Semantic Naming Convention and Software Quality

Stefan Berner, Sigmaplan AG, Switzerland

Summary:
What influence does the quality of names have on software products? Semantic naming conventions can help to improve understandability, correctness and maintainability of software.

A name can be said to be good, when a person who reads it knows immediately what it stands for and what it means. There should be no need to think a long time over the meaning of a name or to read a long description of the element referred to. Good names are *self-explanatory*. Good names avoid misunderstandings, shorten the time to understand something and can help to avoid errors by forcing developers to think more about the *thing* described by the name.

Examples are given to show the mechanisms of understanding the meaning of names. What does *self-explanatory* mean and how can it be achieved?

This paper presents a set of rules and naming conventions for various software elements.

The paper concludes with observations which the author has made during his consulting activities: "What makes it so difficult to employ the presented naming conventions and standards in real-life projects"

Stefan Berner, Sigmaplan AG, Zähringerstr. 61, CH-3012 Bern
email: sberner@sigmaplan.ch, Tel: +41/31/3012365

1. Quality of names - quality of software

In the *good old times* of programming we did not worry about names. I25 or X2 were just fine, especially as most programminglanguages did not allow identifiers longer than 6 or 8 characters.

So why bother with finding good names?

What is the justification for *spending time* on better names? There are several reasons for this, as the quality of the software can thereby be improved:

1.1. Maintainability

The better the name of a software element[1], the faster its meaning is understood.

Compare `a := b / 100 * c * m / 12`
with `interest := amount / 100 * interest_rate / 12 * month`

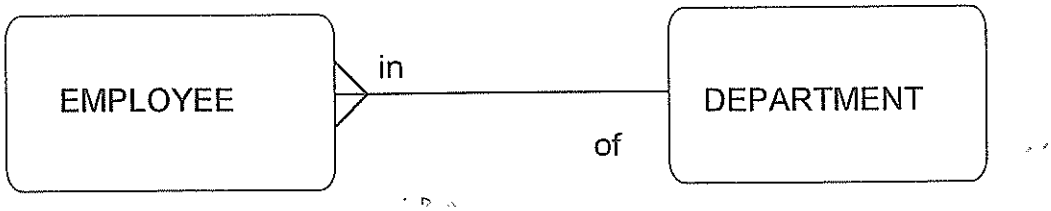
The savings resulting from this quality improvement are obvious for all those who have ever tried to maintain old programs.

1.2. Correctness

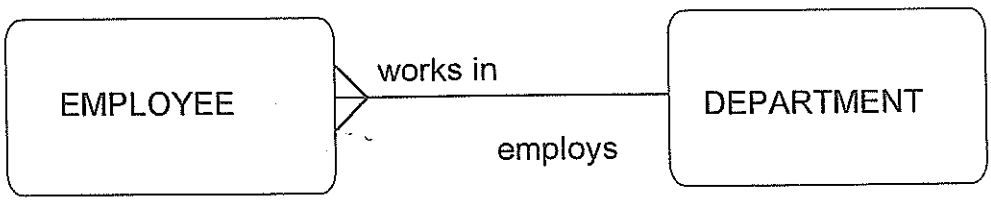
The meaning of an element is enhanced by a more exact specification of its name. Thereby:

- fewer misunderstandings can happen
- more discussion will take place about the name and the *thing* it names
- implying deeper insight of the author into the problem

The following example:



will be accepted by almost everybody because you can interpret **of** and **in** in whatever way you like, whereas:



will inevitably lead to questions like

- who **pays** the employee?
- **for** which department does the employee **work?**

These questions show the benefit of discussion: the knowledge of how a company is organised will be improved by every information confirmed or refuted. The possibility of misunderstandings through different interpretations is significantly lower.

1.3. Understandability

Understandability can be improved in two ways:

1) the reader of a documentation or program code understands it faster and better.
2) in order to find good names and terms, the author of a documentation must have understood the documented topic. He/she is no longer able to hide behind common, meaningless definitions.

---

[1] Here the term software element is used for everything which is given a name in the process of development (attributes, entities, functions, programs, relationships etc.)

These are only some major points of quality improvements. More justification for good names can be found in almost any book about software quality, so we will not go into deeper details of possible gains. This paper concentrates on *how to find good names* rather than on *why do we need good names*?

2. What is a *good* name?

There is a simple rule for a good name:

> A good name is as short as possible but as long as necessary.

The attribute *short as possible* is quite obvious and easy to follow (x, a, i23). So how do we define *necessary*?

Before we can know what is necessary for good understanding, we have to know how we understand names:

> A name should represent its semantics, the meaning of the *thing* being named. Whenever we read or hear a name, we immediately **associate** it with some experience we have made, we imagine some*thing*.

An association can be a set of examples, a range of values, a picture, a sound etc. The understanding depends always on the social, cultural and educational background of that person.

For example: Reading the sports pages in the newspaper: what possible values do *you* associate with **match result**?

If you thought for example of a value like 2:1 you are probably a football or icehockey player (or fan). If you prefer handball, you would think of something like 28:24 whereby a basketball player would imagine a result like 83:67.

A name can be said to be good, if the reader understands the same (or at least similar) *thing* as the author.[2]
There are 3 possibilities:

1. I understand what was meant ⇒ very good
2. I do not understand the name ⇒ not so good but at least no error
3. I understand something different ⇒ bad

There is a kind of measurement for the quality of names:

Reading or hearing a name leaves us with an impression of some meaning. Whenever we learn new information (e.g. new examples) about this *thing* our previous impression is confirmed or contradicted.

---

[2] The best variant is if you understand what was meant and not what was written ☺.

As a further means of measurement consider the following rule:

> the quality of a name is indirect proportional to our astonishment whenever we learn something new about the meaning of the name.

Examples:

Name	probable first idea of range of values
number	≥0
passenger-number	0 .. ca 10^2 to 10^4
bus-passenger-number	0 .. ca 100

Consider the values for the attribute bus-passenger-number such as:

27.73 or 32'415

Both values would destroy your first idea of a bus-passenger-number. They could for example match the meaning of names like bus-passenger-number-average and bus-passenger-number-total respectively.

In this example we can also see, how a more exact name leads to a narrower range of values and therefore to a more accurate understanding.

3. Rules for good names

In this section we will focus on the *communicative power* of names and <u>not</u> on their brevity. We believe that the process of finding good names is an essential one. This does not mean that the (often long) resulting names will be used in all following steps of development in their full length.

The science of linguistics refers to two specific types of names:

Referential names are artificial names, specially chosen to represent a well-defined semantic in a well-defined context.

Iconic or descriptive names are common, basic words which describe the *thing* they represent.

	referential names	descriptive names
examples	sweater	pullover
	result	output
	patates	pommes-de-terre
	Oracle V7	database
	iconic	self-descriptive
advantage	less misunderstandings because it is difficult to find another interpretation	a first time reader can understand it without knowledge of the whole context

disadvantage	the first time reader cannot understand the meaning without some background information	can lead to misinterpretation due to different context or background of reader. Suffers from the informality of natural language.

> Descriptive names are usually easier to read and should therefore be given preference. Whenever there exists a referential name, which in the given context or environment is well known and unequivocally defined[3], it should be used. The usage of referential names should also be considered where descriptive names could lead to significant misunderstandings

Often it is worthwhile to spend some time on finding a good descriptive name. If such a name is too long, then an abbreviation is justified. Usually people get used to abbreviations and they will become over time well-defined referential names.

Here a general rules for compound names:

> Good descriptive names are usually compound names. To improve readability, one should use a consistent rule to build compound names.

In German the rule for building compound names is easy. It follows the grammatical rule, that in a compound name, the last word defines the basic *thing*. In other languages this rule may be different or the usage of compound names is not as common as in German. But in every language there is certainly a way to build compound names of any length which are at least understandable.

An example for an (exaggerated) application of this rule:

The German
Bodensee-Dampfschiffahrt-Gesellschaft-Kapitäns-**Mütze**
is an exact definition of *Mütze*, i.e. "Cap".

In English compound words of this length are not good grammatical style, but they are comprehensible:
Lake-Constance-steam-boat-company-captain-**cap**

In French (as indeed also in Italian) a reversed word order is used, hence:
bonnet-de-capitaine-companie-bateaux-de-vapeur-lac-de-constance

The following example shows the difference derived from the sequence of names in a compound name:

line-number is a **number** of a **line** (e.g. 25)

whereas

number-line is a **line** of **numbers** (e.g. '1 4 56 123123 77')

[3] Caution: often referential names are thought of as well-defined. Only the search for a good iconic name can reveal that there are several, often conflicting *well defined* meanings.

3.1. Entities

Entities are concrete or abstract objects. An object name should describe a *thing* not a property. This means the immediate association should be an object with a couple of typical properties (attributes). For example:

The object	could make you think of
CAR	shape, colour, wheels, type, price ...
PERSON	age, sex, height, name, address, salary ...
ORDER	number, price, date, positions ...

Naming rule for concrete objects:

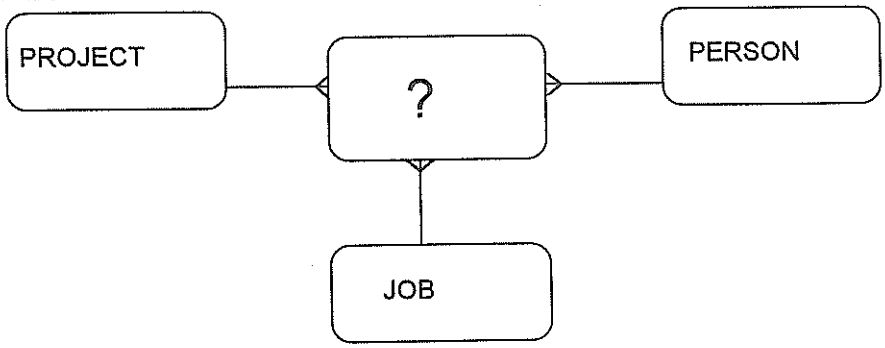
> Name the *thing* as accurately as possible.

Usually there is a suitable name defined in the dictionary:
CAR, ROJECT, PERSON, HOUSE

Naming rule for abstract objects (e.g. intersection entities):

> Use a 'natural' name if there is an established one (e.g. *assignment* for the connection of person to a job).
> If there is no 'natural' name, one should build an artificial one which tells the reader what the key components of the entity are.

For example:



In this example what should be the name of the intersection entity?
As the keys of "?" are the keys of the three other entities, we try:

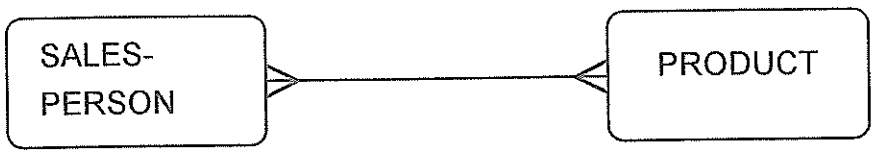
a) PROJECT-PERSON-JOB
b) PERSON-PROJECT-JOB
c) PERSON-JOB-PROJECT
d) JOB-PERSON-PROJECT
e) JOB-PROJECT-PERSON
f) PROJECT-JOB-PERSON

Applying the fule for compound names we decide whether we want to store information about a JOB (a and b), a PROJECT (c and d) or a PERSON (e and f). If the main property of this entity is the time spent doing a certain job on a project, PROJECT-JOB-PERSON (which can be read as a PERSON doing a JOB on a PROJECT) would be a good choice.
This procedure of finding a good name is not formal. It depends on what aspect of the entity you wish to emphasise. Hence choose the name which gives the best (closest) description and which is least likely to be misunderstood.

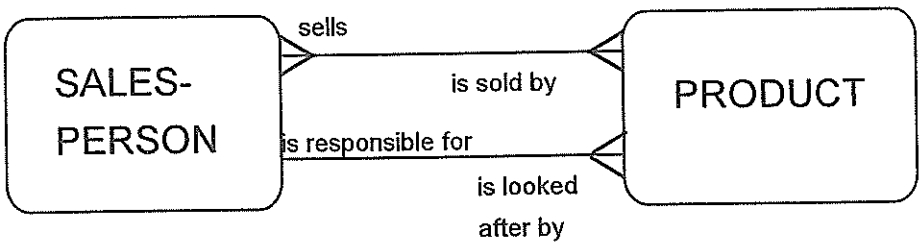
3.2. Relationships

The descriptions[4] of the relationships in an Entity-Relationship-Model (ERM) are the key to understanding the model and the underlying real world.

The diagram



tells you only, that a salesperson is somehow related with a product. Whereas



offers much more information. The descriptions reveal that there are two relationships of a different type.

The following rules for good relationship descriptions are proposed:

- there **must** be a verb. A relationship describes always a state or an activity.
- there should not be a noun in the description. A noun describes an object (entity!) and could therefore hide a transitive (non-normalised) relationship. It is usually possible to replace a noun by a participle or an adverb (e.g. owner of $\Rightarrow$ owning)
- avoid descriptions like *has, is of, is in*. There is almost always a better (more accurate) verb available. Relationships with this kind of generic description often show that the author did not understand the relationship or has not thought enough about it.
- avoid common, universally valid descriptions which are almost always true. Favourites are *is related to, is connected with*.
- describe a relationship in both directions and make sure both express the same semantics (cross-check!)

[4] Some methods *name* the relationships. But as two objects are always related by a verb, *name* is a bad term for it.

- describe the relationship consistently in such a way that you can read the ERM as you would a plain text[5]. The example above can be read as

  A salesperson sells many products
  A product is sold by many salespersons.

3.3. Attributes

An attribute is a property of an entity. A property is expressed by a single value.

An attribute name must be associatable with a value.

For example: age ⇒ 27
colour ⇒ red

For values which need a unit of measure the type of the unit should also be contained in the association.

For example

size could be in m, m^2 or m^3

better would be

length	⇒	m
area	⇒	m^2
volume	⇒	m^3
clothing size	⇒	no physical unit of measure

The rule for compound names is as important for attributes as it is for entities.

For example:

passenger-number	⇒	the number of passengers e.g. 65
expected-passenger-number	⇒	the number of passengers expected[6] e.g. 15'000
passenger-number-average	⇒	the average number of passengers e.g. 32.15

3.4. Functions

The term *function* is a very good example of a referential name with as many different definitions as there are environments.

---

5 There are different ways to describe relationships. One should not make a religion out of it. Choose a description and use it **consistently**.

6 There is a semantic difference between the *number of expected passengers* and the *expected number of passengers*. As the associated range of possible values is the same and passenger-expected-number sounds odd we can leave that one.

As the naming rules are different, we make the distinction between real functions in a mathematical sense (a piece of code which returns one single value) and programs, business functions, tasks, procedures ... in short every piece of code which does something to one or several objects.

3.4.1. Real functions

A real function used in its proper way returns one single value and is therefore used like an attribute.

> The naming rules for real functions are the same as the rules for attributes.

3.4.2. Business functions

We propose the following naming rules for procedures, business functions etc.

- A procedure **does** something to **something**. From that fact we deduct the rule that a procedure name must contain at least a verb and an object. If a procedure does a lot of things to a lot of objects, mention only the 1 or 2 most important activities (verb) and the 1 or 2 most important objects in the name
- Avoid universal verbs like *do, make etc.* Find a verb which narrows possible interpretations.
- Use the proper names (or abbreviations) as defined in a data dictionary for the manipulated objects.
- Do not hesitate to use long names. Calls to procedures are usually on a line of their own, so there is enough room to write a long name.

3.5. Abbreviations

As mentioned above, the usage of abbreviations is encouraged. Every abbreviation is a referential name of its own and is no longer subject to misinterpretation. It is a good technique to define abbreviations as soon as possible. Otherwise developers start to invent abbreviations of their own and this leads to chaos.

We propose the following rules for defining good abbreviations:

- Use standardised abbreviations like *num* for number, *emp* for employee or *abv* for abbreviation
  With this compound abbreviations can be built such as emp-num which are iconic referential names
- Define for one long name several abbreviations of standardised length
  For example: define an abbreviation
  2-3 characters long (to be used as a prefix)
  6-8 characters long (to be used where you have to save space)
  10-15 characters long (if the original is too long to be used in any circumstance).

- Forbid any usage of abbreviations not defined in the scope of the project.

4. Practical advice

4.1. What makes it so difficult to choose good names?

During my consulting activities to several projects I have identified several reasons why developers do not look for good names. I do not claim that this list is complete or applicable for every environment.

- looking for good names takes too much time.
  This is the usual excuse! Of course we do not make a lengthy party game of it. But if you are not capable of finding a good name within reasonable time, you have probably not really understood the problem. So the time is actually spent looking for the semantics and is therefore well-invested time.

- good names are too long
  Another good excuse! If typing time is relevant for the delivery date of your project, it is a funny kind of project anyway. There is no rule which forbids the usage of abbreviations.

- finding good (short) names is a creative task
  Creativity is a skill not everybody has as much as she or he would like to have. To be creative is hard mental work and it cannot be done all day or on demand.

- good names are accurate and require decisions
  To agree concrete, unequivocal meanings of *things* often demands decisions. Many people do not like making decisions. They prefer agreeing to a name whose meaning can be adjusted later.

- good names require full understanding of the *thing* they describe
  If you do not fully understand what you are describing or doing, you are only likely to find the correct name for it by accident.

4.2. How to improve the quality of names

Here is some advice on how the quality of names can be improved in a software development environment:

- teach the developers; to know the nature of good names is a precondition for choosing or finding them.

- make the quality of names a major topic during reviews and other quality checks. The more discussion there is about bad names, the more the developers will look for better ones.

- focus your quality assurance on the areas where the names are bad or ambiguous. There you will most likely find sources of problems, misunderstandings and errors.

- whenever you have to ask for the meaning of a name, discuss alternative names with the author. "What name could have avoided this question?"

5. Conclusion

Experience has shown, that knowing the concepts and rules of good names is half of the game. Once everybody in a development environment is **convinced** that better names lead to better software, the quality of the names will improve significantly.

I personally do not believe that following formal rules strictly for a creative task like finding good names is of great benefit. This process could only be formalised, when *good* could be formally defined. Maybe one day some linguists or informaticans will work on it, but it is questionable that there will be a big profit for the quality of software. A pragmatic approach is good enough to let us find adequately good names in a reasonable time.