A Language for Stating Component Quality*

Xavier Burgués, Xavier Franch
Universitat Politècnica de Catalunya (UPC),
c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)
{diafebus, franch}@lsi.upc.es

Abstract

We present in this paper a language for stating component quality in the framework of the ISO/IEC quality standards. The language consists of three different parts. In the first one, software quality characteristics and attributes are defined, probably in a hierarchical manner. As part of this definition, abstract quality models can be formulated and further refined into more specialised ones. In the second part, values are assigned to component quality basic attributes. In the third one, quality requirements can be stated over components, both context-free (universal quality properties) and context-dependent (quality properties for a given framework -software domain, company, project, etc.). Software components may be then selected by testing whether their behaviour with respect to the quality characteristic satisfy some quality requirements that model the context of selection. This gives some potential benefits in the software selection framework. We show how the language can be used through some examples.

Key words

Software Quality and Metrics, Non-Functional Requirements, Software Components

1. Introduction

Software quality models are used to determine to what extent software components (whatever the type of component is: object-oriented (OO) classes, Commercial Off-The-Shelf (COTS) packages, ERP (Enterprise Resources Planning) products, etc.) satisfy the requirements of a given context of use. This kind of acceptance test is crucial for assuring correct integration of software into applications and companies, and so a great deal of research has been done in the field.

As part of this research, some software-centered quality standards have been proposed [1, 2, 3, etc.]. Although each of them has its own specifities, some guidelines are common: a framework for the whole quality assessment process exist, software quality characteristics are identified and defined in a hierarchical manner, etc. We have studied one of these approaches, the set of ISO/IEC standards to software quality, in detail.

The standards collect usual quality-consumers needs expressed in terms of some high-level features of software, such as efficiency, reliability and others. However, a problem arises when the meaning of these attributes has to be defined and used accurately; usually, informal statements are used, and so the software quality model can be misunderstood. Therefore, incorrect evaluations can result, eventually yielding to rejections of correct components or acceptance of deficient ones.

* This work is partially supported by the spanish national research programme CICYT, ref. TIC97-1158.

We present here an approach aimed at lessen the risk of such misuses of quality models. It is centered on the definition of a language called *NoFun* (acronym for "NOn-FUNctional", meaning that software quality mostly refers to non-functional issues of software). The language consists of three different parts. In the first one, software quality characteristics and attributes are defined, probably in a hierarchical manner. As part of this definition, abstract quality models can be formulated and further refined into more specialised ones. In the second part, values are assigned to component quality basic attributes. In the third one, quality requirements can be stated over components, both context-free (universal quality properties) and context-dependent (quality properties for a given framework -software domain, company, project, etc.).

2. The ISO/IEC Standards for Software Quality

A set of ISO/IEC standards are related to software quality, being standards number 9126 (which is in process of substitution by 9126-1, 9126-2 and 9126-3), 14598-1 and 14598-4 the more relevant ones [1]. The main idea behind these standards is the definition of a *quality model* and its use as a framework for software evaluation. A quality model is defined by means of general *characteristics of software*, which are further refined into *subcharacteristics* in a multilevel hierarchy; at the bottom of the hierarchy there are measurable *software attributes*. *Quality requirements* may be defined as restrictions over the quality model.

The ISO/IEC 9126 standard fixs which are the characteristics at the top of the hierarchy: functionality, reliability, usability, efficiency, maintainability and portability. Furthermore, an informative annex of this standard provides an illustrative quality model that refines the characteristics as shows fig 1.

Characteristic	Subcharacteristics	Short definition
functionality	accuracy	provision of right or agreed results or effects
	compliance	adherence to application related standards or conventions
	interoperability	ability to interact with specified systems
	security	prevention to (accidental or deliberate) unauthorised access to data
	suitability	presence and appropriateness of a set of functions for specified tasks
reliability	fault tolerance	ability to maintain a specified level of performance in case of faults
	maturity	frequency of failure by faults in the software
	recoverability	capability of reestablish level of performance after faults
usability	learnability	users' effort for learning software application
	operability	users' effort for operation and operation control
	understandability	users' effort for recognizing the logical concept and its applicability
efficiency	resource behaviour	amount of resources used and the duration of such use
	time behaviour	response and processing times and throughput rates
maintainability	analysability	identification of deficiencies, failure causes, parts to be modified, etc.
	changeability	effort needed for modification, fault removal or environmental change
	stability	risk of unexpected effect of modifications
	testability	effort needed for validating the modified software
portability	adaptability	oportunity for adaptation to different environments
	conformance	adherence to conventions and standards related to portability
	installability	effort needed to install the software in a specified environment
	replaceability	opportunity and effort of using software in the place of other software

Fig. 1: ISO/IEC 9126 proposal of quality attributes refinement.

Some particular points of the standard could be matter of discussion. For instance, some issues (e.g., type and time of delivery –FTP, e-mail, CD, etc.- and economical cost –freeware, shareware, payment) are not dealt with during software evaluation; they are postponed by the standards to a managerial decision phase started after the evaluation itself. Anyway, we think that it is advisable to adhere to the standards to have a clear and widespread framework.

In order to evaluate these attributes, a *metric* must be selected and *rating levels* have to be defined dividing the scale of mesurement into ranges corresponding to degrees of satisfaction with respect to the attribute. The rating levels must be defined for each specific evaluation depending on the quality requirements. Finally, a set of *assessment criteria* combining the mesures of attributes are necessary to obtain the rating of the intermediate and top characteristics and, finally, the quality of the product.

Usually, this procedure is done in an informal, more or less structured way. However, we feel it is very well suited to be performed in a more formal manner, with the help of a language able to record this kind of definitions. This is the purpose of the NoFun language. In fact, the language presented here is the evolution of a previous version [4], focused on expressing non-functionality characteristics of O.O. classes. The new version of NoFun takes advantage over the old one not only by fitting better to the ISO/IEC standard (taking therefore functionality into account), but also by allowing the characterisation of a more general concept of component, as we will try to illustrate in the examples, and improving the expressive power of the language.

3. NoFun: A General View

To achieve the goal of formalisation, we basically provide three different kind of capabilities. First, there are many kind of modules to get the different kind of concepts defined in the standard. Second, values for these attributes may be given and bound to particular software components, the ones under evaluation. Third, additional constructs for representing quality requirements and assessment criteria are included.

Concerning the first category, there are three main of modules: characteristic, subcharacteristic, and attribute modules. Modules may import others, and also nesting is allowed. Nesting of modules allow to state auxiliary definitions. Following the standard, characteristic modules may not be defined one in terms of another. No such restrictions appear on the other types of modules, and so hierarchies of subcharacteristics and attributes may (and will) arise.

The upper part of fig. 2 shows an example of distribution of a quality model into modules. There are two characteristics defined in terms of four subcharacteristics. Following the ISO/IEC appendix, sharing of subcharacteristics between characteristics do not take place (although the language does not explicitly check this situation). Subcharacteristics do indeed form a hierarchy; they may depend on zero, one or more other subcharacteristics and attributes; a subcharacteristic may influence on more than one subcharacteristic. Last, attributes are defined at the bottom of the model; although also attribute hierarchies may be defined, they are not as usual as in the case of subcharacteristics. Attributes depending on others are named *derived attributes*, as opposited of *basic* ones, whose values must be explicitly computed.

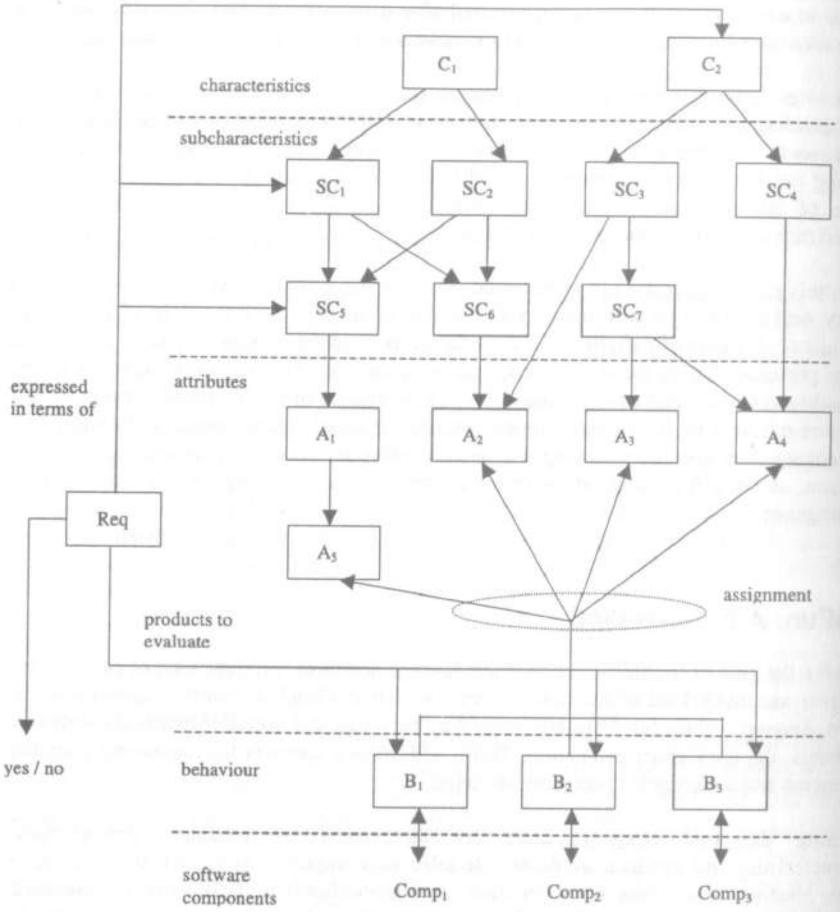

Fig. 2: Layout of a quality model in the ISO/IEC framework represented with NoFun.

Quality characteristics, subcharacteristics and attributes (hereafter, *quality entities*) are declared of a particular type. In addition to predefined types (called domains), mechanisms to introduce new ones are introduced. New types are introduced on top of domains; also new domains may be defined, encapsulated in yet another kind of module. Type constructs are rich enough to allow modelising the usual quality entities. There are sets, functions, tuples and sequences; their use is shown later in the examples.

Assignment of basic attribute values are encapsulated in new modules (*behaviour modules*), bound to the corresponding software components being evaluated. Behaviour modules are

abstractions of software components in the sense that they contain all the rellevant information for quality evaluation.

Last, quality requirements may be defined restricting the values of the quality entities. Assessment criteria can be seen as a set of quality requirements, and so we do not distinguish between them. Quality requirements are stated using operators over the quality entities, and they may be categorised depending on their importance. Requirements refer normally to characteristics and subcharacteristics, and rarely to attributes, due to their lower-level nature.

The rest of fig. 2 adds behaviour and requirement modules. The three software components under evaluation include a behaviour module measuring the basic attributes. Values of the attributes propagate up to the other quality entities (following the arrows in reverse direction). The requirement module containing assessment criteria for the evaluation refers in this case to one characteristic and two subcharacteristics. Here the result is simplified to be just success or failure, but we will see later that things are a bit more sophisticated, because of the categorisation of requirements.

In addition to these elements, an orthogonal concept is the one of refinement. Refinement allows to define quality models in an incremental manner, by specialisation of more general ones. This kind of inheritance-like relationship yields to a structured representation of quality; models can be formulated first in a general way, later refined in particular domains (OO classes, ERP products, bespoke software, etc.), and further specialised for companys, projects, etc.

The rest of the paper develops these elements in more detail.

4. Description of Domains

Domains play a central role in the definition of quality attributes. They are used to fix the type of these attributes, either directly or as part of a complex type definition (those using functions, sets and so on). NoFun has the usual predefined domains, that allow to use integer, real, boolean and string types in attribute definitions, but other types can be defined by enumeration of values.

Although we allow anonymous ones, domains are normally declared in *domain modules*, as shown in fig. 3. There appear two typical examples of domains. The first one enumerates some values of a (part of) a domain, i.e. the areas of a company where software has to be installed. The second one defines a scale of measurement. As it always happens in NoFun, an informal description of module contents is not only encouraged but required. Note that the second domain is declared as ordered. Values of ordered domains can be compared with less-than and greater-than relationships when stating assessment criteria.

```
domain module COMPANY_AREAS
    domain CompanyAreas
        explanation Areas or functions of a company
        defined as Commercial, Logistics, Manufacturing, HumanResources,
                   Accounting, Finances, Quality, Technical, Management Support
end COMPANY_AREAS

domain module UPPER_ADEQUACY_SCALE
    domain ordered UpperAdequacyScale
        explanation Provides a 5-value scale which penalises excessive coverage of features
        defined as NonExistent, Low, Excessive, Medium, High
end UPPER_ADEQUACY_SCALE
```

Fig. 3: Definition of domains.

5. Definition of Quality Attributes

Quality attributes are used in the ISO/IEC approach to measure basic software capabilities. We define them in *attribute modules*, which can contain many related attributes defined with the following information:

- Explanation of their purpose (mandatory). Explanation can be stated globally for some set of related attributes, or individually.
- Declaration of their type. Simple attributes will be declared of predefined types or using a domain. More elaborated declarations can be made using some type constructors: sets, functions, sequences and tuples. Involved domains must be imported in the module.
- Definition of their value. Just for *derived attributes*, i. e., those ones whose value depends on others' (which can be *basic attributes* –i.e., those ones whose value is computed explicitly- or derived, yielding to attribute definition hierarchies). Some language constructs can be used to build the definition.

Fig. 4 shows the definition of some attributes concerning component delivery. We focus on date and manufacturing company (others attributes such as price could be also considered). The month and the year of delivery are declared as integer attributes with some value restrictions (in the case of year, just lower bound is provided). Then, the date itself is declared as a tuple of two integers, defined as the values of the former attributes. Concerning the supplier, represented with a string, a special value is identified standing for the company itself.

```
attribute module DELIVERING_ISSUES
    explanation First, date of delivery of components
        attribute Month declared as Integer [1..12]
        attribute Year declared as Integer [1970..]
        attribute Date derived
            declared as Tuple(Integer, Integer)
            defined as (Month, Year)
    explanation Then, name of company delivering the product. "Own" states for
                software produced in the company
        attribute supplier declared as string special Own
end DELIVERING_ISSUES
```

Fig. 4: Definition of quality attributes for dealing with component delivery issues

Fig. 5 focus on the definition of two more ellaborated attributes for the ERP domains. The first one measures the degree of coverage of company areas by ERP products. It is declared as a function such that for every company area, a value from the given scale is assigned. It is necessary to import the domains defined in section 3, which become the domain and range of the function. A default value is also provided. In top of this attribute, a new one is declared to be the set of the company areas well-covered by specific ERP products. The elements of the set are declared then to be taken from the domain of company areas, and the attribute is computed in terms of the value of the previous function (i.e., it is derived).

```
attribute module ERP_ORIENTATION
    imports COMPANY_AREAS, UPPER_ADEQUACY_SCALE
    attribute AreaCoverage
        explanation Degree of coverage of company areas by an ERP product
        declared as
            function
                from CompanyAreas to UpperAdequacyScale
                default NonExistent
    attribute MainTarget derived
        explanation Company areas well-covered by an ERP product
        declared as
            set
                elements CompanyAreas
        defined as
                set of a in CompanyAreas such that AreaCoverage(a) = High
end ERP_ORIENTATION
```

Fig. 5: Definition of quality attributes for dealing with ERP products orientation

6. Definition of Subcharacteristic and Characteristic Modules

Last, we introduce subcharacteristic and characteristic modules, to capture all the concepts introduced in the ISO/IEC standard. Basically, (sub)characteristic modules just glue together quality attributes and subcharacteristic, either by directly putting them together in the module or by importing them; in the second case, subcharacteristic modules can be nested, but not characteristic ones, according to the standard definition.

Fig. 6, top, defines a subcharacteristic module for accuracy (as defined in fig. 1) related to ERP products, including many of the domains and attributes presented in previous sections. In this case all the quality domains and attributes are introduced inside the module itself. Fig. 6, bottom, outlines a definition of the functionality quality characteristic just by importing the necessary subcharacteristics. In both cases, definition just puts together the imported entities by means of a tuple. Note that the type declaration is not explicitly included; it may be inferred from the definition.

```
subcharacteristic module ERP_ACCURACY
    domain module COMPANY_AREAS
        domain CompanyAreas
        ... // as in fig. 3
    end COMPANY_AREAS

    domain module UPPER_ADEQUACY_SCALE
        domain ordered UpperAdequacyScale
        ... // as in fig. 3
    end UPPER_ADEQUACY_SCALE
    ... // other domains

    attribute module ERP_ORIENTATION
        imports COMPANY_AREAS, UPPER_ADEQUACY_SCALE
        attribute AreaCoverage
        ... // as in fig. 5
        attribute MainTarget derived
        ... // as in fig. 5
    end ERP_ORIENTATION
    ... // other attributes

    subcharactacteristic ERPAccuracy derived
        explanation Accuracy ISO/IEC subcharacteristic bound to the ERP domain
            defined as
                Tuple(MainTarget, ...)

end ERP_ACCURACY

characteristic module ERP_FUNCTIONALITY
    imports ERP_ACCURACY, ERP_COMPLIANCE, ERP_INTEROPERABILITY,
            ERP_SECURITY, ERP_SUITABILITY

    charactacteristic ERPFunctionality derived
        explanation Functionality ISO/IEC characteristic bound to the ERP domain
            defined as Tuple(ERPAccuracy, ERPCompliance, ERPInteroperability,
                             ERPSecurity, ERPSuitability)

end ERP_FUNCTIONALITY
```

Fig. 6: Definition of characteristics and subcharacteristics related to ERP products.

7. The Notion of Refinement

Although the constructs defined so far are well-suited for dealing with quality in a rather comfortable and reliable way, they suffer from a lack of adaptability in some senses. Let's consider the company areas domain. In fact, we have taken a strong decision when introducing the domain, fixing the concrete areas that the company is supposed to have. However, it is obvious that the division in areas will depend of the size of the company, its working domain and others. This definition thus can be useless in many cases.

Also, definition of derived attributes may vary depending on the context. For instance, one could relax the definition of main target allowing areas covered with a medium value. In this situation, the given definition will become invalid. In both cases, the solution would be introducing new but similar attributes reflecting this changes, exploding then the number of definitions and making our approach difficult to use.

To overcome this difficulty, we have introduced the related concepts of *abstract definitions* and *refinement*. Domains and derived attributes can be introduced as abstract, meaning that their definition is not provided in the declaration but elsewhere. Refinement allows to provide the definition of abstract domains and attributes, and also to redefine them. They are encapsulated in new packages which must be bound to the ones containing abstract definitions. Every appearance of an abstract item is labelled with the *abstract* key word[1].

Fig. 7 redefines the example of fig. 6 making the domain definitions abstract (top figure). Then, we provide a particular refinement for a concrete kind of company (middle figure; note that the binding is explicitly stated in the header) which makes explicit the areas but not the definition of the attribute, obtaining thus a partial refinement. We remark that the domain and attributes fully-defined in the abstract package must not be defined again; as a general rule of thumb, we do not repeat any previously given information, although the opposite is also allowed for understandability purposes. Last, we show the customization of the package for two particular companies; they give two different definitions of the attribute. Also note that the second refinement redefines the domain adding a new area.

We would like to stress the high degree of structurability that the refinement construct introduces in our approach. In our example, it is reflected by the fact that the ERP functionality characteristic defined in fig. 6, bottom, does not depend on the particular form that the subcharacteristics and attributes takes. In fact, we can say that we have a kind of polymorphic or generic definition of the characteristic, such that every particular refinement of its subcharacteristics and attributes implicitly produces a different definition.

8. Description of Software Components Quality

In order to be used in the evaluation framework provided by the ISO/IEC standard, software components must be measured with respect to the basic attributes that are rellevant to them. Many metrics have a straighforward measure because they can be computed directly from the information available of the component; this is the case of the attributes defined in *DELIVERING_ISSUES*. But often evaluation is a hard task, requiring well-defined and eventually complex methodologies; this is the case of the *AreaCoverage* attribute: accurate assignment of values in the rating levels used in its definitions is crucial for the whole scheme to succeed. This is the classical problem of quality evaluation, and obtaining results in this field falls outside the scope of this paper.

Once the evaluation of the attributes for a software component is obtained somehow, it just suffices with encapsulating them in a behaviour module as a sequence of assignments. The crucial point here is having a stable definition of the product. In other words, the set of

[1] It could be said that refinement is the equivalent to the OO concept of inheritance, although we prefer not to use this name because we are limiting the use of this construct to the context of refinement.

attributes rellevant to the component should be as fixed as possible; otherwise, the component should be examined over and over, every time new attributes are defined which must be taken into account. Then, the rule of thumb in this context is clear: before producing component quality evaluations, an exhaustive description of the domain of the component (e.g., real-time component, OO class, ERP product, etc.) must be done.

```
abstract subcharacteristic module GENERAL_ERP_ACCURACY
    abstract domain module COMPANY_AREAS
            domain CompanyAreas
                explanation ... // definition not included, because it is abstract
    end COMPANY_AREAS
    domain module UPPER_ADEQUACY_SCALE
            domain ordered UpperAdequacyScale
                explanation ...
                defined as NonExistent, Low, Excessive, Medium, High
    end UPPER_ADEQUACY_SCALE

    attribute module ERP_ORIENTATION
            imports COMPANY_AREAS, UPPER_ADEQUACY_SCALE
            attribute AreaCoverage
              explanation Degree of coverage of company areas by an ERP product
              declared as
                  function from CompanyAreas to UpperAdequacyScale default NonExistent
            abstract attribute MainTarget derived
              explanation Company areas well-covered by an ERP product
              declared as set elements CompanyAreas
    end ERP_ORIENTATION
end GENERAL_ERP_ACCURACY
```

```
subcharacteristic module LOWSIZE_COMPANY_ERP_ACCURACY
                                refines GENERAL_ERP_ACCURACY
    domain module COMPANY_AREAS
            domain CompanyAreas
                defined as Commercial, Manufacturing, Accounting, Finances
    end COMPANY_AREAS
end LOWSIZE_COMPANY_ERP_ACCURACY
```

```
subchar. module ACME_ERP_ACCURACY
    refines LOWSIZE_COMPANY_
            ERP_ACCURACY



    attribute module ERP_ORIENTATION
        attribute MainTarget derived
          defined as
            set of a in CompanyAreas such that
              AreaCoverage(a) >= Medium
    end ERP_ORIENTATION
end ACME_ERP_ACCURACY
```

```
subchar. module SPA3_ERP_ACCURACY
    refines LOWSIZE_COMPANY_
    ERP_ACCURACY
        domain module COMPANY_AREAS
            domain CompanyAreas defined as
                Commercial, Manufacturing,
                Accounting, Finances, Technical
        end COMPANY_AREAS
        attribute module ERP_ORIENTATION
          attribute MainTarget derived
            defined as
            set of a in CompanyAreas such that
                AreaCoverage(a) = High
        end ERP_ORIENTATION
end SPA3_ERP_ACCURACY
```

Fig. 7: Definition of abstract packages and their refinement.

Related to this problem, we are currently considering the possibility of having different perspectives of a software component. This is to say, a component could be involved in different quality models, each one with its own set of attributes; probably those sets would have non-empty intersections. But this is still future work...

9. Statement of Quality Requirements

Quality requirements are defined by the ISO/IEC standard as restrictions over the quality model. As such, they take the form of expressions in the language involving quality entities of the model. They are encapsulated in requirement modules, which contain a preliminary explanation of the intent of the requirement, and the list of individual related quality requirements. For each quality requirement, the following information is stated:

- Name.
- Informal explanation of the requirement.
- Enumeration of the quality entities which the requirement is defined upon. If all the requirements refer to the same module, a single declaration on the header suffices.
- Its definition, using operators bound to type constructs: universal and existencial quantifiers, set membership, etc.
- Its categorisation, which depends on the importance of the requirement during the evaluation process. We have identified four types of requirement categories: essential, important, advisable and marginal.

Quality requirements appear mainly in two contexts. First, as small units for establishing properties on quality entities. Properties may be more or less general depending on the abstraction of the model they are bound to. Fig. 8 shows two examples of such quality requirements. The first one is bound to a general quality model, and so it states a kind of universal property on the *AreaCoverage* attribute: an ERP must be addressed at least to one company area. The second requirement is more specific, referred to *DELIVERING_ISSUES* (see fig. 4) and bound to a particular company (as stated in the header): software made by the ACME company must not be dated before April 1998, which is the date the company started to use OO methodologies. Note the different categorisation of requirements: whilst the first one is labelled as essential, the second one is just classified as advisable: older products are still acceptable.

```
requirement module ORIENTATION_PROPS on ERP_ORIENTATION
   explanation Universal properties of ERP-orientation attributes
   definition
      univ-prop-orient-1: essential
         explanation ERP products must address at least to one company area
         defined as
            exists a in CompanyAreas such that AreaCoverage(a) > NonExistent
end ORIENTATION_PROPS

requirement module DATE_RESTRICTION on DELIVERING_ISSUES for ACME
   explanation States a requirement on the date of creation of ACME delivered software
   definition
      ACME-delivery-date: advisable
         explanation Software made by the ACME company must not be dated before April
                     1998, which is the date the company started to use OO methodologies
         defined as
            Supplier = Own implies
                  (Date.Year > 1998) or (Date.Year = 1998 and Date.Month >= 4)
end DATE_RESTRICTION
```

Fig. 8: Two examples of individual quality requirements.

Quality requirements also appear in the context of assessment criteria. From the ISO/IEC standard point of view, assessment criteria is just a set of quality requirements stated during the evaluation process. In this case, one or more requirements module (typically, one for characteristic or subcharacteristic) are necessary, each one containing a related set of requirements.

Fig. 9 offers an example of this situation. Five requirements concerning ERP products are collected in a single module. It must be said that these requirements have arosen in a real experience of selection of an ERP solution for a spanish company [5]. The first two requirements can be modelled using the quality entities presented so far; the other three use other subcharacteristics and attributes not introduced in the paper. The informal requirements reflect the information obtained from the company; the formalisation step helps sometimes to solve ambiguities, as it happens in the *req-func-2* requirement (the mapping from the informal to the formal requirements demands an exact meaning for "emphasize"). Classification is done according to the priorities expressed by the company.

An important feature appears in *req-func-4*: non-trivial requirements can be decomposed into others. This provides a comfortable way to structure the requirements keeping track of the original statement that generated them. The new requirements must have a priority less or equal than the old one and at least one of the new requirements must have the same priority than the old one.

Although it is out of the scope of the paper, it is worth mentioning that quality requirements can be used not only to check validity of software solutions, but also to select software components that fit well to those requirements. In this case, the language NoFun is used in the context of component selection. Some additional comments to these issue appear in the conclusions.


```
requirement module FUNCTIONALITY_REQ on ERP_FUNCTIONALITY for ACME
   explanation ... // name and location of the document that establishes the requirements
   definition
      req-func-1: essential
         explanation The selected ERP should cover all the company areas
         concerns ERP_ORIENTATION
         defined as
            for all a in CompanyAreas it holds that AreaCoverage(a) > NonExistent
      req-func-2: important
         explanation The selected ERP should emphasize commercial, logistic and
                     management areas
         concerns ERP_ORIENTATION
         defined as {Commercial, Logistic, Management} in MainTarget
      req-func-3: marginal
         explanation The company could adapt its structure to the new software if necessary
         concerns ERP_ADAPTABILITY
         defined as Adaptability.CompanyAdaptability >= None
      req-func-4: advisable
         explanation The selected ERP should be as open as possible, both for adding new
                     functionality and for interconnecting with other software
         concerns ERP_OPENESS
         decomposed as
            req-func-4-a: important
               explanation The selected ERP must satisfy a minimum degree of openess
               defined as Openess.bespoke >= Strong and Openess.COTS >= Strong
            req-func-4-b: marginal
               explanation The selected ERP must maximize its degree of openess
               defined as max(Openess.bespoke) and max(Openess.COTS)
      req-func-5: essential
         explanation The ERP should support Y2K, euro, ISO9000 and multicurrency
         concerns ERP_SPECIFIC_SUPPORT
         defined as {Y2K, ISO9000, Euro, MultiCurrency} in ERPSpecificSupport
end FUNCTIONALITY_REQ
```

Fig. 9: Two examples of individual quality requirements.

10. Conclusions

We have presented in this paper NoFun, a language for supporting the ISO/IEC quality standards as reported in [1]. The language consists of three parts: definition of the domain of discourse; definition of the quality elements; and establishment of assessment criteria. The language contains structuring mechanisms, type definition elements and other constructs that give an appropriate support for defining non-trivial quality models.

We consider that the salient features of our approach are:

- NoFun provides a basis for establishing quality models in a formal way, instead of using natural language. We think that NoFun is a step towards filling the gap of formal definition of quality characteristics and metrics; as mentioned below, to our knowledge just a few approaches exist in this sense.
- In addition to this, we have formulated our approach in the context of a main standard on software quality. As a result, our language provides a common framework that can be

used for people working in the field, sharing common results and building repositories with definition of characteristics, domains, evaluation of products, etc.

- The existence of a language with a well-defined semantics allows building support tools that can save human effort increasing also accuracy on the results. We currently have a prototype for doing software evaluation, based on the description of the domains, the assessment criteria and the evaluation of products, all of this described with NoFun.
- With respect to the expressive power, the language presented here has proved to be useful for defining a large kind of quality characteristics, criteria, etc. In particular, the many ways of defining quality (sub)characteristics and attributes, assessment criteria and specially the notion of refinement compare favorably to other related approaches we are aware of.
- Although we have focused on the ISO/IEC standard, the language can be used in other contexts related with quality. In addition to deal with other quality standards [2, 3], we can use the language as an Interface Description Language (IDL) for many contexts, such as for specification of non-functional issues [6], for enlarging existent IDLs such as the one for CORBA [7], and so on.

The language presented here is the evolution of the previous NoFun IDL [4]. Although many of the lowest-level constructs are similar, changes arise mainly concerning structuring mechanisms, the refinement notion and the way of establishing requirements. There are mainly three reasons behind this evolution

- Previous NoFun was specifically addressed to deal with small components, intended to contain definition of abstract data types implemented with usual data structures [8]. Therefore, the domain of application was component libraries such as LEDA [9], STL [10] and Booch [11] ones. There were many restrictive consequences of this situation; for instance, the notion of efficiency was specifically asimptotical efficiency, measured with the big-Oh notation, which is no longer useful in information systems of ERP products.
- NoFun stands for "NOn-FUNctional", in the sense that just non-functional issues were taken into account. Functionaly aspects were supposed to be covered with usual formal specifications languages, such as Larch [12], Z [13] and other similar ones. It is not the case for the current NoFun version (we are searching for a new language acronym...).
- Previous NoFun was not really bound to any quality standard. Although ISO/IEC could have been modelled with it (except for the functionality quality characteristic), the result would have been a little confusing. For instance, it offered just a structuring mechanisms for the so-called attributes, which has been split into three in the new NoFun.

There exist in the software community many approaches for dealing with software quality, although as far as we know, none of these approaches has been used in the particular framework of defining quality models. Instead, these proposals have been stated in the context of languages and notations for dealing with non-functional aspects of software. But in fact, also our language can be used this way (actually it has been the case up to now [4]), and so it makes sense to establish comparisons with them.

The most widespread approach is the one of the Toronto group, the NFR framework [14, 15]. NFR deal with non-functional requirements at the process level, that is, they use non-functional requirements to guide the software design process. As part of their proposal, they record design decisions with a design-oriented notation, which makes explicit the functional and non-functional goals of the system and their relationships, which can be of many kinds

(synergetic, contradictory, collaboration, etc.). Since the focus of our language has to be more the product than the process, we think that both approaches are really complementary.

Other approaches in at the product-level focus mainly on limited parts of the quality characteristics, or do not allow to represent all kind of attributes, characteristics and requirements. A great deal of the approaches are restricted to state just efficiency information of software components: asymptotic efficiency [16], efficiency of queries in relational structures [17], tight efficiency [18] and real-time efficiency [19]. A classical proposal in this field is the faceted approach of [20], which proposes a component classification scheme based on many dimensions. In all of these languages and systems, quality evaluation is restricted to check if components satisfy some restrictions about time efficiency, or to select or even generate components satisfying these restrictions. No way of defining arbitrary attributes or (sub)characteristics, neither sophisticated (but usual) non-functional requirements are provided. The notion of refinement does not appear in any of these approaches. And requirements such the one in fig. 9 cannot be stated at all.

A few words about experimental results. An experiment on defining a quality model for the selection of ERP products, based on a previous real case [5], has been developed successfully; some excerpts have been showed in the paper. Other previous, more academic work in the component-based software development also exist [21]. In this paper, we focus on tradicional non-functional attributes such as performance, reliability, etc. Currently, another case study is being developed in cooperation with a major spanish software manufacturing company, consisting on the classification of graphical forms, used to access data bases from automatically-generated applications. Up to now, forms were generated in an ad-hoc manner, making understandability and maintenance very difficult. NoFun is being used for defining user profiles (technical users, managerial users, secretary staff, etc.), each of them with different requirements on the generated forms. We are currently addressing the evolution of the form generation part of the tool to include NoFun. last, first steps on using the approach for the general problem of COTS packages acquisition [22] have been stated [23].

On the other hand, we are currently facing the problem of defining concrete metrics for evaluation quality attributes, which is an important part of the ISO/IEC standard. As a first an important case study, we are developing a methodology to measure functionality based on the definition of use cases (expressed in UML) centered on the processes taken over the application being measured. The importance of this line of research is obvious, since systematic methodologies will allow to assign values to attributes in a quite objective way.

11. References

[1] ISO/IEC Standards 9126 (Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their use, 1991) and 14598 (Information Technology – Software Product Evaluation: Part 1, General Overview; Part 4, Process for Acquirers; 1999).

[2] IEEE Computer Society. IEEE Standard for a Software Quality Metrics Methodology. IEEE Std. 1061-1992, New York, 1992.

[3] Rome Air Development Center (RADC). Software Quality Specification Guidebook RADC-TR-85-37, vol. II, 1985.

[4] X. Franch. Systematic Formulation of Non-Functional Characteristics of Software. Proceedings of International Conference on Requirements Engineering (ICRE) (Colorado Springs, USA). IEEE Computer Society, 1998.

[5] Sistach F., Pastor J.A. "Methodological Acquisition of ERP Solutions with SHERPA". First World Class IT Service Management Guide, tenHagenStam, March 2000.

[6] X. Franch. "Including Non-Functional Issues in Anna/Ada Programs for Automatic Implementation Selection". In Procs. International Conference on Reliable Software Technologies - Ada Europe'97, London (UK), June 1997, LNCS 1251, pp. 88-99.

[7] T.J. Mowbray, R. Zahavi. The Essential CORBA. John Wiley & Sons, 1995.

[8] X. Franch, P. Botella, X. Burgués, J.M. Ribó. "ComProLab: A Component Programming Laboratory". In Procs. 9th Software Engineering and Knowledge Engineering Conference (SEKE), Madrid (Spain), June 1997, pp. 397-406.

[9] K. Mehlhorn, S. Näher. The LEDA Platform of Combinatorial and Geometric Computing. Cambridge University Press, 1999.

[10] D.R. Musser; A. Saini. STL Tutorial and Reference Guide. Addison-Wesley, 1996.

[11] G. Booch, D.G. Weller, S. Wright. The Booch Library for Ada95 (1999). Available at http://www.pogner.demon.co.uk/components/bc.

[12] J.M. Spivey. The Z Notation. Prentice-Hall, 1993.

[13] J.V. Guttag, J.J. Horning. Larch: Languages and Tools for Formal Specification. Texts and Monographs in Computer Science, Springer-Verlag, 1994.

[14] J. Mylopoulos, L. Chung, B.A. Nixon. "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach". IEEE Transactions on Software Engineering, 18(6), 1992.

[15] L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, ISBN 0-7923-8666-3. October 1999, 472 pp.

[16] P.C-Y. Sheu, S. Yoo. "A Knowledge-Based Program Transformation System". In Proceedings 6th CAiSE, Utrecht (Holanda), LNCS 811, 1994.

[17] D. Cohen, N. Goldman, K. Narayanaswamy. "Adding Performance Information to ADT Interfaces". In Proceedings of the Interface Definition Languages Workshop, ACM SIGPLAN Notices 29(8), 1994.

[18] M. Sitaraman. "On Tight Performance Specification of Object-Oriented Components". In Proceedings Third International Conference on Software Reuse (ICSR), IEEE Computer Society Press, 1994.

[19] R.H. Pierce et al. "Capturing and verifying performance requirements for hard real-time systems". In Proceedings International Conference on Software Reliable Technologies, London (England), LNCS 1251, Springer-Verlag, 1997.

[20] Prieto-Díaz, R.: Classifying Software for Reusability. IEEE Software 4, 1. IEEE Computer Society, 1987.

[21] Authors. "Browsing a Component Library using Non-Functional Information". In Procs. International Conference on Reliable Software Technologies - Ada Europe'99, Santander (Spain), June 1999, LNCS 1622, pp. 332-343.

[22] N. Maiden, C. Ncube. Acquiring COTS Software Selection Requirements. IEEE Software, March 1998.

[23] X. Franch, J.A. Pastor. "On the Formalisation of ERP Systems Procurement". In Procs. Continuing Collaborations for Successful COTS Development ICSE Workshop, Limerick (Ireland), June 2000.