Secondary Publication

Henrich, Andreas

Repository based software cost estimation

Date of secondary publication: 05.03.2025

Accepted Manuscript (Postprint), Conferenceobject

Persistent identifier: urn:nbn:de:bvb:473-irb-1068990

Primary publication

Henrich, Andreas (1997): Repository based software cost estimation, in: Abdelkader Hameurlain und A Min Tjoa (Ed.), Database and expert systems applications : 8th international conference, DEXA '97, Toulouse, France, September 1 - 5, 1997 ; proceedings, Berlin u.a.: Springer, pp. 653–662, doi: 10.1007/BFb0022073.

Legal Notice

This work is protected by copyright and/or the indication of a licence. You are free to use this work in any way permitted by the copyright and/or the licence that applies to your usage. For other uses, you must obtain permission from the rights-holders.

This document is made available with all rights reserved.

Repository Based Software Cost Estimation

Andreas Henrich

Praktische Informatik, Fachbereich Elektrotechnik und Informatik,
Universität Siegen, D-57068 Siegen, Germany,
e-mail: henrich@informatik.uni-siegen.de

Abstract. One important problem with software development projects is to get an early and nevertheless accurate estimation of the software development costs. In the literature various methods have been developed for this purpose. The most popular examples are Boehm's *COCOMO*, Albrecht's *function-point method* or Sneed's *object-point method*. The two last-named methods are based on early results of the analysis phase, whereas COCOMO is based on an a priori estimation of the software size in *"lines of code"*. On the other hand, modern software development environments usually employ the object management facilities of a repository to store the documents created and maintained during software development. Hence, software cost estimation methods like the *object-point method*, which are based on early analysis results, can be implemented easily on top of the repository. In this paper we present such a repository based realization of software cost estimation methods.

1 Introduction

Most software development projects represent a considerable investment. Therefore it is essential to estimate the corresponding costs and benefits in advance, to decide whether the project should be realized or not and to control the project budget. With respect to the benefits different types of projects have to be distinguished. There are projects aiming for rationalization, projects aiming for a competitive edge, and projects necessary due to legislative regulations, to mention only a few possibilities. On the other hand, each project causes costs. The principal components of project costs are [11]: (1) hardware costs, (2) travel and training costs, and (3) effort costs (the costs for paying software engineers). Whereas a quantitative estimation of the project benefits depends on the concrete situation, hardware costs as well as travel and training costs are relatively easy to estimate. However, effort costs are harder to estimate, but fortunately various techniques for their estimation have been proposed:

1. *Percentage methods:* Cost estimation techniques falling into this category are based on some type of waterfall model assuming that the software development process is made up of a number of stages such as requirements specification, software design, and so on. The basic assumption is that the distribution of the effort costs over the phases is nearly constant, at least for projects dealing with systems of a similar type. This *"constant"* distribution

can be exploited to calculate an estimate of the effort for the whole project when the actual effort of the first phase is known.

2. *Techniques based on the size of the software product:* Two prominent representatives of this category are *Putnam's cost estimation model* [8] and *COCOMO* [2] (COCOMO = COnstructive COst MOdel). The basis for COCOMO, for example, is the formula $\text{Effort} = c_1(\text{KDSI})^{c_2}$, where KDSI is the number of thousands of delivered source instructions and c_1 and c_2 are constants which vary depending on the type of the project. To refine the estimation, a series of multipliers can be applied which take into account factors such as product reliability, database size, execution and storage constraints, personnel attributes and the use of software tools.
3. *Techniques based on analysis results:* In 1979 the *function-point method* was proposed by Albrecht [1] as an alternative to using code size as the basis for the cost estimation. The *function-point method* is based on an analysis of the functionality of the software rather than on its size. In the course of the stronger emphasis on the maintained data in analysis methods in 1990 Sneed proposed the so called *data-point method* [9]. This method is essentially based on the information contained in an entity-relationship diagram. Finally in 1996 Sneed presented the *object-point method* [10], which is based on the information gathered applying object-oriented analysis techniques.

For the applicability of these software cost estimation techniques a corresponding tool to automate the application of the techniques is needed. To this end, the cost estimation techniques can be realized based on the functionality of a repository. Roughly spoken a repository is a database system underlying a software development environment. The intention is that all tools store the documents created and maintained during software development in this open repository. In the present paper we describe the realization of a software cost estimation system based on a repository. This system employs the *percentage method* for first rough base estimations (cf. section 3) and the *object-point method* for a more sophisticated estimation (cf. section 4). We selected these techniques because — in contrast e.g. to COCOMO — most of the information needed for their application can be derived directly from documents in the repository. Before actually describing this cost estimation system, we have to introduce the repository underlying our considerations.

2 The Repository

The ideas presented in this paper have been developed and implemented based on a concrete environment consisting of H-PCTE [6] (an implementation of the OMS of PCTE) and the query language P-OQL [4].

PCTE (*Portable Common Tool Environment*) is the ISO and ECMA standard for a public tool interface (PTI) for an open repository [7]. As one of its major components PCTE contains a structurally object-oriented object management system (OMS). The data model of the PCTE OMS can be seen as

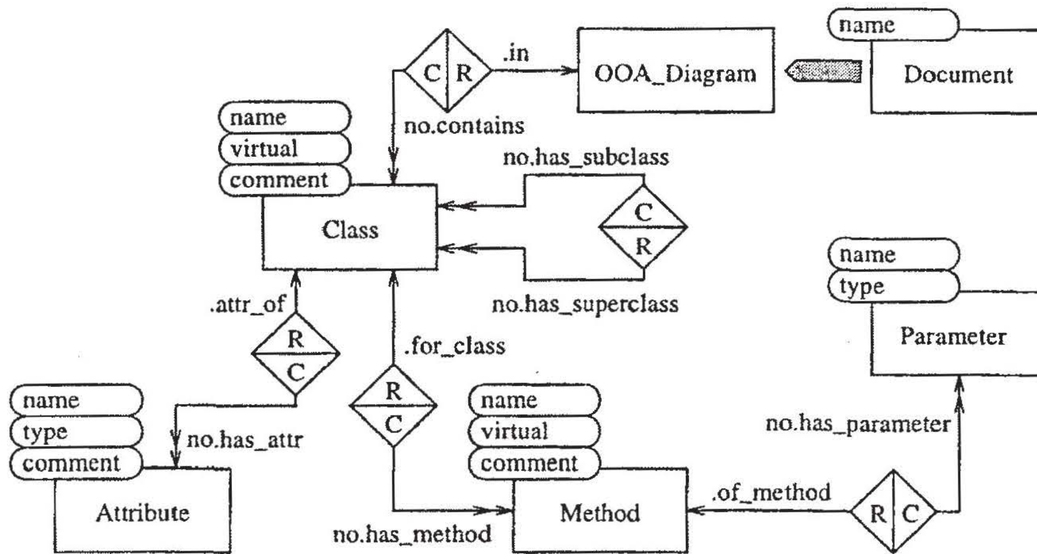

Fig. 1. Simplified example schema for OOA-diagrams

an extension of the binary Entity-Relationship Model. The object base contains objects and relationships. Relationships are normally bi-directional. Each relationship is realized by a pair of directed links, which are reverse links of each other, i.e. point into opposite directions.

The type of an object is given by its name, a set of applied attribute types and a set of allowed outgoing link types. New object types are defined by inheritance.

A link type is given by a name, an ordered set of attribute types called key attributes, a set of (non-key) attribute types, a set of allowed destination object types, and a category. PCTE offers five link categories: *composition* (defining the destination object as a component of the origin object), *existence* (keeping the destination object in existence), *reference* (assuring referential integrity and representing a property of the origin object), *implicit* (assuring referential integrity) and *designation* (without referential integrity).

Let us consider the example schema for OOA-diagrams in figure 1. As usual, object types are given in rectangles, attribute types are given in ovals, and link types are indicated by arrows. A double arrowhead at the end of a link indicates that the link has cardinality *many*. Links with cardinality many must have a key attribute. In the example the numeric attribute *no* is used for this purpose. For example the link type *contains* from *OOA_Diagram* to *Class* has such a key attribute and is hence described as *"no.contains"*. Therefore an instance of this link type can be addressed by its link name which consists of the concrete value for the key attribute and the type name separated by a dot – e.g. *"3.contains"*. A *'C'*, *'E'*, or *'R'* in the triangles at the center of the line representing a pair of links, indicates that the link has category *composition*, *existence*, or *reference*.

Finally the schema contains a subtype relationship between the object types *Document* and *OOA_Diagram* which is indicated by a broad shaded arrow.

P-OQL [4, 5] is an OQL-oriented [3] query language for PCTE. A query in P-OQL is either a select-statement, or the application of an operator like *sum*.

Assume that we search for the name and the applicable methods of all classes in the OOA-diagram named *"Clearing"*. This corresponds to the P-OQL query:

```
select C:name, normalize C:[_.has_superclass]*/_.has_method/->name
   from D in OOA_Diagram, C in (D:_.contains/->.)
   where D:name = "Clearing"
```

In the **from-clause** of this query two base sets are defined: Base set `D` addressing all OOA-diagrams and base set `C` addressing all objects which can be reached from the actual object of base set `D` via a path matching the regular path expression "`D:_.contains/->.`". In this path expression the prefix "`D:`" means that the actual element of base set `D` is used as the starting point of the definition. "`_.contains`" means that exactly one link of type *contains* must be traversed. Here the underscore ("`_`") is used as a wildcard denoting that arbitrary key attribute values are allowed. "`/->`" is used to address the destination object of the path. In addition "`->`" can be used to address the last link of the path. The "`.`" at the end of the regular path expression means that the object (or link) under concern is addressed. It is also possible, to address an attribute – which is for example the case in the target-clause of the above query – or a tuple of values (see [4] for more details).

The **select-clause** of the example query states that a multi-set of pairs is requested. Each pair consists of the name of the class and a set containing the names of the methods which can be applied to instances of the class. Since methods defined for superclasses of the class under concern can also be applied to instances of the class, we have to address these methods as well. To this end, the regular path expression "`C:[_.has_superclass]*/_.has_method/->name`" is used in the query. The starting point of this expression is the actual element of base set `C`. The meaning of "`[_.has_superclass]*`" is that zero or more links matching the link definition "`_.has_superclass`" have to be traversed. Alternatively P-OQL knows the iteration facilities `[` *path_definition* `]+` to indicate that a path matching *path_definition* has to be traversed at least once and `[` *path_definition* `]` to indicate that a path matching *path_definition* is optional. After traversing an arbitrary number of *has_superclass* links, exactly one link of type *has_method* is traversed using the link definition "`_.has_method`". Finally the *name* attribute of this method is addressed using "`/->name`". The resulting multi-set is transformed into a set using the unary operator `normalize`.

In addition to the link definitions used in the above example P-OQL allows the specification of a set of link categories with the meaning that all links having one of the given categories fulfill this link definition. E.g. the expression "`[{c, e}]+/->.`" addresses all objects which can be reached via a path consisting only of links with category *composition* or *existence*.

3 Implementation of the Percentage Method

To apply the percentage method the schema given in figure 1 has to be extended. Figure 2 illustrates these extensions. Whereas the introduction of an object type

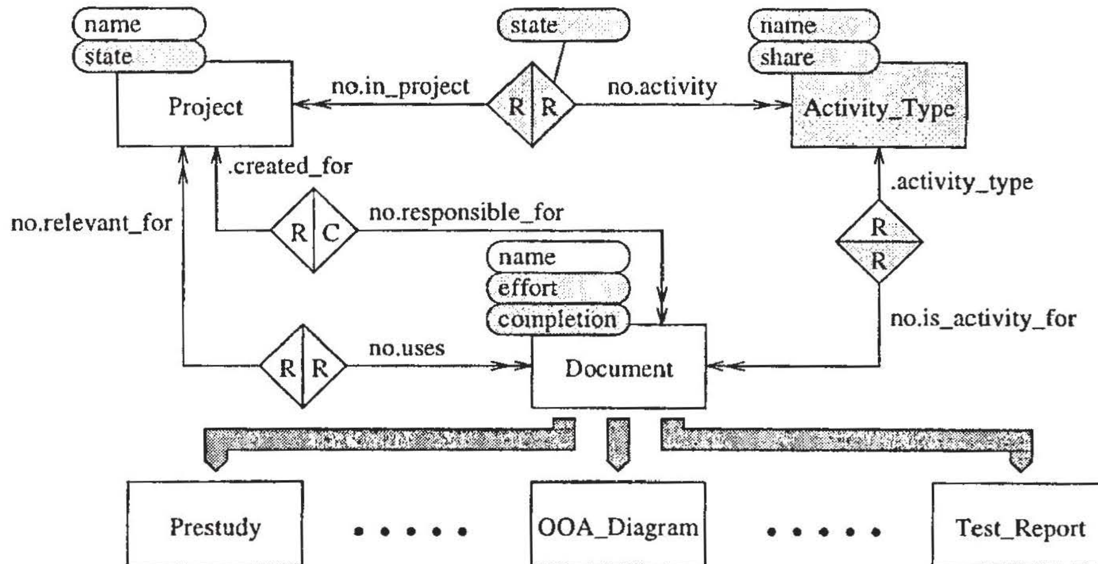

Fig. 2. Schema for the application of the percentage method

Project with corresponding link types is necessary for arbitrary project oriented cost estimation techniques, the densely shaded object types, link types and attributes are specific for the percentage method.

To represent the different activities (or phases) in a software development project the object type *Activity_Type* is introduced. The objects of this type represent the stages of the waterfall model on which the percentage method is based. Each activity type (such as prestudy or requirements analysis) has a *name* and a *share*. The *share* represents the average portion of the effort spend for the activity. The link type *activity* from *Project* to *Activity_Type* has a non-key attribute *state* representing the state of the activity in the project. Each document is linked to exactly one activity type by a link of type *activity_type*. Furthermore, for each document the *effort* spend up to now and the degree of *completion* (0 to 100 %) are maintained. Finally the *state* attribute for projects represents the state of the project as a whole.

To apply the percentage method for a concrete project we first have to fill the *share* attributes for the *Activity_Type* objects. If there are no finished projects in our repository, the corresponding values can be estimated by experts. Otherwise they can be extracted from the repository applying the following query:

```
select AT:name, AT:.,
      (sum (select Doc1:effort
              from Doc1 in (AT:_.is_activity_for/->.)
              where Doc1:.created_for/->state = FINISHED)
       / sum (select Doc2:effort
               from Doc2 in Document^
               where Doc2:.created_for/->state = FINISHED))
   from AT in Activity_Type
```

For each activity type this query yields a triple with the name of the activity

type, an object reference for the corresponding object in the repository, and the average share. The object reference is requested using a dot as the second component in the select-clause of the query. This object reference can be used to set the value of the attribute *share* of the activity type. The average share is calculated employing two sub-selects. The first sub-select yields the effort for all documents in finished projects which are associated with the activity type under concern. To this end, in base-set `Doc1` all documents are addressed which can be reached from the activity type under concern via a link of type *is_activity_for*. In the where-clause of the sub-select these documents are restricted to those contained in finished projects. Finally the `sum` operator is applied to the result of the sub-select summing up the efforts dedicated to the activity type under concern. The second sub-select does very much the same, but the base-set here addresses all documents irrespective of the associated activity type.

When the share attributes for the activity types are set, the effort of all projects with at least one finished activity can be estimated by the following P-OQL query:

```
select P:name,
       (sum (select Doc:effort
                  from Doc in (P:_.responsible_for/->.)
                  where Doc:.activity_type/->name
                        <= (select Act1:/name
                                 from Act1 in (P:_.activity->.)
                                 where Act1:state = FINISHED))
        / sum (select Act2:/share
                     from Act2 in (P:_.activity->.)
                     where Act2:state = FINISHED))
   from P in Project
   where P:state = RUNNING
     and 0 < sum (select Act3:/share
                       from Act3 in (P:_.activity->.)
                       where Act3:state = FINISHED)
```

In base-set `P` the query considers all running projects for which the share of at least one finished activity is greater than zero. This second condition is necessary to prevent divisions by zero in the select-clause of the query. To state this condition, we employ a sub-select addressing the finished activities of the project under concern. To be more precise, in base-set `Act3` we address the links of type *activity* originating from the project under concern by the regular path expression "`P:_.activity->.`". In the where-clause of the sub-select we check whether the *state* attribute applied to the link has the value FINISHED, and in the select-clause we address the share attribute of the destination object by the expression "`Act3:/share`". Finally these *share* values are summed up.

For each project fulfilling the conditions the name and the estimated effort are reported. The estimated effort is again calculated employing two sub-selects.

In the first sub-select the documents associated with the project under concern are addressed in base-set `Doc` using the regular path expression

"P:_.responsible_for/->.". In the where-clause of the sub-select we check if the document under concern is associated with an activity type which is already finished for the project under concern. This test is performed applying a further sub-select, which determines the names of all activities which are finished for the project. In this case the relation "<=" stands for the subset relation, because it is applied to (multi-)sets.

The second sub-select is exactly the same as the sub-select used in the where-clause of the query.

4 Implementation of the Object-Point Method

The *object-point method* uses so called *object-points* as a measure for the size of object-oriented software, and hence, as a measure for the effort costs caused by the development of this software. Sneed bases his method on three sub-models: an object model, a communication model and a process model. The estimation of the effort costs is then performed in three steps:

A measure for the *coding effort* is derived from the classes in the object model. To this end, for each class the number of attributes, the number of relations, the number of methods and the novelty are considered. The coding effort is estimated by so called *class-points*:

$$\text{Class-Points} = ((\text{attributes}) + (\text{relations} \times 2) + (\text{methods} \times 3)) \times \text{novelty}$$

For the estimation of the *integration effort* the message exchange described in the communication model is considered. For each message the number of parameters, the number of sources, the number of destinations, the complexity (*low* = 0.75, *medium* = 1.0 or *high* = 1.25), and the novelty are considered:

$$\text{Message-Points} = ((\text{parameters}) + (\text{sources} \times 2) + (\text{destinations} \times 2)) \times \text{complexity} \times \text{novelty}$$

Finally, the process model is employed for the estimation of the *system testing effort*. System processes (multiplier 6), batch processes (2), online processes (4), and realtime processes (8) are distinguished. In addition the complexity, and the number of variants is considered for each process:

$$\text{Process-Points} = (\text{process type} + \text{variants}) \times \text{complexity}$$

The aggregate effort measure *object-points* is derived as follows:

$$\text{Object-Points} = \text{Class-Points} + \text{Message-Points} + \text{Process-Points}$$

The effort estimation derived in this way is then refined by considering a range of factors representing the quality requirements (such as reliability, time efficiency, or portability) and project attributes (such as the technical support, the reliability of the network, or the employed methods). Let QRM represent the multiplier for the quality requirements and PAM represent the multiplier for the project attributes. Then the *adjusted-object-points* can be determined as follows:

$$\text{Adjusted-Object-Points} = \text{Object-Points} \times \text{QRM} \times \text{PAM}$$

According to Sneed, one Adjusted-Object-Point corresponds to a development effort of approximately 0.25 person-days. Of course this relation has to be verified and customized for each company.

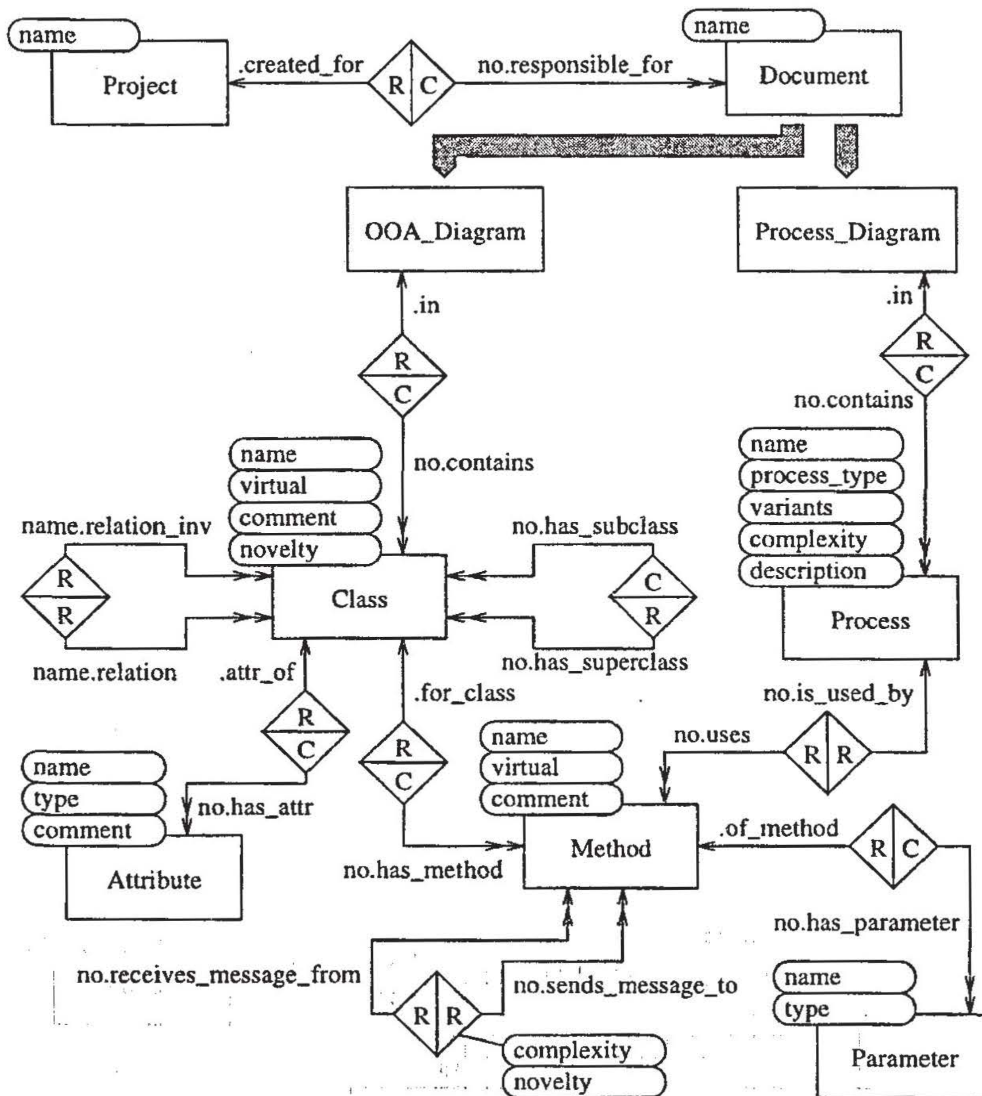

Fig. 3. Schema for the application of the object-point method

In the following we assume that the employed software development environment contains tools to create and edit the object model, the communication model and the process model, and that these tools store the data in the repository in accordance to the schema depicted in figure 3. In this schema classes have an additional attribute *novelty*, which is 1.0 for classes which are completely new and 0.0 for classes for which approved implementations exist. In addition instance relationships between classes are represented by links of the type *relation*.

To represent Sneed's communication model, we have introduced the link types *sends_message_to* and *receives_message_from* for methods. Analogously to Sneed links of type *sends_message_to* have an enumeration attribute *complexity* and a float attribute *novelty*. However, in contrast to Sneed in our data model a message is always considered in the context of a sending and a receiving method. As a consequence a message has always exactly one source and one destination.

Messages which would have multiple sources or destinations in Sneed's model correspond to multiple links in our model. Hence, for our schema Sneed's formula for the Message-Points simplifies to

$$\text{Message-Points} = ((\text{parameters}) + 4) \times \text{complexity} \times \text{novelty}$$

Finally, in our schema the process model is represented by a further subtype of the object type *Document*. From this subtype *Process_Diagram* the process descriptions can be reached via links of type *contains*. The objects of type *Process* have associated attributes *name*, *process_type*, *variants*, *complexity* and *description*. Furthermore the relationship to the employed methods is represented by links of type *uses*.

Based on this schema the following P-OQL query determines the object-points for a project named *"Clearing"*.

```
(sum                                                    // add Class-Points
   (select ((count C:_.has_attr->.
            + ((2 * count C:*.relation->.)
               + (3 * count C:_.has_method->.)))
           * C:novelty)
      from P in Project, C in (P:[{c}]+/->.)
      where P:name = "Clearing" and C:. is of type Class)
 + (sum                                                 // add Message-Points
      (select ((count M:/_.has_parameter->. + 4)
               * (M:novelty
                  * (case M:complexity = LOW      ==> +0.75,
                          M:complexity = MEDIUM   ==> +1.00,
                          M:complexity = HIGH     ==> +1.25)))
         from P in Project, M in (P:[{c}]+/_.sends_message_to->.)
         where P:name = "Clearing")
   + sum                                                // add Process-Points
      (select (((case Pc:process_type = SYSTEM   ==> 6,
                      Pc:process_type = BATCH    ==> 2,
                      Pc:process_type = ONLINE   ==> 4,
                      Pc:process_type = REALTIME ==> 8)
                 + Pc:variants)
               * (case Pc:complexity = LOW      ==> +0.75,
                       Pc:complexity = MEDIUM   ==> +1.00,
                       Pc:complexity = HIGH     ==> +1.25))
         from P in Project, Pc in (P:[{c}]+->.)
         where P:name = "Clearing" and Pc:. is of type Process)))
```

In this query the results of three sub-queries determining the class-points, the message-points and the process-points are added up. In each sub-query a base-set `P` addressing all Projects is used. In the where-clause this base-set is restricted to the project named *"Clearing"*.

In the sub-query determining the **class-points** all components of the project under concern are addressed via the regular path expression "`P:[{c}]+/->.`" in a second base-set `C`. In the where-clause this base set is restricted to classes. For each class the number of attributes, the number of relations, and the number of

methods are determined counting the outgoing links of the corresponding types. The numbers derived in this way are weighted according to Sneed's formula and multiplied by the *novelty* of the class.

To calculate the **message-points** all *sends_message_to* links in the complex object forming the project under concern are addressed in a base-set `M`. To this end, the regular path expression "`P:[{c}]+/_.sends_message_to->.`" is used. This path expression starts traversing one or more composition links. Finally a link of type *sends_message_to* has to be traversed and this link itself is addressed using the notation "`->.`". For the messages represented by these links the number of the parameters of the destination method it determined using the expression "`count M:/_.has_parameter->.`". The number determined in this way is then multiplied with the novelty and the complexity of the message. To convert the enumeration type attribute *complexity* into a numerical value, the `case` statement of P-OQL is used.

Finally the **process-points** are determined analogously to the class-points.

In principle the query described above is used in our software cost estimation tool which proceeds as follows: (1) The user of the tool has to enter the name of the project. (2) The object-points are determined by a P-OQL query corresponding to the above query except that the project name *"Clearing"* is replaced by the actual project name. (3) The result of the query is presented in a result window. In this window the user can add the quality requirements and the project attributes needed to calculate the adjusted object-points. (4) The adjusted object-points are calculated when a corresponding button is pushed.

References

1. A.J. Albrecht. Measuring application development productivity. In *Proc. Application Development Symposium*, pages 83–92, Philadelphia, Penn., USA, 1979.
2. B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.
3. R. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Mateo, Cal., USA, 1993.
4. A. Henrich. P-OQL: an OQL-oriented query language for PCTE. In *Proc. 7th Conf. on Software Engineering Environments*, pages 48–60, Noordwijkerhout, 1995.
5. A. Henrich. Document retrieval facilities for repository-based system development environments. In *Proc. 19th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 101–109, Zürich, 1996.
6. U. Kelter. H-PCTE: A high-performance object management system for system development environments. In *Proc. 16th Annual Intl. Computer Software and Applications Conf.*, pages 45–50, Chicago, Ill., USA, September 1992.
7. Portable Common Tool Environment - Abstract Specification / C Bindings. Standards ECMA-149/-158, 3rd edition and ISO IS 13719-1/-2, 1994.
8. L.H. Putnam. A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering*, 4(4):345–361, 1978.
9. H.M. Sneed. Die Data-Point-Methode. *ONLINE, ZfD*, (5):48, May 1990.
10. H.M. Sneed. Estimation of the development costs of object-oriented software. *Informatik-Spektrum*, 19(3):133–140, 1996. in German.
11. I. Sommerville. *Software Engineering*. Addison-Wesley, 4th edition, 1992.