent on other applications, which is modeled by two associations to the offered or used interfaces (**`offers`** and **`uses`** respectively).

`BusinessProcess` A business process is here understood as a sequence of individual functions with connections between them. A business process as used in this information model should not be identified with a single process step or individual functions, but with high-level processes at a level similar to the one used in value chains. In executing a business process, a number of interfaces offered by business applications is used (see association **`uses`**), therefore effectively making the process execution dependent on the utilized business applications. Such a dependency is exemplified in Figure 1, giving an indication of the interfaces used by the business process “Order Management”.

`OfferedInterface` An offered interface represents a service, which is provided by a **`BusinessApplication`** and is intended for external use by e.g. another business application or **`BusinessProcess`** (in which it could be a user interface employed in manual execution). Such an interface has an interface type associated (see **`isOfType`**) and can be connected to many using entities (see **`connected`**). This connection relationship is exemplarily visualized in Figure 1, showing **`OfferedInterfaces`** of business applications via the lollipop symbol.

`InterfaceUsage` Interface usage represents the fact that a business application or a business process makes use of an **`OfferedInterface`** of another business application during execution. For having a more detailed view on the services rendered via an interface used actually employed in execution, the interface usage refers to an interface type (see **`isOfType`**). Interface usages are exemplarily visualized in Figure 1 via the socket symbol, e.g. showing that the Employee Portal uses the Archiving interface offered by the Archiving business application.

`InterfaceType` The type of an interface gives information about e.g. the different types of information exchanged or services rendered/used.

[1] Concerning the attributes of the classes displayed, it is assumed that every class holds an attribute **`name`**. This attribute is therefore omitted in the diagram.

Different interface types can be related using a sub-super relationship (see **`hierarchy`**), such that an **`OfferedInterface`** could provide some set of information/services, from which a corresponding **`InterfaceUsage`** only uses a part. In this example, a subtype would be offered, i.e. a supertype would be used. The information represented/services described by an **`InterfaceType`** are detailed by the **`DescribingAspects`**. Figure 1 annotates the type of each **`OfferedInterface`** and **`InterfaceUsage`** next to the respective symbols.

`DescribingAspect` An aspect of an interface could be a specific operation provided by the interface. These aspects then together build the entire interface with its full-scale functionality. **`DescribingAspects`** are not explicitely shown in Figure 1 and 2.

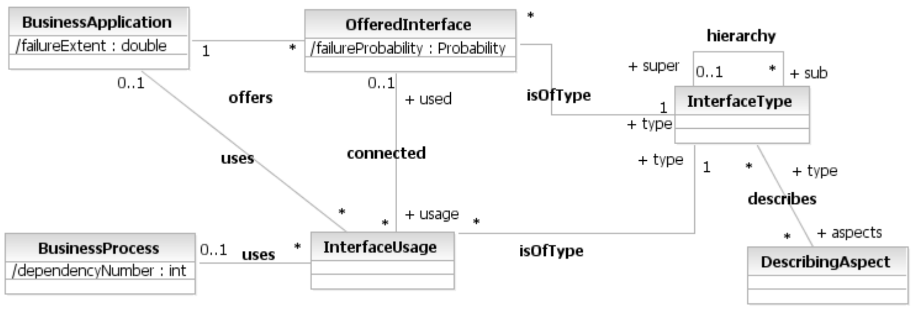

Figure 3: Information model for inter-application failure propagation

On the information in Figure 3, some constraints apply, with the essential ones being:

```
context: InterfaceType
inv: (super==null) or aspects->includesAll(super.aspects)

context: InterfaceUsage
inv: (used==null) or used.type.aspects->includesAll(type.aspects)
```

These invariants establish the sub-/supertype hierarchy of the **`InterfaceType`**s and constrain an **`InterfaceUsage`** to be connected only to an appropriate **`OfferedInterface`**. Refining these concepts of failure propagation on the intra-application level, the core concepts of the intra-application failure propagation can be described in an information model as in Figure 4.

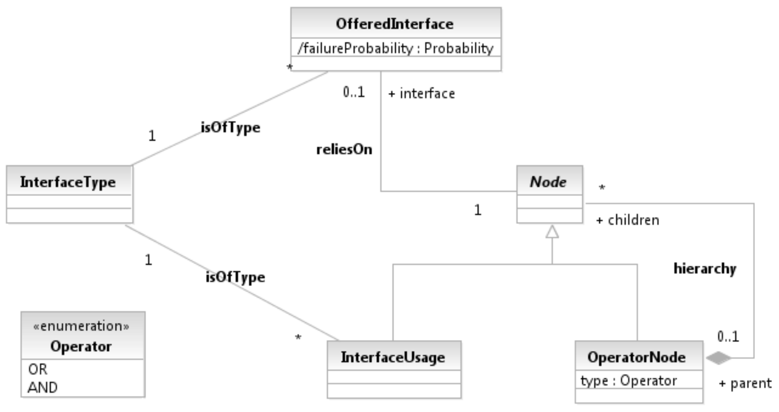

Figure 4: Information model for intra-application failure propagation

`Node` A node represents the abstract basic element of a failure tree. It either can be a leaf node as represented by an **`InterfaceUsage`**, or an **`OperatorNode`** corresponding to a Boolean operation as detailed below. A node can evaluate to different status regarding operation, i.e. operational and non-operational.

`OperatorNode` This node is connected to child nodes (see **`children`**) which represent prerequisites for the **`OperatorNode`** to have operational state. The node itself has an attribute referring a corresponding Boolean operation associated, determining whether all prerequisites (Operation **`And`**) or at least one prerequisite (Operation **`Or`**) is necessary for the **`OperatorNode`** to be in operational state.

Additionally, rules for establishing the tree-like buildup have to be incorporated:

```
context: Node
derive: ancestors=new Set(){parent}->union(parent.ancestors)
derive: descendants=children->union(children.descendants->asSet())
inv: descendants->excludes(self)
inv: ancestors->excludes(self)
```

Subsequently, the derived attributes are explained, beginning with the simplest metric which is contained in the information model, the **`dependencyNumber`** of a **`BusinessProcess`**. It represents the number of **`BusinessApplications`**, of which the failure directly leads to the failure of an **`OfferedInterface`**, which the process uses. The **`failureProbability`** metric of an **`OfferedInterface`** is calculated as the probability that the **`OfferedInterface`** is not operational, based on the respective failure propagation structure, and assuming an availability *A* for each **`BusinessApplication`** (i.e. it is assumed that an business application fails with probability *1-A*, leading to all its **`OfferedInterfaces`** being not operational):

$$\text{failureProbability(oi)} = \sum_{e \in \{b \in IB^{AppNr} | F_{oi}(b) = false\}} A^{|\{i|e_i = true\}|}(1-A)^{AppNr - |\{i|e_i = true\}|}$$

Thereby, B^{AppNr} is a set of vectors, which have a Boolean value for every **`BusinessApplication`**, characterizing whether the application has failed or not. $F_{oi}(b)$ indicates, whether interface *oi* is operational given that business applications have failed as indicated by *b*. Thus, $F_{oi}(b)$ relies on the failure propagation structures.

The **`failureExtent`** of a **`BusinessApplicaton`** *b* calculates the deterioration of the **`failureProbability`** of all **`OfferedInterfaces`** in case *b* has failed. The metric value is the average over all these values.

It has to be noted, that above metrics do not take into consideration a more fine-grained structure of the business processes. If such information is available, the approach could be adapted to consider this information. However, it may be more adequate to use information from metrics as above as input into another technique more focused on the process level, e.g. using cause-and-effect diagrams [Is90] to analyze problems in a process.

2.3 Viewpoint

Picking up the distinction between inter- and intra-application aspects of failure propagation, two different viewpoints for these aspects can be applied. Examples for these viewpoints have already been given in Figure 1 and Figure 2. Complementing the visual examples, here some more details on the makeup of these viewpoints are given. The *application landscape viewpoint* on failure propagation (as exemplified in Figure 1) centers around one or more business processes of interest, which are shown together with the application landscape supporting them. Additionally, the visualizations of the offered interfaces or the business applications can be annotated with additional information from metrics i.e. derived attributes in the above information model. Figure 5 exemplifies such an annotation, by indicating the **`failureProbability`** metric value of an **`OfferedInterface`** as well as the **`failureExtent`** of a **`BusinessApplication`** by color coding. Therein, *red* indicates high, *yellow* medium, and *green* low values of the respective derived metric attribute.

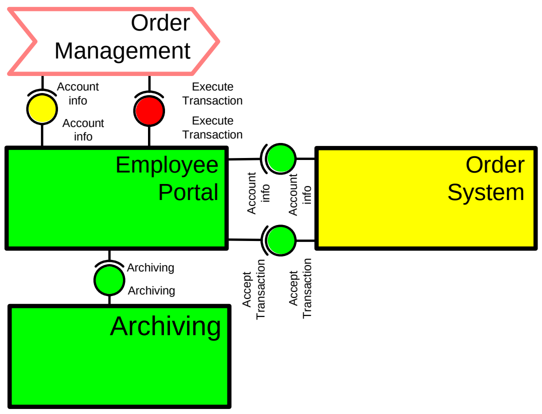

Figure 5: Application landscape view with visualized metrics values

The *intra-application viewpoint* on failure propagation (as exemplified in Figure 2) focuses on the failure trees of a specific business application. Every interface offered by the respective application is thereby put in relation to the interface usages of the application. By this the causes for a distinct interface to be non operational are explicitly modeled. Another viewpoint for metrics information is subsequently presented in the case study (see Section 3).

2.4 Methodology

Above metrics and viewpoints are designed as instruments in evaluating and discussing the tendency of an application landscape to failure propagation. Subsequently, we outline, how to use these instruments to address concerns as "examine to what extent the application landscape structure is prone to failure propagation". Therefore, the usage of the metrics introduced is described, focussing on how to make sure that the assumptions, under which they can be used as an indicator of failure propagation, actually apply.

A considerable part of the effort in metrics utilization occurs in the introduction of the metrics. Aspects of change management, as sparking interest and getting approval of the respective stakeholders, belong to the organizational part of this effort. Thereby, it should also be surveyed, in which use cases the metrics are to be employed.

The methodical part of introducing metrics lies in getting and interpreting the necessary data. Thereby, it has to be distinguished, whether data already available, e.g. in repositories already maintained for Enterprise Architecture Management, is to be used for metrics calculation, or the data is yet to be collected.

If already otherwise collected and maintained data is to be used, it may not be expected that it has been collected in a structure and according to definitions as presented with the information model in Section 2.2. However, as this information model contains rather basic concepts of an application landscape, it might very well be possible that at least some of them are present in the data, e.g. the (deployed) business applications. Other kinds of information, as e.g. relating to the internal failure propagation structures, might be less readily found in existing repositories. If it is not feasible to collect this information, rather general assumptions might be used, e.g. that all offered interfaces of a business application fail, if at least one of the interfaces used by the application fails. Thereby, it should be discussed, whether the assumptions are valid and useful in addressing the respective concern, and explore possibilities for this being not the case.

If data is yet to be collected, getting the support of stakeholders becomes even more important, but the definitions from Section 2.2 can most probably be directly used. However, also here, collection effort might lead to using assumptions instead of more elaborate data collection.

In all cases, steps for validating the model have to be taken, in order to ensure that it is actually able to measure failure propagation tendencies in the respective use case. A very straightforward approach therefore is to contrast the results from the metrics with actual availability indicators collected in the operation of the application landscape. This can be done graphically and intuitively via an appropriate software map, or statistically, e.g. by trying to predict actual availability values from the **`failureProbability`**-metric values via regression. Assumptions about the failure propagation within business applications could be confirmed or refuted by source code analysis, or by analyzing log data about failures, if available. If assumptions have been made instead of actually collecting data, this data could be completed in cases, where a validation reveals the assumptions to be problematic.

Finding the right value for the constant *A* (the assumed availability of each business application) can be seen as a related problem. It should be set to a plausible value, to improve the interpretation of the metrics. However, the importance of setting *A* should not be overrated, as it is a rather arbitrary value, affecting more the absolute metrics values, but not their relations, which can be considered as most important in interpreting the metrics. However, it is important to leave *A* the same, once it has been set, in order to keep the metrics values comparable and focussed on the application landscape.

Use cases, in which the metrics could be employed, include:

- Understanding specific structures in the application landscape and their effects on failure propagation and thus availability of the offered services
- Comparing scenarios regarding their tendency towards failure propagation
- Depicting scenarios and their benefits, without focusing on technical details
- Setting requirements for the evolution of the application landscape in respect to failure propagation, without prescribing the actual realization strategy

3 Case study

The metrics based approach for evaluating failure propagation tendencies in application landscapes is currently tried in a practical use case at a large Swiss bank. Thereby, the metrics have been applied to a subset of the application landscape supporting private banking for Switzerland, concentrating on the applications running on the mainframe platform.

This subset of the application landscape consists of 255 applications, organized into 75 subdomains, which are themselves organized into 18 domains. Together, the applications amount to about 12 millions lines of PL/1 code. These applications rely on each other by extensive dependencies of different kinds:

- Synchronous dependencies, which are simple function calls
- Messaging, i.e. asynchronous information exchange via message queues
- Bulk/file exchange, where an application can send a preferably larger amount of information to one or more other applications in a file
- Database facilitated dependencies, with two or more applications accessing a shared database

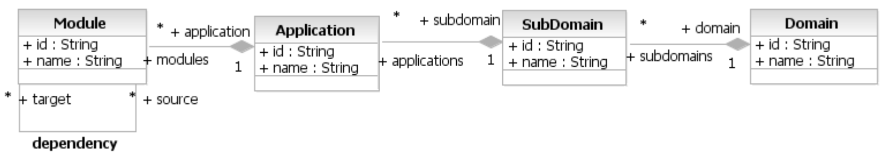

Figure 6: Structure of the available information

The above described subset of the application landscape was used for a case study with the metrics described above, as a large dataset describing it was available, created from automatically parsing the code deployed on the mainframe. Figure 6 shows the structure of the information in this dataset, as it is available to the case study.

The case study is conducted in an iterative way, to be able to accommodate both findings about the application of the newly developed metrics, and concretizations of the concerns to be addressed by using the metrics. The first iteration of the case study was started with the concern being to address “operational independence”, with failure propagation being one aspect thereof, and the one on which this paper focuses.

In the first iteration, several interpretations of the available data where tried in order to apply the failure propagation metrics from Section 2. One interpretation consisted of taking the “Applications” in the available data as the **`BusinessApplications`**. Thereby, **`OfferedInterfaces`** where not explicitly considered, however, it was assumed that all services rendered by an application fail (leading to the application failing totally) if at least one of the applications it depends on fails. However, these assumptions turned out to be overly conservative in respect to how failures are propagated. In spite of setting *A* to rather optimistic values, the metrics values were unrealistically pessimistic.

The second interpretation took the modules as **`BusinessApplications`**, offering one kind of service (via a respective interface), and failing if at least one of the modules it depends on fails. However, also this interpretation had some difficulties. First of all, it yielded only reasonable results, if the module-specific *A* was set rather high. Moreover, we view it more reasonable to consider applications as the units having the availability, and not the units in which their source code is organized.

Thus, a more sophisticated interpretation was used, which relies on interpreting the “Applications” from the data as **`BusinessApplications`**, and examining the module structure, in order to derive information about **`OfferedInterfaces`** and **`InterfaceUsages`**.

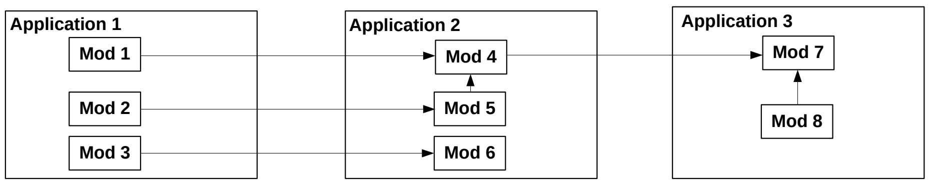

Figure 7: Utilizing the module structure

In Figure 7, an arrow signifies a function call. Thereby, the calling module is at the end of the arrow, the called module at the arrowhead. Modules called by modules from other applications are considered as **`OfferedInterfaces`** (“interface modules”). Such an “interface” is assumed to fail if the application, which offers the interface, fails itself, or an application, of which the “interface module” uses (also transitively) a module, fails.

Figure 8 shows an example of a software map as it has been employed to visualize the metrics data. Thereby, grey rectangles symbolize domains. The subdomains of a domain are represented by white rectangles nested within the respective domain-rectangle. In a similar way, each subdomain-rectangle indicates the subdomain’s applications via colored rectangles. Size and color of the application-rectangles thereby represent two metrics: The higher the **`failureProbability`**-metric of an application (aggregated via the applications interface modules by a simple mean), the bigger the respective rectangle. The **`failureExtent`**-metric determines the rectangles color, with lower metric values leading to greener, higher values to more yellow colors.

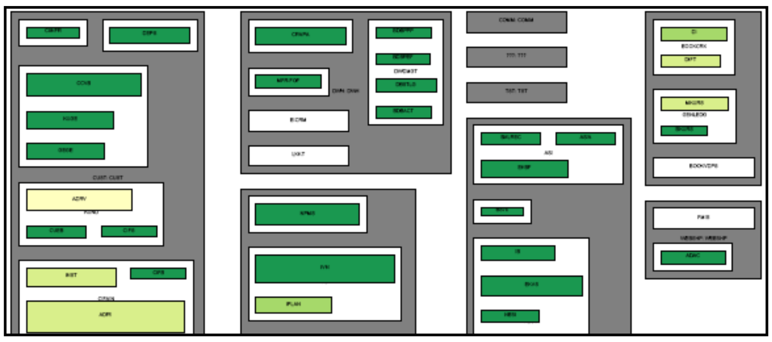
Figure 8: Exemplary Software Map (falsified data)

After presenting the approach to stakeholders, it was decided to employ it for quantitatively comparing the as-is situation to two possible scenarios of the application landscape, which have been designed to improve the operational independence in the application landscape. This evaluation is currently performed. The goal is to evaluate, how the metrics can contribute to decisions, and how domain experts judge their utility.

4 Outlook

While we view above metrics as a valuable instrument in assessing tendencies to failure propagation in application landscapes, which we plan to further evaluate via case studies, they measure merely one in a set of interesting aspects about an application landscape. Of course, we do not consider metrics as the only instrument for assessing application landscapes, with scenarios as used in software engineering e.g. being an alternative. However, we see certainly potential for research into metrics for other aspects. An aspect we thereby look into is modifiability, i.e. to what extent the structure of the application landscape makes changes in the landscape's applications (e.g. in the context of maintenance) more difficult to perform. One approach we consider thereby is looking into influencing factors on productivity used by effort estimation techniques as Function Point, to predict them from information about the structure of the application landscape.

A basic prerequisite for using metrics, and thus for evaluating them in case studies, is a tool allowing automated metric calculation. As we also view adequate metrics visualization as an enabler for metrics utilization, we align the metrics research with the development of our SoCaTool [Bu07b], a prototypical tool for automated generation of application landscape visualizations. Thus, we are setting a tool-specific stage for research into how metrics can improve the management of application landscapes.

Bibliographical References

[Bu07a] Buckl, S. et al.: A Pattern based Approach for constructing Enterprise Architecture Management Information Models. In 8. Internationale Tagung Wirtschaftsinformatik, Karlsruhe, 2007.

[Bu07b] Buckl, S. et al.: Generating Visualizations of Enterprise Architectures using Model Transformations. In: 2nd International Workshop on Enterprise Modelling and Informations Systems Architectures - Concepts and Applications, St. Goar/Rhine, Germany, 2007.

[BW05] Braun, C.; Winter, R.: A Comprehensive Enterprise Architecture Metamodel and Its Implementation Using a Metamodeling Platform. EMISA 2005, pp. 64 - 79.

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

[Fr02] Frank, U.: Multi-Perspective Enterprise Modeling (MEMO) – Conceptual Framework and Modeling Languages. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences 35 (2002), pp. 1258-1267.

[Ge98] Gertsbakh, I.B.: Statistical Reliability Theory. Marcel Dekker, Inc, New York, 1989.

[Is90] Ishikawa, K.: Introduction to Quality Control. Chapman&Hall, London, 1990.

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

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

[La05] Lankhorst, M.: Enterprise Architecture at Work: Modeling, Communication and Analysis. Springer, Berlin, Heidelberg, New York, 2005.

[LW04] Langenberg, K.; Wegmann, A.: Enterprise Architecture: What Aspects is Current Research Targeting? EPFL Technical Report IC/2004/77.

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