Defining Metrics for Conceptual Schema Evolution

Lex Wedemeijer

ABP Pensioenen, The Netherlands
L.Wedemeijer@wxs.nl

Abstract. It is generally believed that a well-designed Conceptual Schema will remain stable over time. However, current literature rarely addresses how such stability should be observed and measured in the operational business environment with evolving information needs and database structures. This paper sets up a framework for stability of conceptual schemas and proceeds to develop a set of metrics from it. The metrics are based on straightforward measurements of conceptual features. The validity of the set of metrics is argued here from theory, operational validity may be demonstrated by a longitudinal case study into the evolution of conceptual schemas. The main contribution of this paper is the realization that the measurement of conceptual schema stability is an essential step for understanding and improving current theories and best-practices for designing high-quality schemas that will stand the test of time.

1 Introduction

According to the 3-schema architecture, a well-designed Conceptual Schema (CS) satisfies many quality requirements [5,30,35]. It is the task of the designer to meet these requirements in the best possible way. In particular, the CS is required to be stable enough to support a long-term systems lifetime and be flexible enough to meet future information demands. Many design strategies exist that claim to improve the flexibility of the CS design. Why they should enhance flexibility is often explained, sometimes demonstrated, but rarely proven by actual business cases. A designer that wants to prepare the CS for future changes, must trust to experience and to state-of-the-art design practices, there is no way to pick the 'best' design strategy for a particular business case at hand. Also, a cry for wholesale flexibility of CSs is not a very specific requirement that designers can meet with:

- flexibility can only be established 'on the fly'. A potential for change can only become apparent when a structural change occurs, and not when discussing a new schema
- there is no distinction between structural changes that ought to be accommodated by the flexibility in the CS design, and those that are beyond the desired flexibility, and

– there is no way to verify that a CS has ‘enough’ flexibility, or to discover that ‘more’ flexibility is needed.

It is clear that the notion of flexibility is too general and unspecific to be of value in assessing the quality of a CS design, and does not contribute to an understanding of the evolution of the CS. The main problems with the concept of flexibility are both in the dependence on future events, and its lack of specificity. What is needed is sound criteria that can be measured and researched by looking at the actual schema evolution as changes occur over the operational lifetime, and that can be used to improve current best-practices for CS stability and flexibility. The central goal of this paper is to propose such a set of metrics. We do not claim that the proposed set of metrics is exhaustive but, to the best of our knowledge, the comprehensive set of metrics for schema evolution as defined in this paper have not been reported before in the literature.

The paper is organized as follows. Section 2 introduces the general framework for stability. Section 3 derives the principal requirements for stability and proposes suitable metrics. Section 4 argues the validity of the set of metrics. Section 5 discusses how these metrics can be applied in a field study of schema stability. Section 6 looks at some related work. Section 7 draws conclusions and outlines directions for further research.

2 The Framework

We assume the reader is familiar with the traditional 3-schema architecture [3] (Figure 1). Our interest is in the CS being the single best way of perceiving the Universe of Discourse (UoD), not only at design time but as they both evolve over time. It is in their joint evolution that the CS must demonstrate its stability and flexibility.

Intuitively, flexibility means adaptability, responsiveness to future changes in the environment. And ‘more’ flexibility will mean a smaller impact of change. Stability covers much of the same ground but where flexibility refers to a future capacity for change, stability refers to the past, being evidence that any required changes have been accommodated and that flexibility has been delivered. This leads us to conclude that flexibility and stability share the following three ‘dimensions’ that are orthogonal to each other:

– an environment where changes originate, namely the Universe of Discourse,
– time required to adapt, i.e. the time needed to propagate changes to the other components of the information system, and
– the potential to adapt.

These three dimensions are further refined into a number of high-level mechanisms and best-practices that aid the designer in enhancing the future flexibility of a CS. These mechanisms refine the framework as shown in Figure 2. The large number of mechanisms and their wide variations in scope may possibly explain why there is as yet no generally accepted and unambiguous definition of the concept of schema stability.

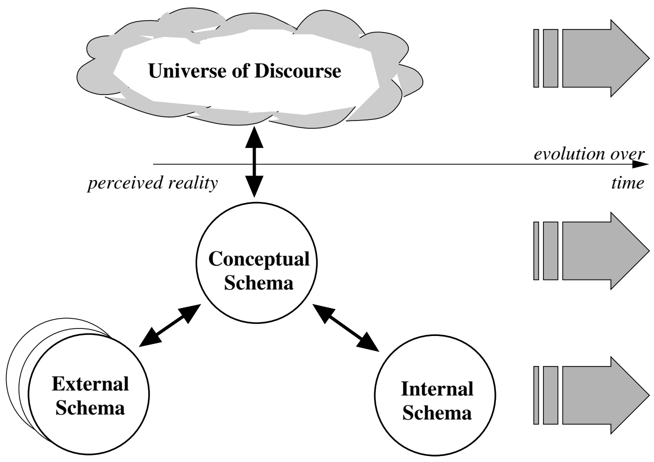

Fig. 1. 3-Schema Architecture

This framework provides a sound basis to evaluate overall flexibility of CSs. It is built on the 3-Schema Architecture and establishes a clear cause-and-effect relationship between "structural changes" in the UoD and those in the CS. It restricts the relevant environment from which changes stem to the Universe of Discourse, and no more. This prevents inappropriate demands of flexibility on the CS. For instance, it excludes changes in responsibilities and tasks of business unit management, changes in the database management system, in the design methodology or duties of the maintenance team etc. An important feature of the framework is that it can be used not only to understand flexibility as a potential for future change. It also provides us with a yardstick to measure to what extend the CS flexibility has actually been exploited in the past. The next section explains the importance of the past evolution of the CS in this respect.

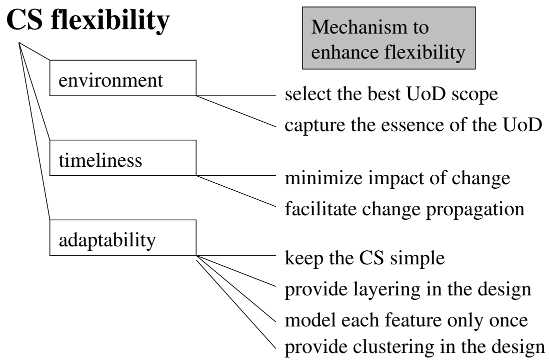
CS flexibility
Mechanism to enhance flexibility
environment
select the best UoD scope
capture the essence of the UoD
timeliness
minimize impact of change
facilitate change propagation
adaptability
keep the CS simple
provide layering in the design
model each feature only once
provide clustering in the design

Fig. 2. Conceptual Schema framework for flexibility and stability

This paper is about evolution of the CS over its operational life time. It does not address the issue of quality of the CS as output from the initial life cycle phase of schema design, be it delivered in the traditional 'waterfall' method or by some iterative approach. For the same reason, we assume that a single data model theory is used. The data model theory defines the constructs and constructions of the CS, and any change in the data model theory must propagate down to the CS [13]. As a result, changes in the CS are precipitated that are not driven by new user requirements (but by the data management department).

Many design strategies exist that claim to deliver high-quality, flexible CS designs. To name some important ones:

- schema transformations approach [16]
- reflective approaches [40,53]
- global schema integration [3,46]
- component-based development, or: (re)use of schema patterns [10] and
- ontological approaches [49,54]

Why any of these particular strategies should enhance the future flexibility of a CS design is often argumented, but the literature is very scarce on actual proof of flexibility in live business cases. While there is no real understanding how these strategies succeed in delivering flexibility, we do not intend to research this issue. The aim of this paper is to understand the mechanisms that are involved in exploiting flexibility as a potential for change.

The life cycle phase of testing, when an unfinished CS is being completed, is also beyond our scope. It is quite common for this phase that many changes occur: be it correction of design failures, or enhancement of initial design quality. But the need for adjustments in this stage indicates progress in the understanding of requirements and improvements in the way of incorporating them into the design. We feel that the amount and types of changes in this phase is a hallmark of the designer's ability and experience rather than an expression of real changes in the UoD. Some interesting research in this area has been done by [7,8], but not exploring the consequences in the operational life cycle phase.

3 Metrics for Conceptual Schema Evolution

The general framework serves to develop hypotheses on how schema stability ought to be expressed in operational environments. With each hypothesis we associate a metric that may be used to test the hypothesis for evolving conceptual schemas in operational businesses. A metric can be defined as 'a function whose inputs are elementary measurements of an IT-artifact, and whose output is a single value that can be interpreted as the degree to which the artifact possesses a given property or satisfies a given hypothesis' [45]. Each of our metrics produces objective (i.e. repeatable) outcomes, and shows the desired tendency for the associated hypothesis.

3.1 Justified Change

By definition, a CS is a complete and correct model of the information structure of the UoD, and nothing else. As long as the business activities of the organization remain unchanged, the information needs remain the same. It follows that a change in the CS is only justified if a change in the UoD information structure is causing it. Any change in the CS that cannot be linked with some driving cause in the UoD is by definition an unjustified change or instability. For instance, the CS should be indifferent to technical changes: increasing transaction volumes, more efficient data fragmentation plans, installation of additional infrastructure etc. So our first demand that must hold in quality CSs is:

Hypothesis:	*every change in the CS is justified*

To establish whether a change is justified, we need to

– determine every single CS change, and
– associate each one with the appropriate change driver(s) from the UoD.

The metric for justified change is the ratio of single CS changes that can be associated with an appropriate change driver, over the total number of CS changes (either with, or without change driver). Ideally, the ratio is equal to 1.

The metric is sensitive to the definition of 'single CS change'. Usually, the 'single changes' are identified with elementary, i.e. indecomposable changes as defined in the data model's taxonomy [6,21]. But care must be taken because many taxonomies consider only transformation of a single construct or construction at a time, while the actual semantics may be a single, coherent change in several schema constructs at once. For instance, dissolving a generalization [55] involves deleting the generalized entity, removing the associated is-a relationships, plus moving all aggregation relationships that the generalization was involved in.

The metric is also sensitive to the demarcation of the UoD. Selecting the right scope for the UoD is an important topic in design and will receive much attention. But once the design phase is finished, the scope of the UoD is fixed. After that, the CS is presumed to be the complete and correct model of the UoD information structure and vice versa: the UoD is that which is modelled by the CS. Consider for instance an enterprise that operates an integrated customer database. To change its CS in order to model which regional offices manage a fragment of the customer database is unjustified because the internal organization of the enterprise has not been included in the UoD. It is suggested in [32] to distinguish between change drivers that are external to the enterprise ("a more stable external environment enhances stability") and those arising from somewhere within the own organization ("a more simple internal environment enhances stability").

3.2 Proportional Change

In physics, the property of stability is defined for a system in (near) equilibrium as: any disturbance in the system's state will cause a reaction that is proportional to the size of the disturbance. Analogously, we want a small change in the UoD to cause a proportionally small change in the CS, assuming the change is justified. To wit: it is not uncommon that a relatively small change in the UoD triggers an avalanche of changes in the CS. Such a situation deserves to be called an instability, and we want our metrics to single it out as such. So we conjecture

Hypothesis: *every change in the CS will be proportional to the change in the UoD that causes it*

To establish whether a change is proportional to the change driver, we need to measure:

- the size of the change in the CS, and
- the severity of the change driver in the UoD

The metric for proportional change is established as the ratio of size of CS change over the severity of UoD change. Ideally, the ratio should have a low upper bound.

There is a problem here in observing the 'size' or 'severity' of the single change in the UoD. This concept cannot be formalized rigorously, for the same reason that 'the information structure of the UoD' cannot be formalized without referring to some kind of conceptual representation. It is blatantly incorrect to let the maintenance engineer decide on this: the severity of the UoD change will then of course be judged by its impact on the CS! Nevertheless, an operational measure of size could be the number of paragraphs explaining what has changed in the UoD.

In contrast, the size of CS change is easily determined as the number of affected constructs. Depending on the data model theory the size count can be further refined into counts by type such as entity, attribute, constraint etc.

3.3 Proportional Rate of Change

Likewise, it can be said that a system constantly undergoing some kind of change is not very stable. An operational CS which is meant to support many user applications, must have an acceptable low rate of change. But what rates are acceptable, what is not low enough? Users will generally relate it to the business environment that is being modelled. A very turbulent environment changes frequently, and users will accept a correspondingly high rate of change for the CS that models it. That same rate will probably not be accepted in a stable environment, such as a company engaged in the growing of a forest.

Too high a rate is an unstable system, and users and management will not tolerate this for long. On the other hand, a CS with a very low rate of change may not keep abreast with changing business requirements, and might actually be too rigid to change at all. This holds for fragile legacy systems where any change might precipitate an avalanche of unexpected side effects. So we have:

Hypothesis: *the rate of change in the CS will be proportional to rate of change in the UoD*

First, one has to measure the rate of change in the CS. This derives from two measurements:

- the difference between old and new CS, i.e. the number of changes made in creating the new CS version, and
- the lifetime of the CS versions, i.e. elapsed time between subsequent versions going operational.

The rate-of-change is then calculated as the ratio of the number of differences over the version lifetimes. The CS stability expressed in this rate of change improves over time if either the lifetimes of CS versions increase—but this may also reflect rigidity—or if the number of changes between versions decreases.

Next, a measure for rate of change in the UoD must be devised that is targeted at changes in information structure. We are not concerned with changes in information, that are handled by ordinary transactions and data updates. In a similar fashion as above, we propose a rate of UoD measurement to be the ratio of two numbers:

- the difference between old and new user requirements, i.e. the number of changes made in the requirements deriving from the UoD, and
- the lifetime of the consecutive sets of user requirements.

The turbulence in the UoD can then be expressed as the ratio of the number of changes in requirements, over the lifetime of requirements. Of course this is a somewhat hypothetical measurement. When confronted with real business situations, it will be next to impossible to come up with an exact and verifyable 'count' of differences in requirements. An alternative is to count the number of change drivers, as discussed above in the metric for justified change.

The metric for proportional rate-of-change is established as the ratio of both measurements: rate-of-change of the CS over rate-of-change in the UoD. Ideally, the ratio should have a low upper bound. A first approximation is to set the lifetime of user requirements equal to the lifetime of the CS versions, making it cancel out of the equation. The metric simplifies to the ratio of:

- the difference between old and new CS, i.e. the number of changes made in creating the new CS version, and
- the difference between old and new user requirements, i.e. the number of changes made in the requirements deriving from the UoD.

The rate-of-change measurement per CS can be used to benchmark CSs that cover a similar UoD. The CS with lowest rate of change is best, because it will incur lowest cost and least interruption of service to customers. A similar metric was employed by [9] in their study of the evolution of software programs.

There is a caveat, because the rate-of-change measurement is biased. It will appear to be better for small CSs than for highly integrated CSs. If the UoD is larger, then more features of the UoD can change, so the rate of change in the CS will probably be higher. Imagine to cut up a large and complex CS in two parts: the versions for each part can be expected to have half as much changes, and a twice as long lifetime, so the overall rate of change is 4 times as low. The hypothesis should not be misinterpreted as an advise to fragment large CSs. Other features of a non-conceptual nature may also influence the rate of change: the capacity of the maintenance department. The rate will be significantly lower if the department is understaffed.

This metric is not always applicable. A fundamental assumption is that the entire CS is versioned [43]. Some approaches use other versioning mechanisms, e.g. O-O data modeling theories allow versioning per construct [2]. In such a case the hypothesis may still hold, but the metric, and some others to follow, will not work and another one is needed.

3.4 Compatibility

Compatibility aims to ease change. Compatibility, the demand to keep the impact of change as small as possible, is a natural drive towards stability. It will ease schema evolution because the need for complex data conversions is intensionally minimized.

We define a new CS to be compatible with the old one, if no data present in any construct of the old CS needs to be altered or discarded to fit the new schema. As compatibility will considerably lower overall cost, time and effort of change, designers will go out of their way to achieve it. As a result, a CS change may be compatible, but other quality aspects may be compromised. So we conjecture:

Hypothesis: *the rule is compatible change, the exception is incompatibility at specific places in the schema*

To establish at what locations a CS change is incompatible, we must look at the general pattern of changes in data instances, and ignore for the time being changes in schema constructs. The data that needs attention must be separated from the data that can be left unchanged. By definition, the set of data to be edited is a temporary External View on the old CS. A measure of compatibility for CS change can be based on the relative size of that External View, so we count per type of construct:

– the number of constructs in the 'data-to-be-edited' External View, and
– the number of constructs in the old CS

The level of compatibility is then calculated as 1—the ratio of these two counts. Or, equivalently, it is calculated as the number of constructs in the old CS not affected by the change divided by the total number of constructs in the old CS. Ideally, the ratio is equal to 1, when all the data instances of the old schema fit seamlessly in the new schema of things.

Whereas the previous rate-of-change metric was found to be biased towards small CSs, this compatibility metric is biased towards large CSs. If the same change is accomodated in two different CSs in the same way, then the metric produces a more favourable outcome for the larger one.

Compatibility is closely related to the concepts of logical and physical data independence [3,14]. A methodical way for improving compatibility is developed by [26], but their approach is limited to changes in a single entity (or rather, relation).

Incompatibility is when data instances have to be edited, moved wholly or partially from one entity into another, when relationships have to be reestablished etc. It requires the editing of data instances beyond the scope of either the old and the new CS. Such data conversion efforts are not uncommon in business situations, but are rarely accounted for in the literature [31,33].

A form of incompatibility that is even harder to accommodate is when the level of abstraction changes, causing differences between schemas that are known as semantic discrepancies [47]. A methodical approach that supports the detection and prevention of such incompatibilities in schema evolution is found in [56].

3.5 Extensibility

New ways of doing business are generally supposed to augment existing business procedures and methods, not to replace them. It follows that when information requirements change, the new requirements are additional to what is already accounted for in the old CS. The most obvious changes of this kind are additions of new constructs to the CS.

A type of change that often goes unnoticed is extension of the entity definition. While the entity name and composition are not changed, it is fundamentally altered. This is because the intent is broadened, so many more data instances can and will be recorded for it. An example is when the definition of 'person' is first restricted to customers only, whereas after extension it also covers their spouses. A consequence of extension is that the old CS becomes a valid External View on the new CS, preferably an updateable one so that old update routines can remain unchanged. On the data level, change by extension leaves the old data fully compatible with the new schema, as discussed above. This line of reasoning leads us to formulate:

Hypothesis: *the rule is schema extension, the exception is modification of existing constructs*

To establish whether a change in the CS is an extension, we take the metric for compatibility and refine it. For each type of construct in the new CS we count:

– the number of pure additions, and
– the number of constructs in the new CS that differ from the old CS in any way at all

The metric for extension is established as the ratio of the first over the second count. Ideally, the ratio equals 1 meaning that there are only additions and no other changes.

The metric is insensitive to the deletion of constructs, because a deleted construct does not show up in either count. This is unfortunate because a CS change may appear to be a pure extension while actually, the new construct is a variant of some construction that is deleted simultaneously.

Taxonomies are often based on the idea that change in a construct is a simple concatenation of construct deletion plus construct addition [23]. However, this does not hold at the level of data instances because data will be lost as soon as a construct is deleted. If users are aware that a new construct is actually a variant of something old, then they will demand data compatibility to safeguard their data assets, i.e. that old data instances must be carried over into instances of the new schema. Lossless transformations [16] are introduced into taxonomies to guarantee that any relevant data instances are retained. Therefore the application of metrics for extension and compatibility depends very much on the choice of taxonomy.

3.6 Complexity Hampers Change

It is generally agreed that complexity is a main determinant in maintenance of any product, be it hardware, software, or a conceptual schema [4,15].

As businesses depend more and more on information systems, and as most changes to information systems augment the support for the business operation, it can be expected that the overall size and complexity of information systems will increase.

Surprisingly, the concept of complexity is often discussed only intuitively, for instance [17] introduce their concept of complex object type as 'simply a boundary line drawn around a set of objects and relationships in the schema' (p.425).

The usual feeling is that complexity has to do with the combined effect of both a large number of things, and the coupling/interdependence between them, the result being a difficulty to understand the entire setup. The complexity of the composite system is then determined by the number of components, the number of ways in which the components are interrelated, and how these may change over time.

Authors point out that the complexity of a system has a negative impact on its overall quality. As stated by [22] '"the more relationships the less comprehension" is possibly due to the accompanying increase in complexity.' (p.348). We are not interested in complexity as such, but in the effect of complexity of a CS and its constructs on the overall stability. The general idea is that as complexity of a CS is greater, change is more difficult. The maintenance engineer will generally avoid to mess with complex structures, so we conjecture:

Hypothesis:	*a more complex CS will change less frequently*

A metric for this hypothesis requires measures for the notions of schema complexity and frequency of change. So if we can decide on objective measures for

- the complexity of each CS version, and
- the lifetime of each CS version

then their ratio is a first characterization for this hypothesis, assuming a linear dependence between the two. Next, the hypothesis should be tested by comparing these ratios for a number of CSs with equal and/or different complexities.

A prerequisite in this hypothesis is the objective indicator of CS complexity—which is hard to find. Size is a first indicator of schema complexity but, as has been observed by [37] ‘this assessment of complexity ignores the number of relationships, named and unnamed, in a given model’ (p.41). Moreover, complexity of a CS is not only dependent on the information structure of the UoD alone. Other factors are of perhaps greater importance, such as ease of use of the data model theory, capabilities of the designer, restrictions due to demand for compatibility etc. [28], when researching software complexity, finds that ‘surprisingly, much of the observed complexity appears to be technically unnecessary (and) excessive schedule pressure and hasty design tend to be a common root cause’ (p.100).

Considering the many aspects that contribute to complexity, it can be doubted that a single number suffices to express overall complexity. For instance, complexity of a CS is very much dependent on the chosen data model theory. Influencing factors are the kinds and levels of abstraction of data model constructs and constructions, the ease of use for maintenance engineers etc. We will briefly discuss two measures for complexity, and their consequences for our metric of change.

A simple measure for complexity of the aggregate mechanism may be obtained by regarding the CS as a lattice where each node is an entity and each edge represents an aggregation relationship. In a more complex lattice, the number of edges (i.e. relationships) will exceed the number of nodes (entities), and integrity constraints are required to ensure overall data consistency. In measuring the complexity of any lattice of a certain size, we need to consider what the ‘minimal’ complexity will be and set this to 0. We also need to account for the fact that some CSs are actually not a single lattice, but are made up of several unconnected subschemas. Our cyclomatic complexity metric for CSs is calculated as:

$$\begin{aligned} &\text{number of unconnected lattices (subschemas)} \\ -\ &\text{number of nodes (entities)} \\ +\ &\text{number of edges (relationships)} \end{aligned}$$

A simple lattice like two entities connected by a single relationship has a cyclomatic complexity of 0. Slightly more complex is a lattice of three entities that are all connected, with a cyclomatic complexity of 1. This number has a sound interpretation: it means that 1 constraint may suffice to guarantee referential integrity in the lattice.

Our complexity metric is not new. McCabe’s measure of cyclomatic complexity for software code follows the same line of reasoning; it can even be retraced to the mathematician Euler (1707–1783). [24] apply McCabe’s software metric in 7 case studies in the US Department of Defense. Their findings ‘suggest that maintenance productivity declines with increasing complexity density’ (p.1287), which agrees with our hypothesis. However, closer inspection reveals that the suggestion actually derives from a single outlier point in their small set of case studies, so the argument is not really strong.

Just like aggregation, the mechanism of generalization can be a cause of complexity. It was noted by [18] how class hierarchies may come to be used inconsistently due to misunderstanding the overall structure of generalizations and specializations.

A measure for complexity for the generalization mechanism is obtained along similar lines. The generalized entity is devolved into a lattice with specializations being nodes, and edges representing the generalization / specialization relationship. Attempts at understanding and clarifying this lattice structure have been described in [11,27,29].

3.7 Abstraction Reduces the Need for Change

The notion of CS abstraction is markedly similar to that of complexity. Like complexity, the level of abstraction is an important design consideration. [15] states that the stability of a CS depends on its level of abstraction. The general idea is that a more abstract design will have a better stability. This is because a less abstract, thus more detailed CS has more constructs and constructions that need to be changed in order to adapt equally well to new requirements. So we conjecture:

Hypothesis: *a more abstract CS will go through less changes*

A metric for this hypothesis should include

- the level of abstraction of the CS, and
- the number of constructs in the CS that change over time

Their ratio is a first characterization for the hypothesis, assuming a linear dependence between the two. In order to test the hypothesis, ratios should be compared for a number of CSs.

Like complexity, the metric for abstraction ought to build on a generally accepted and well-defined measure of abstraction, which again is found to be lacking. It is beyond the scope of this paper to suggest a solution to this issue, but a few remarks are in order. First, it is evident that a CS with a higher level of abstraction should have less constructs with more instances; while a lower level of abstraction results in a CS with more constructs with fewer data instances. Second, abstraction in the CS is strongly related to the data model theory that is used. Some data models (e.g. those based on ontological principles [54]) are considered to be more abstract than others. Third, CS designs are often documented on multiple levels of abstraction [38], and the metric ought to show consistently better outcomes on the higher levels. Finally, it must be noticed that the terms abstraction and clustering (aggregation) are sometimes used interchangeably [22].

3.8 Susceptibility to Change

It is a common assumption that attributes of an entity will change more frequently than the entity as a whole, and descriptive attributes will change more often than primary-key attributes; [1] uses the term ‘sensitivity’. [57] argues that: ‘it is likely that “rules” set by management or other political bodies will change more frequently and quickly than inherent properties, and that rule changes will more frequently affect relationships among entities than the related entities themselves’ (p.1241). In other words, some types of constructs provided by data model theories are presumably more stable than others. Many designers exploit this by doing CS design following the straightforward top-down approach, perhaps calling it abstract-to-concrete. Entities and relationships are presumed to have best stability and hence are modeled first. Attributes and relationship cardinalities are assigned later on, while integrity constraints and business rules are the most volatile and are added to the schema as late as possible. So we conjecture:

Hypothesis: *some types of construct in the CS are more susceptible to change*

Obviously, metrics for this hypothesis must differentiate between the types of construct. A simple measurement will include:

- the various types of construct as provided by the data model theory,
- the total number of constructs per type that is present in the CS, and
- the number of constructs per type that change (perhaps refined by including the type of change, i.e. addition, alteration, or deletion)

The susceptibility to change per type of construct is calculated as the ratio of the number of changed constructs, over their total number in the CS. These ratios can then be compared between types. It seems reasonable to expect that the ratio will be low for entities, while constraints will have a high ratio, meaning they are very susceptible to change. The ratios can also be compared among different CSs.

The hypothesis implicitly assumes that UoD features that are modeled with one type of construct at one time, will be modeled with the same type of constructs at all times. That is, type persistence is assumed [31], while [33] assume a type compatibility invariant when changing a CS. [42] argument that: ‘an object type may not evolve into a method, and a constraint may not evolve into an instance’ (p.357). Some authors concede that a construct might change its type, e.g. in object-orientation [34], but this is not covered by our metric.

We already pointed out that there is no intrinsic reason why type persistence should hold. It is up to the maintenance engineer to decide on the best way to represent UoD features in the new CS, and the choice of construct can differ from the one made in the old CS.

3.9 Preservation of Entity Identity

If the CS is drawn up using a relational data model theory, the previous hypothesis can be applied to changes in the important constructs of candidate-key and functional-dependency. This also sheds some light on the preservation of entity identity, because the set of candidate keys provides a sound understanding of the entity identity as they discriminate each instance of the entity from the others. So we conjecture:

Hypothesis: *the rule is no change in candidate key, the exception is change of composition of keys*

The measurements for susceptibility to change apply, but care must be taken to account for composite keys. What needs to be established is per entity the composition of all candidate keys as present in the old and the new CS, and then determine:

– the number of candidate keys that have been changed from the old CS, and
– the total number of candidate keys for each entity in the new CS.

The ratio of keys changed over the total number of keys is an indication of the susceptibility to change of the candidate keys, and thus of the entity identity itself. If keys are stable, then none will change, and the ratio is equal to 0. It is reasonable to expect that this ratio is tightly linked with the susceptibility-to-change metric for the entities, in other words candidate keys will change only if the entity itself is observed to change.

In a live business environment, it may be very hard to establish beyond doubt what constitutes a change of entity identity. For instance, if an Employee table is defined, do we consider the table intention changed if data on temporary help is entered into the table? A careful count is required that detects homonyms, synonyms and other inconspicuous alterations in the composing attributes; and the count must establish beyond doubt whether any one of the candidate keys is affected by such alterations.

3.10 Change Is Local

It is a common assumption that changes in the CS are local, i.e. only a single feature of the CS is affected whenever a single requirement changes. As formulated by [5]: ‘every aspect of the requirements appears only once in the schema’ (p.140), or reversely [30] ‘a random grouping of attributes (lack of cohesiveness) will make the E-R model difficult to maintain; however, the database accuracy is not seriously compromised’ (p.685). This aspect of CS stability is often thought to be the result of good schema design. Normalization is generally regarded to take care of this aspect of stability, although normalization targets at eliminating update anomaly in data instances, not in data structures. The assumption being that in a high-quality CS, a single feature of the UoD is modeled in only a single construction of the CS, we stipulate:

Hypothesis: *a single UoD change will cause change in only a single CS construct or construction*

It is evident what should be measured to establish this localization property:

- identify each single change driver in the UoD, and
- determine the number of constructs in the CS that change as a result

The metric is the ratio of the sum of change drivers over the sum of affected constructs. Ideally, this ratio will be equal to 1. Notice that a single CS construct can be affected by two different UoD changes, if that construct is ‘overloaded’ in the sense that it represents more than one UoD requirement.

There is a close relationship with the metric for justified change, but the difference is in the perspective. Justification looks at the changes in the CS and related them to some UoD change driver. The localization metric takes a single UoD change and locates the constructs in the CS that are impacted.

As in the hypothesis of proportional change, there is a problem here as we need to focus on single UoD change. A fairly objective and easy measure of change drivers might be to count the number of paragraphs in the Change Request form, assuming each paragraph identifies a single need for change. A further restriction is that unjustified changes must be ignored for obvious reasons.

3.11 Change Is Restricted to a Single Module

The above claim that changes in the CS are local is often supplemented with a claim that a modular CS has better stability than a CS without modules. The modules are expected to absorb changes and to isolate other modules from the impact of change; comparable to the property of information hiding in O-O approaches. So we conjecture:

Hypothesis: *a single UoD change will cause change in only a single CS module*

We can use the previous measurements and apply them to establish a metric for this localization property after each module and its exact boundaries has been determined:

- identify each single change driver in the UoD, and
- determine the number of modules where a change is made as a result

The metric is the ratio of the sum of change drivers over the sum of affected modules. Ideally, this ratio will be equal to 1 but it may turn out to be higher.

The literature remains vague on the definition and handling of the ‘module’ construct. There is no outstanding best-practice to determine good modules for a CS. Nor can the ‘goodness’ or ‘optimality’ of the chosen modularization be assessed in a rigorous way. Some methods for choosing modules have been described in [17,39,50]. Size (granularity), complexity and even more so the criteria for clustering are critical issues in determining good modules, but it is rarely explained how the right choice will enhance schema stability. It may be speculated that modularization improves stability by way of the 'time to adapt' dimension, because the impact of any change will be confined to only one or two modules.

The exact boundaries between modules are also important, we feel that it is an 'unjustified change' if some feature of the UoD is modeled first in one module, but shifted into another one later. Our metric may be included in strategic studies to find out which method may be most favorable in a particular business situation to enhance stability of the CS by modularity.

3.12 Modules Are Stable

Once it is decided to decompose a CS into a set of modules, there will be a feeling that each module has 'a life of its own'. That is, each module is the valid and complete model of an isolated part of the UoD, and satisfies all the usual quality requirements such as understandability, correctness, data independence etc. The logical implication is that each module can and will evolve as an independent unit within the CS, and its evolution can be traced over time. So we conjecture:

Hypothesis:	*modules in the CS are stable*

Some authors take the concept of module even so far that the module is redefined as a single entity [52]. It does have an internal structure, but that remains hidden from outside the module. This is a form of information-hiding, which is a familiar concept in O-O approaches. However, we feel that the idea cannot be easily extended to the relational model, because it infringes upon some of the basic axioms on which the relational data model is built.

Notice furthermore that instability of a module does not mean that the CS as a whole is unstable. The rates of change and levels of complexity and abstraction can also vary greatly among modules, this is related to the dynamics of their corresponding UoDs which may vary from extremely slow to very turbulent. The hypothesis actually brings us back to where we started: to understand stability of the CS. Only now the hypothesis concerns modules only, not the CS as a whole. We gather that all of the previous hypotheses and metrics can be used to study the stability of the CS modules separately.

4 Soundness of the Metrics

Having established the hypotheses on stability and the procedures to measure them by, we must ascertain their quality. Internal validity is: establishing the cause-and-effects as distinguished from spurious relationships. The metrics should produce verifiable outcomes based on clear measurement procedures, and be independent of the observer as well as the timing of the observation. In addition, the metrics ought to show the desired tendencies: a more stable CS should show more favorable outcomes of the metrics, and a less stable CS should show worse metrics. To illustrate this point, consider the example provided by [28] of a careless use of a 'cost-per-line-of-code' metric. A more powerful, productive programming language will obviously produce less lines of code. But the cost per line of code, calculated as the ratio of (variable-cost+ xed-cost) / (lines-of-code) may go up when xed costs are included in total cost.

We claim that our set of metrics possesses internal validity. This is because they are well associated with our framework for CS stability with three dimensions depicted in Figure 3. This is an argument from theory only; we do not claim validity based on statistical correlations [45]. However, we do not claim that all properties and mechanisms of stability are covered equally well, for instance no metric adresses the "facilitate change propagation" mechanism. This does not signify that the mechanism is unimportant in our framework; if so, it would have been left out. Rather the reason is that any metric for this mechanism involves non-conceptual features of the business environment. The change at the CS level must somehow be sized against the time and e ort spent in adapting applications and transaction-processing software, user interfaces, data storage etc. This approach can be seen in project planning methods such as Function Point Analysis, and it is evident that the metrics involved in FPA are not conceptual in nature.

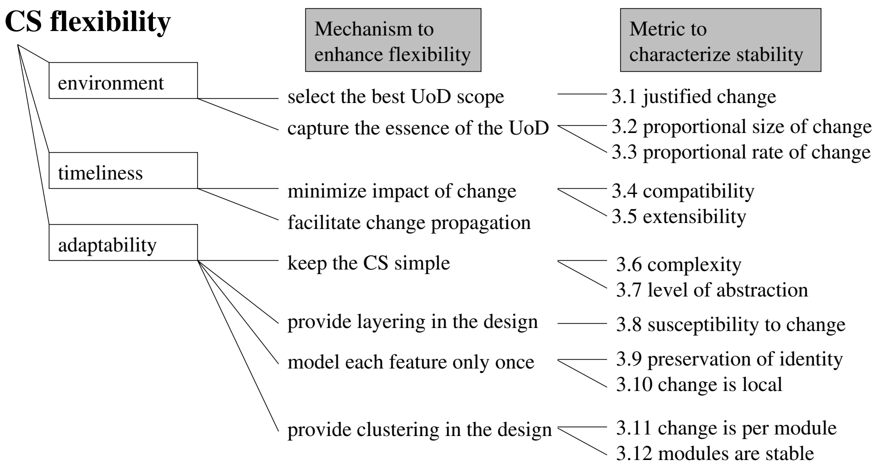
CS flexibility
Mechanism to enhance flexibility
Metric to characterize stability
environment
timeliness
adaptability
select the best UoD scope
capture the essence of the UoD
minimize impact of change
facilitate change propagation
keep the CS simple
provide layering in the design
model each feature only once
provide clustering in the design
3.1 justified change
3.2 proportional size of change
3.3 proportional rate of change
3.4 compatibility
3.5 extensibility
3.6 complexity
3.7 level of abstraction
3.8 susceptibility to change
3.9 preservation of identity
3.10 change is local
3.11 change is per module
3.12 modules are stable

Fig. 3. Metrics to characterize stability based on the framework

Internal validity rests on the fact that the metrics target only conceptual properties of operational CSs. Therefore, all CSs that satisfy the rst principles of good conceptual composition, can be subjected to our metrics. It can be easily checked that no metric explicitly includes non-conceptual characteristics of the business environment or database system such as:

– overall size of the database, i.e. the same set of metrics can be applied to study small to very large databases
– types of data access, an area covered by CRUD analyses [19], cohesion in methods [4,20] and other approaches
– intensity of data access and volatility (number of daily update transactions)

– number of users and user applications that access the database
– characteristics (constructs and constructions) of the specific Data Model Theory in use
– features of the software- or hardware-architecture of the enterprise such as data distribution or fragmentation across multiple sites, and
– the preferred design approach or the organizational/architectural design strategies.

But there is a complication. The metrics can have an implicit dependence on non-conceptual features of the business environments. Bias was pointed out in several metrics: rate of change, compatibility, and complexity. Finally, it must be kept in mind that the metrics are not geared towards design. If for some UoD, one design approach is superior to all others, given the particulars of the business environment, then this will not be discovered by our metrics.

We have no proof of completeness for our set of metrics, although we consider it rather convincing that the metrics cover the dimensions and mechnisms of the framework rather well. Even so, we cannot claim that all the dimensions and mechanisms of the framework are covered to their full extend. No metric for instance covers the 'facilitate change propagation' mechanism. The implication is not that the mechanism is unimportant. If we thought so, it would have been left out. Rather the reason is that any metric for this mechanism must involve non-conceptual features of the business environment.

5 Field Study Setup

Although we claim internal validity of these metrics, we do not claim their external validity. External validity of our set of metrics rests upon their successful application to schema evolution in actual business environments. The setup would be a longitudinal study into the evolution of one, or perhaps several CSs that lie at the heart of vital business information systems. The field study must investigate the phenomenon of change in the CS over a considerable length of time, long enough for the CS to evolve through several versions ([37] requires that at least two CS versions be secured). The aim of the study would be to demonstrate that the metrics can be applied in an operational setup, are objective and reliable enough, that they yield meaningful outcomes, and that they are adequate in understanding the overall flexibility of the CS in the long run.

A first test for feasibility of most of the metrics can be provided by a single case study. Some metrics yield only relative outcomes and to test them would require that multiple business cases be compared. Another argument in favor of multi-case field study is that more testing will lead to outcomes that are more reliable in a statistical sense [45]. Reliability is essential for the next step in our line of research, i.e. to switch from a study of past stability to a prediction of future flexibility. This challenging area of research is to study if, and how our metrics assist in the prediction of future CS changes. Our basic metrics would probably have to be aggregated into more effective ones, in a similar fashion to the approach taken in Function Point Analysis. The field studies can also be used to test metrics for topics that have not been covered yet, e.g. metrics for derived data, data-dependencies etc.

To detect change in the CS in a business situation can and will encounter problems such as

- lack of documentation. What has been changed in the CS may be discovered with database reverse-engineering methods and tools [44,25], but the business motivation for the change can only be learned from the stakeholders
- abundance of designs that represent the identical semantic structure of the UoD in syntactically different ways
- lack of coordination, where multiple schema releases are being constructed in parallel and the actual sequence of changes implemented in the CS remains uncertain
- strategic changes, such as a switch in the strategies for data processing
- technical change drivers, such as a change of database software. Businesses often find that new software releases invalidates current design decisions, and thus causes serious impact on the existing CS.

All of these problems must be addressed and resolved in order to conduct reliable field study into usability of the metrics. We claim that the study should use operational CSs and be conducted within their business context. The option to use a small-scale experiment is insufficient in our opinion for several reasons:

- real changes in a business environment are always subject to numerous explicit and implicit constraints
- seemingly unrelated changes in other information systems may have an unexpected impact on the present CS
- live systems have a degree of fault tolerance, that allows minor defects to be present in a CS without affecting the overall system quality
- lack of formal CS documentation in legacy systems maintenance, often being balanced by
- huge experience and personal knowledge of maintenance engineers.

We feel that a laboratory setup cannot reproduce these features realistically. To subject metrics intended for an operational environment to empirical validation [4,8] is in our view inadequate.

6 Related Work

Although many data modelling techniques exist that claim to deliver CSs of high quality, relatively few attempts have been made at studying the stability of schemas that are actually produced. It is remarkable that current literature pays so little attention to the important topic of measuring the stability as a determinant of CS quality. For instance, a paper by [36] is devoted to understanding quality in conceptual modeling, but it concentrates on the design phase and mentions modifiability only in a sidebar. Indeed the whole area of software measurement is considered to be immature [12,58].

[33] discusses how a satisfactory schema evolution can be supported. The focus is on enabling the propagation of changes, by creating a series of schema versions that coexist in the database system. However, the taxonomy they use consists of only 9 elementary schema transformations, and only the 'timeliness' dimension of CS stability is addressed, ignoring the other dimensions of business environment and schema adaptability.

[37] reports on a research into the stability of 7 CSs denoted in a relational-like data model theory. The following counts are used to define three metrics, namely the ratios of primary, secondary and tertiary attributes over the total number of entities:

- total number of entities
- total number of primary attributes, i.e. those that are essential to understand what the entity represents
- total number of secondary attributes, i.e. relevant data attributes that are not fundamental for understanding
- total number of tertiary attributes, i.e. those used to control and sustain processing needs

The study is limited to a single evolution step of the CSs. It is observed how the average number of primary and secondary attributes per entity increases significantly, whereas the ratio of tertiary attributes per entity halves. We feel that the observations in the report describe symptoms, rather than the essence of the stability problem, and any conclusions drawn from them remain largely intuitive because a formal framework linking the metrics with CS stability criteria is lacking. The idea of the three discerned types of attributes is appealing, but it is unclear what basis it has in theory or accepted best-practices.

[48] describes a longitudinal field study of the evolution in a single relational CS covering parts of both the development period and the operational phase. His findings are that all entities are affected by change at least once over the duration of the field study. However, a serious drawback in his approach is that a very simple taxonomy is used that lacks elements like attribute transformation. It is found that the numbers of attribute deletions and additions are approximately equal, but this finding may indicate that attributes are mostly altered in some way, and this goes undetected because of the poor taxonomy.

[32] develop a theory for strategic information systems planning that includes several hypotheses related to stability and complexity of the systems environment. The theory tries to capture the main determinants for stability and complexity of the strategic information systems, and the tacit assumption is these determinants will also ensure the stability of the CSs that will lie at the heart of the systems.

We have paid little attention to the issue of facilitating change propagation. Database facilities and techniques to enable propagation of changes with minimal interruption of database services are the subject of ongoing research, especially in object-orientation [2,23,41].

Whereas our focus is on evolution in the CS, several researchers are investigating the area of evolving data model theory. Because the impact of changes of data model theory at the CS and data levels can be huge, we feel that such research should first consider what the intended benefits are for flexibility of the CS and quality of the operational information systems in the long term. Only if proper goals are set can metrics be introduced to investigate and understand what might be called: meta-evolution [51].

7 Conclusions

This paper has introduced a framework for assessing schema stability, consisting of three dimensions. These were further refined into a number of mechanisms and 'best-practices' for enhancing the flexibility of conceptual schemas. We used this framework to develop a set of metrics that measure the evolution of the CS with respect to each mechanism. Each metric has been rigorously defined in an operational sense, so that outcomes will be consistent and repeatable when applied to an evolving CS.

Nevertheless, some of our metrics for schema evolution build upon measures for static CS composition, which are not always available and well-defined. For instance, the hypothesis that more abstract CSs will go through less changes, requires a preestablished measure of schema abstraction, which is found to be lacking. But although the metric cannot be defined in an operational way, the hypothesis can still be formulated and indicate the tendency in CS evolution.

The proposed set of metrics, which we do not claim to be exhaustive, can provide valuable insights into the working mechanisms for schema evolution. Only when the elusive relationship between current characteristics of the Conceptual Schema and their behaviour in future changes is well understood, can we hope to improve current practices in database schema evolution.

Research directions. An important goal of current research is to determine stability of an operational CS from a business point of view, i.e. to understand the relationship between the syntactic change in the CS and the semantics of the change driver. To this end, we are validating the metrics as a set of objective measures for stability as a quality aspect of a given CS. Field research is in progress to bring out which of the above metrics are best suited to gauge the stability of schemas, and the impacts of proposed changes. The next challenge in research is to study if, and how our metrics assist in the prediction of future CS changes. We want to use these metrics and develop from them a set of maintenance guidelines how to safeguard and enhance schema quality when faced with changing information requirements and evolving schemas. A related area where research is generally lacking is in bridging the gap between the design and operational phases of the CS life cycle. It is common business experience that the process of mapping a CS into a feasible database schema, requires many implementation choices to be made. A considerable amount of those choices are conceptual in nature, and ought to be incorporated as adjustments and amendments on the CS design. No theoretical framework nor practical research is available that charts the kinds of changes that are made, and whether the effect of changes on the CS is detrimental or beneficial to the schema stability in the long run.

Fundamental research is needed in several areas where clarity of terms is lacking. We already indicated how the notion of schema abstraction needs to be clarified, the same problem was encountered in schema complexity. A promising direction for theoretic as well as applied research is in disclosing the mechanisms underlying our set of hypotheses. This research should include how data model theories contribute to each hypothesis. Proponents of state-of-the-art modelling approaches and design strategies make a variety of claims about schema stability and flexibility. However, their references to stability are mostly unspecific, leaving unclear if claims of stability are substantiated and by what mechanism the promise of stability is realized. A paper is planned to analyse what mechanisms underlies the claims of design strategies, using our framework from Section 2.

Another line of research which can be pursued is strategic alignment, i.e. to match CS stability with business strategy and planning, in order to understand the dynamics of the joint evolution of the business environment and information systems. Other areas where these metrics may prove worthwhile is in estimating cost and effort of a proposed change, in portfolio analysis, in benchmarking organizations on their maintenance performance, etc.

References

1. Amikam A. On the automatic generation of internal schemata, Information Systems vol 10 no 1 [1985] 37–45
2. Andany J., Léonard M., Palisser C. Management of Schema Evolution in Databases, 17th Int. Conference on VLDB'91 Very Large Data Bases, Barcelona (Spain), Morgan Kaufmann, San Francisco [1991 09] 161–170
3. ANSI/X3/SPARC Study Group on Data Base Management Systems Interim Report, ACM SIGMOD Newsletter vol 7 no 2 [1975 02 08]
4. Basili V.R., Briand L., Melo W.L. A Validation of Object-Oriented Metrics as Quality Indicators, IEEE Transactions on Software Engineering vol 22 no 10 [1996 10] 751–761
5. Batini C.W., Ceri S., Navathe S.B. Conceptual Database Design: An Entity-Relationship Approach, Benjamin/Cummings Publ. CA [1992]
6. Batini C.W., Di Battista G., Santucci G. Structuring primitives for a Dictionary of ER Data Schemas, IEEE Transactions on Software Engineering vol 19 no 4 [1993] 344–365
7. Batra D., Antony S.R. Novice errors in Conceptual Database Design, European Journal of Information Systems vol 3 no 1 [1994] 57–69
8. Batra D., Davis J.G. Conceptual data modelling in database design: similarities and differences between expert and novice designers, Int. Journal of Man-Machine Studies vol 37 [1992] 83–101
9. Belady L.A., Lehman M.M. A model of large program development, IBM Systems Journal vol 15 no 3 [1976] 225–252
10. Castano S., de Antonellis V., Zonta B. Classifying and Reusing Conceptual Schemas, ER'92 Entity-Relationship Approach, editors Pernul G., Tjoa A.M., Springer Verlag series LNCS 645 [1992 10] 121–138
11. Chen P.-P., Ming-rui Li The lattice structure of entity sets, ER'86 Entity-Relationship Approach, editor Spaccapietra S. Elsevier Science Publ. North-Holland [1986] 217-229

12. Chidamber S.R., Kemerer C.F. A metrics suite for Object-Oriented Design, IEEE Transactions on Software Engineering vol 20 no 6 [1994 06] 476–493
13. Claypool K.T., Rundensteiner E.A., Heineman G.T. Evolving the software of a Schema Evolution System, Database schema Evolution and Meta-Modeling — 9th Int.Workshop on Foundations and Models of Data and Objects (FOMLADO/DEMM'00), editors Balsters H., de Brock B., Conrad S., Springer Verlag series LNCS 2065 [2001] 68–84
14. Date C.J. An introduction to Database Systems, volume I, fourth edition, Addison-Wesley Publ. [1986]
15. Delen G.P.A.J., Looijen M. Beheer van Informatievoorziening, Cap Gemini Publishing Netherlands (isbn 90-7199650-6) [1992] (in Dutch)
16. de Troyer O. On Data Schema transformations, PhD thesis Tilburg University [1993 03]
17. Dittrich K.R., Gotthard W., Lockemann P.C. Complex entities for Engineering Applications, ER'86 Entity-Relationship Approach 1986 editor Spaccapietra S., Elsevier Science Publ. North-Holland [1986] 421–440
18. Dvorak J. Conceptual Entropy and its Effect on Class Hierarchies, Computer [1994] 59–63
19. Ebels E.J., Stegwee R.A. A Multiple Methodology Approach towards Information Architecture Specification, Proceedings of the '92 IRMA Int.Conference, Charleston SC, editor Khosrowpour [1992 05] 186–193
20. Etzkorn L., Davis C., Li W. A Practical look at the lack of cohesion in methods metric, JOOP Journal of OO Programming vol 11 no 5 [1998 09] 27–34
21. Ewald C.A., Orlowska M.E. A Procedural Approach to Schema Evolution, CAiSE '93 Advanced Information Systems Engineering, 5th Int.Conference Paris France, Springer Verlag series LNCS 685 [1993 06] 22–38
22. Feldman P., Miller D. Entity Model clustering: structuring a data model by abstraction, The Computer Journal vol 29 no 4 [1986] 348–360
23. Ferrandina F., Meyer T., Zicari R., Ferran G., Madec J. Schema and database evolution in the O2 object database system, Proceedings of the 21st VLDB'95 Very Large Data Bases conference, Zürich, editors Dayal U., Gray P.M.D., Nishio S. [1995 09] 170–181
24. Gill G.K., Kemerer C.F. Cyclomatic complexity density and software maintenance productivity, IEEE Transactions on Software Engineering vol 17 no 12 [1991] 1284–1288
25. Hainaut J.-L., Engelbert V. DBMAIN: a next-generation meta-CASE, Information Systems vol 24 no 2 [1999 04] 99–112
26. Jensen O.G., Böhlen M.H. Evolving Relations, Database schema Evolution and Meta-Modeling — 9th Int.Workshop on Foundations and Models of Data and Objects (FOMLADO/DEMM'00), editors Balsters H., de Brock B., Conrad S., Springer Verlag series LNCS 2065 [2001] 115–131
27. Jianhua Zhu, Nassif R., Pankaj G., Drew P., Askelid B. Incorporating a model hierarchy into the ER paradigm, ER'91 Entity-Relationship Approach [1991] 75–88
28. Jones C. Software Metrics: good, bad, and missing, Computer vol 27 no 9 [1994 09] 98–100
29. Jones M.C., Rundensteiner E.A. An Object Model and Algebra for the Implicit Unfolding of Hierarchical Structures [1999] (internet, downloaded on 2000-01-31)
30. Kesh S. Evaluating the quality of Entity Relationship models, Information & Software Technology vol 37 [1995] 681–689

31. Lautemann S.-E. A propagation mechanism for populated schema versions, Proceedings of the IEEE Int. Conference on Data Engineering [1997 04] 67–78
32. Lederer A., Salmela H. Towards a theory of strategic information systems planning, J. of Strategic Information Systems vol 5 no 3 [1996 09] 237–253
33. Lerner B.S., Habermann A.N. Beyond schema evolution to database reorganization, Proceedings of the ECOOP/OOPSLA'90 conference, SIGPLAN Notices vol 25 no 10 [1990 10] 67–76
34. Lerner B.S. A Model for Compound Type Changes encountered in Schema Evolution, Technical Report 96-044 Univ Massachusetts [1996 06] (internet, downloaded on 2000.02.29)
35. Levitin A.V., Redman T.C. Quality dimensions of a conceptual view, Information Processing & Management vol 31 no 1 [1995] 81–88
36. Lindland O.I., Sindre G., Sølvberg A. Understanding Quality in Conceptual Modeling, IEEE Software [1994 03] 42–49
37. Marche S. Measuring the stability of data models, European J.of Information Systems, a publication of the Operational Research Society, vol 2 no 1 [1993] 37–47
38. Mistelbauer H. Datenmodellverdichtung: Vom Projektdatenmodell zur Unternehmens-Datenarchitektur, Wirtschaftsinformatik vol 33 no 4 [1991 08] 289–299 (in german)
39. Pels H.J. Geïntegreerde informatiebanken: modulair ontwerp van het conceptuele schema, Stenfert Kroese Leiden NL [1988] (in dutch)
40. Peters R.J., Tamer Özsu M. Reflection in a Uniform Behaviour Object Model, ER'93 Entity-Relationship Approach 1993, 12th Int.Conference Arlington (Texas), editors Elmasri, Kouramajian, Thalheim, Springer Verlag series LNCS 823 [1993 12] 34–45
41. Peters R.J., Tamer Özsu M. An axiomatic model of dynamic schema evolution in objectbase systems, ACM transactions on Database Systems vol 22 no 1 [1997 03] 75–114
42. Proper H.A., van der Weide T.P. Towards a general theory for the evolution of application domains, Proc. Australasian Database Conference'93, editors Orlowska, Papazoglou, World Scientific [1993 02] 346–362
43. Roddick J.F, Craske N.G., Richards T.J. A Taxonomy for Schema Versioning Based on the Relational and Entity Relationship Models, ER'93 Entity-Relationship Approach, 12th Int.Conference Arlington (Texas), editors Elmasri, Kouramajian, Thalheim, Springer Verlag series LNCS 823 [1993 12] 137–148
44. Sauter C. Ein Ansatz für das Reverse Engineering relationaler Datenbanken, Wirtschaftsinformatik vol 37 no 3 [1995] 242–250 (in German)
45. Schneidewind N.F. Methodology for Validating Software Metrics, IEEE Transactions on Software Engineering vol 18 no 5 [1992 05] 410–422
46. Shanks G., Darke P. Understanding corporate data models, Information & Management vol 35 no 1 [1999] 19–30
47. Sheth A., Kashyap V., So Far (schematically) Yet So Near (semantically), Proceedings of the IFIP WG2.6 DS-5 Conference, Lorne Australia [1992 11 16] 272–301
48. Sjøberg D. Quantifying Schema Evolution, Information & Software Technology vol 35 no 1 [1993 01] 35–44
49. Storey V.C., Ullrich H., Sundaresan S. An Ontology for Database Design Automation, ER'97 Entity-Relationship Approach 1997, editors Embley, Goldstein, Springer Verlag series LNCS 1331 [1997] 2–15
50. Teorey T.J. Database Modeling & Design: The fundamental Principles, 2nd edition Morgan Kaufmann Publ. [1994]

51. Terrasse M.-N. A Modeling Approach to Meta-Evolution, Database schema Evolution and Meta-Modeling — 9th Int.Workshop on Foundations and Models of Data and Objects (FOMLADO/DEMM'00), editors Balsters H., de Brock B., Conrad S., Springer Verlag series LNCS 2065 [2001] 201–218
52. Urtado C., Oussalah C. Complex entity versioning at two granularity levels, Information Systems vol 23 no 3/4 [1998] 197–216
53. Veldwijk R. Hoe rekbaar is flexibel, Database Magazine [1996] 38–43 (in dutch)
54. Wand Y., Monarchi D.E., Parsons J., Woo C.C. Theoretical foundations for conceptual modelling in information systems development, Decision Support Systems vol 15 [1995] 285–304
55. Wedemeijer L. Semantic Change Patterns in the Conceptual Schema, ECDM'99 Advances in Conceptual Modeling, editors Chen, Embley, Kouloumdjian, Liddle, Roddick, Springer Verlag series LNCS 1727 [1999 11] 122–133
56. Wedemeijer L. A Method to Ease Schema Evolution, IRMA'00 International Conference, Anchorage AK, [2000 05] 423–425
57. Wilmot R.B. Foreign keys decrease adaptability of database designs, Communications of the ACM vol 27 no 12 [1984 12] 1237–1243
58. Zuse H. A Framework of Software Measurement, Walter de Gruyter Berlin, NewYork [1998]