Are the Terms "Version" and "Variant" Orthogonal to One Another?

- A Critical Assessment of the STEP Standardization -

Hartmut Wedekind
University of Erlangen-Nuremberg, Germany
E–mail: Wedekind@immd6.informatik.uni–erlangen.de

Abstract

Versions and variants are an important issue in industrial DB-applications like CAD/CAM. It is argued that a clean distinction between both terms is of great importance.

1 Introduction

In the 1980's, practical and applied computer science carried out extensive discussions about the problems of version and variant behavior in different contexts, the themes "concurrency control" and "development and management of technical products" were the center interests [DL 88], [Ka 90]. The latter theme lead to considerable confusion, because the terms "version" and "variant" are used differently, depending on the perspective from which the changes to technical products are viewed. For instance, one perspective could be the release of a technical product; a validation question is tied to this perspective: "What is valid from which point in time?" Another view might be coupled with the question: "Which intermediate states occur during the problem solution (development) in order to achieve an end result that meets the product specification?" (solution question).

The many facets of the version/variant discussion is reduced here to the fundamental question whether or not the terms "version" and "variant" in scientific language usage are orthogonal (independent) to one another. The question of the term constitution is to be the focus of this discussion.

2 Versions and Variants as Orthogonal Terms

Orthogonality suggests a binary relationship R between two terms, a and b. We write: a R b. R is assumed to be symmetrical, i.e. $a\,R\,b \rightarrow b\,R\,a$. Orthogonality between a and b means that when specifying an expression for b ($y \in b$), the selection of expressions for a ($x \in a$) is independent, and vice versa.

$$a\,R\,b =_{Def} \exists_{x \in a}\, \exists_{y \in b}\, xRy \leftrightarrow \exists_{y \in b}\, \exists_{x \in a}\, xRy$$

When considering the technical product aspect, it is helpful to discuss versions and variants with regards to a parts list (bill of material). Both terms are thus seen uniform in the context of a part/whole relationship (Figure 1).

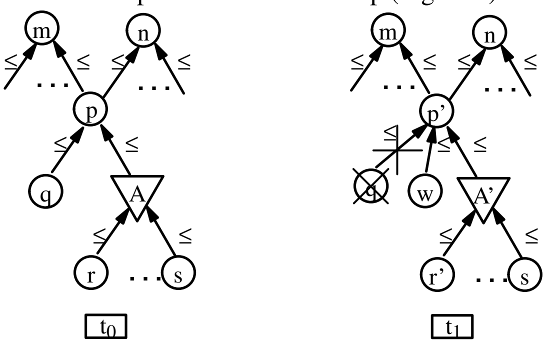

Figure 1: Extracts from a variant parts list at point in time t_0 and later point in time t_1

An arrow in Figure 1 depicts a part/whole relationship that will be noted by the ≤ symbol. Circles represent parts, such as q, r, and s, that are individual parts at the point in time t_0, because they are not composed of subbassemblies. Circles p, m, and n are composite parts (sub-assemblies). They represent a whole with respect to their components. Because all components are required, a node for a composite part can be interpreted logically as a conjunction (∧). The inverted triangle with the name A is called an alternative node. It represents an exclusive-or: either part r or part s will be combined with part q to produce the whole p. The variant parts list is distinguishable by the alternative nodes. One differentiates between "can"-variants (optional, such as a right passenger side rear-view mirror) and "must"-variants (required, such as 4 or 5 speed transmissions). For can-variants, an empty part (no-part) is made part of the selection set. In Figure 1, we purposely left out quantity entries for the arrows in order to create a simpler representation. Thereby, only one unit of the corresponding part is employed. At time t_0 in Figure 1, p is described by the must-variants $p_1 =_{Def} q \leq p \wedge r \leq p$ and $p_2 =_{Def} q \leq p \wedge s \leq p$.

The creation of a parts list is a design process concern. However, the point in time that a parts list validation is obtained is another story altogether. It is not the design department with their answers to a problem, but a standards office working in accords with instructions from a technical guide who determines the time t_0 when a parts list with its variants takes effect. A design parts list without validation approval from the standards office may be theoretically important; but in practice it is of no consequence. As Kant would say, here the primacy of a practical reasoning over a theoretical becomes clear. This is often a painful experience for design engineers, because only wastebaskets are filled. The pains of technical management and the standards office are of a completely different nature. They arise when parts lists having validation are meaningless, because nobody is interested in the product.

A transition from t_0 to t_1, the point in time of saving another validation, puts the parts list in Figure1 through multiple changes. In the language of the orthogonal term creation, one says that pieces of the product are "versioning". At time t_1, the following engineering changes take effect: Part q is replaced by part w. Part q disappears. The part r changes to part r' at time t_1. As a consequence of the component changes, the assembled component p versions to p'. r' and p' are modified inherent names (primary keys) for objects. The modification is represented for instance through time stamping: $<p,t_0>$ for p and $<p,t_1>$ for p'. It is assumed that the parts m and n are not versioned. Whether or not an assembly is versioned whenever its components are versioned is discussed in the following sections.

Two relationships, ξ and η, are to be considered:

$p \,\xi\, p'$	means	p is <u>versioned to</u> p'
$p_1 \,\eta\, p_2$	means	p_1 is <u>varied to</u> p_2,

wherein, with respect to Figure 1, $p_1' \,\eta\, p_2'$ can be asserted, with $p_1' =_{Def} w \le p' \wedge r' \le p'$ and $p_2' =_{Def} w \le p' \wedge s \le p'$

2.1 The Version Relationship

Versions and variants are two orthogonal concepts; meaning, one is -- terminolog speaking -- completely free (independent) when and how to version and how to build variants. The independence is supported by institutions and enforced by organizations: standards offices here and design departments there. Versioning is time dependent; variant building is time independent.

The relationships ξ and η have fundamentally different characteristics. What can be generally said about ξ is extremely meager:

ξ is comparable; meaning, it holds that $p \,\xi\, p' \vee p' \,\xi\, p$.

If one were capable of making the following assertions:

ξ is reflexive: $p \,\xi\, p$ and

ξ is transitive: $p \,\xi\, p' \wedge p' \,\xi\, p'' \rightarrow p \,\xi\, p''$,

then ξ would describe a linear ordering (total ordering). The objections against reflexivity are mostly of a language logical nature. After all, a part can not be versioned into itself. Even more difficult are the arguments against transitivity. In general, a version p' is needed for reasons of insight (market entrance, technical functionality). A skipping over p' and a transition $p \,\xi\, p''$ is excluded as a rule in a world in which new ground is touched.

Even though no linear ordering is imposed in a strict sense, the version creation can be represented linearly (Figure 2).

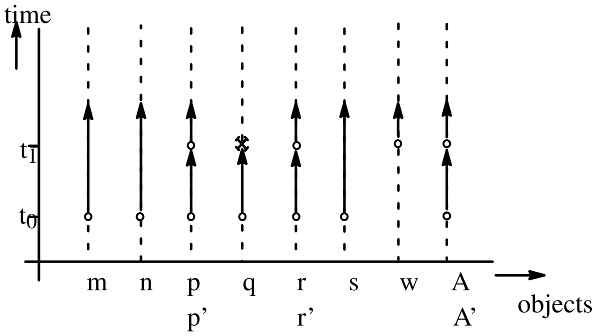

Figure 2: Part versions in a time graph. ✕ means "death of a part".

Belonging to the term "version", aside from comparability, is the condition that old versions are not allowed to be changed anymore. Arbitrarily given actual states in the past are historical, and change after the fact to the history is called "history tinkering". Old versions are only allowed to be read. When this occurs from a historical perspective, such as to find out what was once valid, then there is nothing objectionable in the reading process. But when this occurs in the course of building a system from old versions, a difficult problem arises that is dubbed in American terminology as "legacy" ("old burden"). Since old versions also arrive outside of the system, there is no system based solution for legacies (out-of-system problem). If a factory has to manage 30,000 and more parts today, the "legacy" problem becomes extremely significant. The standards office can reduce the problem by placing read-locks on old versions very early.

One can see from Figure 1 that not only the part versions but also the structure versions have to be managed. Part/whole relationships (≤) disappear or are entered new. In relational database systems, such management creates essentially no great problems [MSW 84]. The time independent part origin files and structure files,

part(<u>p#,</u> type, ...) and
structure(<u>up#, lp#,</u> ...),

having p# = part number, type = node type (e.g. part node t or alternative node A), up# = upper part number, lp# = lower part number, are then expanded around the date attribute registering the validation times. An additional boolean attribute alive=(true, false) provides information about whether a version is valid at a point in time or can be viewed as deleted.

The part lists in Figure 1 are represented with the following two relationships:

PartVersion (p#, date, alive, version dependent attribute)

m	t_0	true	d_1
n	t_0	true	d_2
p	t_0	true	d_3
p	t_1	true	d_3'
q	t_0	true	d_4
q	t_1	false	d_4'
r	t_0	true	d_5
r	t_1	true	d_5'
s	t_0	true	d_6
w	t_1	true	d_7
A	t_0	true	d_8
A	t_1	true	d_8'

StructureVersion (up#, lp#, date, alive, vers.depend. attre.)

A	r	t_0	true	s_1
A	r	t_1	true	s_1'
A	s	t_0	true	s_2
p	q	t_0	true	s_3
p	q	t_1	false	s_3
p	w	t_1	true	s_4
p	A	t_0	true	s_5
p	A	t_1	true	s_5'
m	p	t_0	true	s_6
m	p	t_1	true	s_6'
n	p	t_0	true	s_7
n	p	t_1	true	s_8

A special situation happens during the version building whenever a new version is employed similarly as previous versions. The ISO standard "Standard for Exchange of Production Definition Data (STEP)" [ISO 91] speaks about "alternate products" in this case and suggests the "employed equally" relationship as an equivalence relationship. "Employed equally" deals with an abstract equality, because in reality the parts remain different. (The ISO Standard mentioned an example of two screws of the same size: one becomes a phillips screw while the other a regular.) We use the symbol "=" for the "employed equally" relationship and assume that the employment of the parts can be specified through their interface, which is logically given as a propositional form S(x). Here, x is the inherent name of a part. Employment equality can be expressed through substitution of one part with that of another:

$$x = y \wedge S(x) \rightarrow S(y)$$
$$x = y \wedge S(y) \rightarrow S(x)$$

The logical conjunction of these two statements can be transformed to:

$$(x = y) \rightarrow [S(x) \leftrightarrow S(y)]$$

Two parts are employment equal if their truth values do not change when substituting in the interface S. It is said in object-oriented programming that the objects demonstrate an equal behavior. In logic, one speaks of an invariant statement S with respect to "=".

Speaking about invariant propositions leads to an abstraction. It might be thinkable to view p and p' in Figure 1 as employment equal, meaning p = p'. One then speaks generally of abstract part p and saves versioning. Managing "engineering changes" in this non-visible manner is normal in practice. In automobile manufacturing, there are thousands of engineering changes per year that do not cause one to reconsider the cars and their names. The abstraction is not allowed if it concerns itself with the parts list decomposition. Taking a unit apart into its individual pieces, at a specific level one sees that p and p' are definitely different. It is common to decompose only with respect to the last active version. The spare parts group with their "legacies",are demand controlled from stock and not production program controlled, as with the active versions. It is worth noting that versions which, in general, can only be considered with the necessary predicate "comparable" can also occur at the "higher levels" of an equivalence relation. This means for "=":

= is reflexive: $p = p$
= is transitive: $p = p' \wedge p' = p'' \rightarrow p = p''$
= is symmetrical: $p = p' \rightarrow p' = p$

These characteristics hold only for abstract p. For actual p, it remains with the comparability.

2.2 The variant relationship

According to the understanding of many database theorists, the "version building" theme belongs in the area of "temporal databases". In expanding upon temporal approaches, a theory of change is even offered today [RE 83]. Unfortunately, this deals with an axiomatic and not with a constructive theory. Assumptions about an object membership are made and no construction of objects is presented. Temporal statements and their theory, temporal logic, are fundamentally intensional and not extensional. Extensional propositions with equal truth values can be substituted both ways, as occurs in classical logic, without influencing the truth values of the whole proposition. This is not so for intensional proposition. Whoever, for instances, exchanges statements about the past with those about the future and believes that the truth values of the whole statement are left unchanged, is making a big logical mistake.

For variant building, time is not constitutive. Variants can be seen as free of time. A specification of the variant relationship can be made over the language usage without having to go into intensional questions of gaining insight.

For η, the variants building operator, the following propertis hold:

η is symmetrical: $p_1 \, \eta \, p_2 \rightarrow p_2 \, \eta \, p_1$
η ist transitive: $p_1 \, \eta \, p_2 \wedge p_2 \, \eta \, p_3 \rightarrow p_1 \, \eta \, p_3$

Reflexivity is to be excluded. A part can have no variants to itself. Irreflexivity is the case.

Whoever asserts, that p_1 is a variant of p_2, may also assert that p_2 is a variant of p_1 (symmetry). Whoever says, that p_1 is a variant of p_2 and p_2 one of p_3, may also without risk establish the same relationship for p_1 and p_3 (transitive). p_2 is not a necessary transition part with respect to the problem solution, but is an elimination part whose detour can be cut-out. For creating orthogonal terms, no one can cut out a version as being a detour. Detours only occur when the goal is known; who already knows how a part - shall we say - at the end of next year will look? At the conclusion of next year, then in certain cases we can speak of detours. After the fact one is always wiser. Whoever speaks of the future as if it were the past speaks pseudo-temporal.

3 Versions and Variants as Synonyms or as Species/Genus terms, resp.

In the wealth of literature where the question of the solution and not the question of validation is the center of attention, this pair of terms is treated as species/genus term or even as synonyms. The ISO STEP standard introduces "variants" as genus. Versions then are the species. For orthogonal thought, the following dreadful sentences appear in [ISO 91, pg. 223]:

"A product version is a variant of a product. It brings together all the information that relates to a specific version of a particular product. At any given time there may be multiple active versions of a given product and multiple obsolete versions for the same product."

If it can be additionally concluded that variants are also versions, then their synonymous nature (employment equality) comes into question. Synonymity is an equivalence relationship which makes the terminology pair one solo term. One arrives at an abstract level wherein it must be decided whether one is speaking about abstract variants or, equivalently, about abstract versions. In computer science, the term "version" is preferred on an abstract level. In the engineering realm, one still speaks of variant construction; under the influence of computer science, this may soon become a version construction.

In jurisprudence - which some may dismiss as a debatable science - criminal and civil law are orthogonal to one another. A sub-ordering or over-ordering of both legal systems or even a synonyming would lead to a completely different term "legal state", having nothing to do with today's. Even in economics - according to Dijkstra (On the cruelty of really teaching computer science): A "mischievous science" the orthogonality between personal and institutional earning distribution would not be given up without necessity. Those in practical and applied computer science have the freedom to behave self-reliantly. Liberality has however its limits when it leads to chaos. Software releases, publication of books, diverse yet valid editions of legal text, and even running account balances are effortlessly recognized by everyone as versions; and a car dealer knows that, with 4 and 5 speed transmission offers, the selection is available between simultaneously valid products (variants).

If we adopt the ISO terminology, product development could proceed in a manner corresponding to Figure 3.

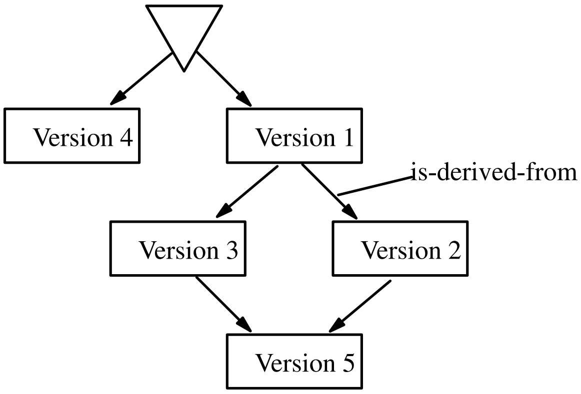

Figure 3: Versions in a derivation graph

In Figure 3, a triangle again represents an exclusive-or. Version 4 is considered an alternative to versions 1, 2, 3, and 5 [KM 93] or correspondingly a "parallel version" [KA 90]. The relationship has the characteristics of a variant relationship. The relationship is-derived-from (ρ) [KA 90] is an ordering relationship like the part/whole relationship; meaning, in addition to reflexivity and transitive, anti-symmetry is required:

ρ is anti-symmetrical: $x \, \rho \, y \wedge y \, \rho \, x \rightarrow x = y$

The relationship "is-derived-from" that takes us from version to version is studied in detail in psychology [DÖ 76]. Psychology talks about interpolation problems with the problem depicted in Figure 3. "With an interpolation problem, it depends on the 'path' constructed with known operators from the known initial situation to the known end situation." [DÖ 76, pg.56] With a known beginning and end, the problem is finding the intermediate states, named interpolation by the psychologists. Dörner impressively demonstrates an interpolations problem with the zoological cultivation of "swimming oil-eating-beetles". By applying different irradiations (operators) it is possible to create the desired "end beetle" from the "start beetle" after six "intermediate beetles". Mathematical calculations are prototypes for interpolations problems, in which an end figure is derived from starting figures through application of defined operators.

Carrying out a derivation relationship (derivation graph) as an ordering relationship in addition to a part/whole relationship (part list) cannot be refused. There is no problem interpreting Figure 3 in a time diagram (Figure 4) and adopting an orthogonality. Diverse interpretations are possible, because the validity question in Figure 3 is not posed at all.

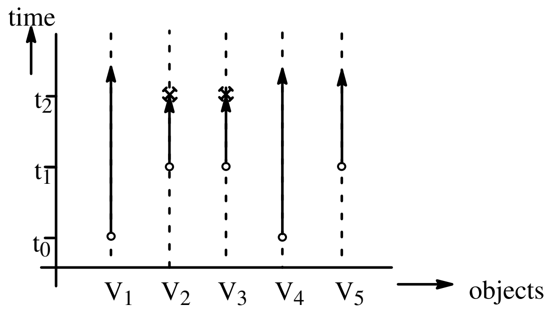

Figure 4: Figure 3 as a time diagram with V1, V2, V3, V4, and V5 as variants.

There is a fundamental difference between orthogonal and non-orthogonal thoughts which can be drawn out of the general aspect of problem solving. For this purpose, we turn again to the book from Dörner (Figure 5).

		Clarity of goal criteria	
		high	low
Knowledge rate of the medium	high	interpolation barriers I	dialectic barriers III
	low	synthesis barriers II	dialectic barriers and synthesis barriers IV

Figure 5: Classification of barrier types in problems according to dimensions “knowledge rate of the medium” and “clarity of goal situation” [DÖ 76].

Interpolation problems (I) cover only a small portion of the types of problem solutions, because the knowledge rate of the means (operator) and the clarity of the goal criteria have to be high. Even more difficult is the problem if the operators are unspecified, as in case (II). Katz [KA 90] even discusses this case of a synthesis barrier not at all. Our knowledge is meager whenever the clarity of the goal criteria is low or whenever the goal specification is subject to dynamic change, because we have to adjust constantly to a changing world (cases III and IV). Panta rhei -- everything flows. If variants and versions are seen orthogonally as in this paper, the question of the problem solving type becomes irrelevant. For the former, the fundamental problem is to exclude meaning less variants, whereas for the latter, the fundamental question “quid iuris” (what is valid?) has to be answered when creating a new version.

References

[BM 88] Beech, D. und Mahbod, B.: Generalized Version Control in an Object-Oriented Database, in: Proc. of the 4th Int. Conf. on Data Engineering, 1988, S. 14-22

[DL 88] Dittrich, K.R. und Lorie, R.A.: Version Support for Engineering Database Systems, in: IEEE Transactions on Software Engineering, Vol. 14 (1988), No. 4, S. 429-437

[DÖ 76] Dörner, D.: Problem solving as information processing, Kohlhammer-Verlag, Stuttgart 1976

[ISO 91] International Organization for Standardization: Product Data Representation and Exchange-Part 44, Product Structure Configuration, ISO CD 10303-44, August 1991 und Part 203, Application Protocol: Configuration Controlled Design ISO, CD 10303-203, September 1991

[KA 90] Katz, R.H.: Toward a Unified Framework for Version Modeling in Engineering Databases, in: ACM Computing Surveys, Vol. 22 (1990), No. 4, S. 375-408

[KM 93] Käfer, W. und Mitschang, B.: Flexible design management for CAD frameworks: concept implementation and assessement, in: Stucky, W. und Oberweis (Hrsg.): Tagungsband der GI-Fachtagung “Datenbanksysteme in Büro, Technik und Wissenschaft”, Springer-Verlag, 1993, S. 144-163

[MSW 84] Müller, T., Steinbauer, D. und Wedekind, H.: Control of Versions in Database Applications, in: Proc. IEEE Conf. on Trends and Applications, Gaithersburg, MD, 1984, S. 308-316

[RE 93] Revesz, P.Z.: On the semantics of Theory Change: Arbitration between Old and New Information, in: Proc. Principle of Database Systems (PODS), Washington DC, 1993, S. 71-82

[WM 81] Wedekind, H. und Müller, T.: Partslistorganisation in presence of a high number of variants, in: Angewandte Informatik, Heft 9, Sept. 1981, S. 377-383