Approach to comparing complex software implementation methods

Přístup k porovnání metod implementace složitého software

I. Vrana[1], J. Vrána[2]

[1]*Czech University of Agriculture, Prague, Czech Republic*
[2]*Komix a.s. Prague*

Abstract: Some of agriculture-food sector information systems are characterised by a high complexity and a large size. There are often many alternative solutions or technologies (implementation methods) and more then one possible way or approach to an information system design. Individual alternatives could considerably differ by their properties, e.g. costs of design of initial functionality, development and operational costs, run-time costs and technical parameters of the resulting information system (e.g. the access time). Unfortunately, existing metrics for quantification of this task usually do not deliver precise results but a rough estimate depending on many variable conditions. The paper will outline typical implementation methods and show approaches to assessment and comparison of certain types of properties of information systems for a computer support for management of large data systems, which use relational database. Authors presented part of these results also at the Agrarian Perspectives conference 2004 in the Applied Informatics session (Vrana I., Vrána J. 2004).

Key words: Information system, implementation method, quality assessment

Abstrakt: Informační systémy musí poskytnout uživateli požadovanou funkcionalitu při zpracování určitých datových struktur. Tohoto cíle lze zpravidla docílit různými alternativními postupy nebo technologiemi (implementačními metodami). Jednotlivé alternativy se mohou podstatně lišit svými výslednými vlastnostmi, např. náklady na vytvoření požadované funkcionality, náklady na rozvoj systému, náklady na provoz i technickými vlastnostmi výsledného systému (přístupovými dobami, apod.). Složitost návrhu a dosažení požadovaných vlastností silně závisí např. na rozsahu systému. Kritická situace může nastat u rozsáhlých systému, které jsou typické i pro některé aplikace z agropotravinářského komplexu, např. u složité sítě obchodních řetězců. Příspěvek se bude zabývat metodami porovnávání některých vlastností informačních systémů používajících relační databázi. Část zde uvedených výsledků již autoři prezentovali na mezinárodní konferenci Agrární perspektivy 2004 v sekci Aplikovaná informatika (Vrana I., Vrána J. 2004).

Klíčová slova: informační systém, implementační metoda, hodnocení jakosti

INTRODUCTION

Let us consider the development of a software application dedicated to management of large and complex systems as e.g. a food distribution chain. There exist a number of ways (implementation methods) of achieving the required functionality of the software application, see e.g. Aho (1974, 1983), Greene (1981). The typical implementation methods will be described later in this paper. There also exist a number of possible criteria for evaluating a quality of computer applications and information systems. Suitability of the certain criteria depends on the nature of the solved problem (of the application domain). The objective of this paper is to describe a method, which could help us to improve the following aspects of applications created for management of large systems:

- Costs of development of initial functionality of the application,
- Costs of maintenance and extension of functionality of the application,
- Runtime efficiency of the developed application

The term *costs* represents here the total demands of the application development. It need not have necessarily a direct projection to costs expressed in terms of money, it is rather some generalized rate of complexity, a rough labour demand, etc. of the examined part of the system. To convert these generalized costs to a real economic value, several other aspects (i.e. price of a labour work, experience of members of the development team, etc.) should be considered.

The above-mentioned three aspects have a fundamental influence on the (business) success of the

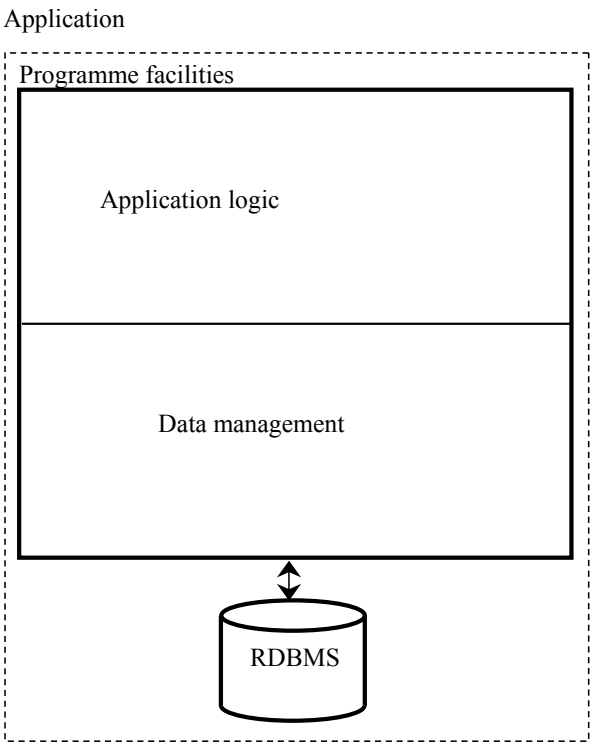

Figure 1. Schematic picture of an internal structure of the developed software application

created application. The first two aspects represent direct economic factors of the development process and the third aspect has a principal influence on practical usability (or non-usability) of the created application. We shall not consider the hardware side of the application, because the objective of this work was to improve the software of the application.

Let us consider that the developed application uses a relational database and it has its internal structure according to the schematic picture in Figure 1.

Application logic implements all algorithms of the so-called business logic of the application and it is independent on the way of data storing. At the other hand, data management implements algorithms for storing and manipulation of the maintained data. This splitting of the considered application, particularly ratio between the size of application logic and the data management parts, and the sharpness of the boarder between them, will enable us to better describe the properties of individual methods of management and storing data.

RELATIVE ASSESSMENT OF QUALITY OF COMPUTER APPLICATIONS

We shall outline in this paragraph the basic aspects and parameters to be considered in order to assess the quality of a computer application and compare its materialisation by different implementation methods.

Initial development costs – IDC

Initial development costs (costs of initial development of application functionality) can be considered as total costs needed for creation of the initial functionality of the prospective application. Initial costs could be further split to costs of the individual phases of the application development lifecycle (see e.g. Flaatten 1991): an *analysis* of the problem, a *development and design* of a solution, an *implementation* and, at the end, a *testing*. The developed application passes individual phases in various loops, when some activities should be iteratively repeated within a certain phase, or it repeats the whole phase or several phases, respectively. For simplicity, we shall consider a direct model when all phases were passed only ones without any loops in development of initial functionality. When comparing properties and features of methods for data storing and data management, the activities within the above-mentioned development phases could be further decomposed with respect to partial costs due to storing and processing of data and partial costs due to other aspects of the developed application, e.g. a user interface, communication features, an integration with other applications, etc. We shall focus only on the part of activities dealing with storage and processing of the managed data.

Initial phases of the lifecycle should be implementation-independent, when the respective analysis is not burdened by any implementation constraints or decisions. Costs of analysis of the application domain depend mainly on the complexity of the application domain itself and they are almost independent on the way of a design and an implementation of the application. Because an implementation method does not substantially influence the costs of the analysis, the analysis need not be considered for comparison of prospective implementation methods.

Costs of the development phase considerably depend on the complexity of the problem domain itself, but also on the used implementation technologies. The costs consist namely from identification and design of all possible branches of application functionality and of a design of its logical algorithms.

It is possible to quantitatively distinguish two types of design works with respect to whether it is needed to design in detail a number of smaller branches of functionality (according to the used method) each of them solving one branch of application functionality

(a concretised action or operation), or whether the design has a character of creation of more universal instruments, which will be capable to solve a broader range of application functionality branches.

In the first case, the design has a character of creation of rather greater number of more-or-less repeating simple algorithms and it is possible to achieve almost the implementation using proper design instruments, like various CASE tools.

In the second case, we should first create the mentioned universal mechanism and then develop its parametric description allowing it to achieve the desired functionality. The development in this case has the character of creation of a smaller number of more complex algorithms and then of working out an abstract parameterisation. The abstractness of the designed algorithms and their parameterisation is proportional to their expected generality and universality.

In spite of difficulties with the objective quantification of the design phase, the coarsest quantification is the amount of the spent time. But this measure does not take into account the character of the task, the experience of designers, etc. The character of the task can be taken into account e.g. by means of so-called *coefficient of complexity*, which grows with the increased complexity and by which the rough quantification should be multiplied. No direct method exists for obtaining correct values of this coefficient, but the concrete values are usually determined on the basis of experience.

Costs of the implementation phase depend on the used implementation method still more distinctively than in the design phase. In contrast to the previous design phase, the costs of the implementation phase could be quantified e.g. with regard to the scope (number of lines of a programme code). But as before in the design phase, also here it is necessary to take into account the variable complexity of created programme code with regard to use of different implementation methods.

Costs of the testing phase have a similar dependence on the used implementation method as two preceding phases had. It is possible to quantify costs of this phase (for the purposes of comparison of different alternatives) e.g. by the number of programme branches, which should be checked and tested in individual alternatives. A different complexity of tested programmes, when using different implementation methods, could be again expressed by the coefficient of complexity.

Costs of the whole development of the initial functionality can be obtained as the sum of costs of given activities during individual phases of the development lifecycle.

Maintenance and further development costs – MDC

Most of applications pass more than one passage through the main cycle of development of a certain new compact functionality. The literature, which deals with development and design of software projects (e.g. Horowitz 1978, Kingston 1990, Wirth 1976 etc.), presents many good reasons, why any software product, which is larger than a school task, cannot be created during one development cycle. It is necessary to take this fact into account when realizing any software work and to count with it since the first phases of a lifecycle, when strategic decisions concerning used technologies were adopted. The used implementation technology can radically influence labour consumption of the following development cycles of this software work.

Equally as in the previous paragraph, which dealt with costs of the development of the initial application functionality, costs can be decomposed to the costs of individual partial phases (analysis, design, implementation and testing) also in the case of maintenance and mainly in the case of extending an overall functionality. Analogically with the previous paragraph, we shall separate the costs associated with storing and processing of data from costs associated with other parts of the development process.

Similarly as in the case of development the initial functionality, also here the phase of analysis depends above all on the complexity of the problem itself and is almost independent on the used implementation method. That is why it can be again neglected.

The design and elaboration of the solution is a very important phase for comparing costs and labour input of different implementation methods. Costs and labour input of this phase distinctively depend on the selected implementation method and on character of intended changes and on extending of functionality. Small changes of functionality have a rather *additive character*, i.e. adding a new functionality without changing the existing one. Then the dependence of costs on selected implementation method need not be strong. In the case when proposed changes of functionality might have a stronger influence also on the existing functionality or they have a *multiplicative character* (when a small change of functionality caused a multiple growth of complexity), then costs and labour input are strongly dependent on the used implementation method.

Although the quantitative assessment of the design phase is rather difficult, the relative size of areas touched by some change with respect to the total size of the application can serve as one of possible

criteria. The detected extent of changes should be still corrected by the coefficient of complexity, as mentioned in previous cases.

Costs and labour input for implementation of changes depend directly on the implementation method. In a non-appropriately chosen implementation method, each change of functionality of the whole application can cause changes in functionality of the data management part. But in other cases, majority of later modifications need not project itself to the data management part.

Costs and labour input for debugging and testing of a new functionality are proportional to the size and character of changes, which happened in the previous step. Thus, the same rules hold for this phase as for the phase of implementation.

Total costs of the maintenance and extension of application functionality are again the sum of costs of individual phases of the lifecycle. The later phases have the main contribution to the total costs.

Runtime efficiency

Previous two paragraphs dealt with economic aspects and measures, which should be taken into account in considerations concerning a technology used for creation of the application and also concerning the question, how these measures depend on the used implementation methods. The aspect depicted in this paragraph has also its economic dimension, but its principal dimension is the usability or non-usability of the created application in real operational conditions.

When thinking about a runtime efficiency of the created application, or of data storing and management, respectively, it is necessary to examine requirements, which each method lays on the storing of the expected volume of input data. On the basis of the way of data storing used by the particular implementation method, it is then still necessary to examine the principal complexity of the most often used operations, which would be performed on the data (selective selections, mass selections, comparison, modification, etc.). When considering the capacity and computational complexity of the compared methods, it is still necessary to take into account not only requirements on data storing, on processing, on the volume and complexity considered in the beginning of the design, but also the dependence of the mentioned features on the possible increase of the volume and complexity of input data. Underestimation of the runtime efficiency might bring considerable troubles for the future real use of the developed application and can also ruin invested resources and means.

Evaluation of aspects

The necessary step in assessment of individual aspects is to choose some descriptive formalism that allows to quantify the problem itself and also to quantify the compared methods. According to the selected observed aspects, we had to quantify costs of the initial development of application, costs of its maintenance and extending and modifying its functionality as one type of quantitative aspects. The runtime performance of the developed application should be considered as another type of quantitative aspect.

The first mentioned type of quantification is based on the character and the quantity of the program code necessary to fulfil the required functionality of the application. The more program code is necessary, the higher are the costs for its initial development and further maintenance. On the other hand, the more complex the program code is, the more qualified (and probably the more expensive) development staff is needed. From another point of view, the more general and universal the program code is, the easier and cheaper is its adapting to different functionality and implementing changes, with respect to the program code designed tightly to the given problem.

The second mentioned type of quantification is based on the quantity of raw data stored in the database and mainly on the type of operations used to access and manage the data. The more data (data rows) have to be stored in the database, the more time is probably needed to store and maintain it. On the other hand, the better and more efficient operations are used to store and manage the data, the more efficient this management is.

In order to evaluate the amount and character of the program code, we first have defined the mapping between individual required functionalities of the application logic (application logic branches) and between the different branches of the program code that implements the functionality. We also have defined some groups of changes to the application logic and consequently to the program code that may be necessary to implement. The groups were:

– Additive changes, i.e. changes that imply only adding some new parts to the existing application logic or program code or cause only a small scale and localized change to the existing application logic and the program code.
– Multiplicative changes, i.e. changes that imply massive extension or change to a considerable part of the application.

Having this grouping, then it is possible to estimate the character of changes in different stages of the application lifecycle (initial development, further extend-

ing, maintenance, deploying to another domain, etc.) for applications developed using individual compared development methods. An important aspect that acts against the raw work complexity (the raw requirements to create or modify given number of program code branches and lines) is the character of the code. It should be, therefore, distinguished whether the code is straightforward and easy to mentally manage (and thus not requiring any specially qualified development staff) or contrary, whether it is complex with complex internal relations difficult to mentally manage and thus requiring the ability of an abstract thinking.

In order to evaluate the runtime efficiency of the developed application, it is necessary to estimate the raw volume of data (number of data rows) that need to be stored in the database. A formal set description applied to the data model (even very complicated) provides a good apparatus for estimation of data that should be stored and further processed. Making a list of the most typical and the most frequent operations with data is the next step in quantifying and estimating the runtime efficiency of the application. For example, the list of operations can contain:

– Selection of one value of one attribute
– Selection of all historical values of one attribute
– Selection of actual values of all attributes of one row
– Selection of all valid combinations of attributes in one row
– Selection of actual values of all attributes in a parameter table
– Etc.

The raw volume of data itself usually does not cause any big problem, because the contemporary relational databases can handle huge volumes of data. But the different implementation methods employ different algorithms for selected operations and this can imply differences in theruntime efficiency by many orders.

CONSIDERED IMPLEMENTATION METHODS OF DATA STORING AND MANAGEMENT

For purposes of the basic comparison of properties of various alternatives of implementing software application, we shall consider the following three implementation methods of data storing and management:

– General object method
– Fixed data structure
– Dynamic relational data storing

The main idea of the **general object method** is to achieve maximum accuracy of the stored model with possibility to store the model of the subject system in full generality and in the natural form as individual objects and associations between them. The main objective of this approach is:

– To store logical model of the subject system in the full generality
– To enable maximum independence of application on the concrete application domain and
– To enable maximum flexibility of application with respect to changes of the application domain

The general object method is universal and capable to store and process an arbitrary system of objects by means of a rather simple transformation. The application logic significantly dominates over data management in the programme facilities (Figure 1). That is why the part of the application for data management is relatively simple. There is also a clear boarder between both parts of the application thanks to the universality of the transformation. In this case, the boarder has the shape of universal functions of the type:

– Select values of one attribute of certain object
– Select previous (historical) values of the given attribute of certain object
– Select values of all attributes of certain object
– Select all neighbour objects to the certain object
– Modify value of the selected attribute of the selected object
– Modify associations between objects
– Recognize history of selected associations
– Etc.

The main advantage of the general object method is its generality and universality. The application development complexity is proportional mainly to the scope and complexity of the application domain. Complexity of all development phases rapidly decreases in the later phases of the application life. It is also relatively easy to implement the developed application to another application domain.

A considerable disadvantage of the general object method is a poor operational efficiency of applications based on this method. This feature limits the potential usability of such applications only to the application domains of rather small size. The capability of abstract thinking of all project-team members and non-existence of suitable development tools for creation, management and debugging of a logical model of the application domain are another disadvantages of this method. But these disadvantages are usually less important with respect to the poor operational efficiency of the created applications.

The general object method is suitable only for management of application domains of small size but with a possibly variable data structure.

The main idea of the **fixed data structure** method is to achieve maximum operational efficiency of the

created application. Storing and management of the logical model of the subject system with use of the fixed data structure consists in building a primary transactional information system in a "classical" way. This way and approach of building primary information systems is sufficiently described in numerous references dealing with building information systems and designing database structures. Therefore, we shall mention only the most important characteristics and will introduce relationships with approaches used in other compared methods of data storing and management.

In the case of the fixed data structure, the data part of the subject system is described by a normalized E-R model (consisting of entities and their relationships). Entities are then stored as tables in the host relational database. The subject system is functionally described by a set of algorithms, which process the individual entities and their relationships. Implementation of the developed application (information system) in another application domain is not usually supposed.

The main objective of this method is:
– To utilize all possibilities of the host database engine
– To enable maximum operational efficiency of applications based on this method using relational processing of stored data.

The data management part occupies a considerable part in programme facilities and the boarder between the application logic and data management is not sharp (Figure 1). The fixed data structure method has a great specificity with respect to the given application domain. This causes a lower flexibility but higher operational efficiency. The structure of the host database for application based on the fixed data structure reflects the subject application domain, it has the character of mutually interconnected application-specific tables and usually it is impossible to generalize it. But thanks to this arrangement, it is possible to optimise individually the branches of data management and to achieve maximum performance and efficiency. With use of the potential of relational technology, the following types of requirements can be effectively solved:
– Select all attributes of the given entity
– Select all historical combinations of attribute values of the given entity
– Select all attribute values of the given entities set or all entities
– Select all neighbour entities to the given entity
– Compare attribute values for the given entities
– Find history of the selected relationships
– Modify values of selected attributes of the given entity
– Modify relationships between the entities
– Etc.

The main advantage of the fixed data structure method is achieving maximum operational efficiency of developed application by means of utilisation of properties of the host relational database engine and by means of the individual optimisation of each operation. It dedicates this method for applications, which have huge data amounts in their database.

Another advantage is a relatively easy development of the initial functionality of the application. Although a raw laboriousness is proportional to the scope of the application domain, it is possible to use the existing development tools and also the experience of designers who are used to development of "classical" information systems and this development does not require any abstract work.

The principal disadvantage of the fixed data structure method is its rigidity towards changes of structure of the application domain. Even though the development of an initial functionality is relatively easy, there is a high difficulty of later adjustments, which still tend to increase with growing number of branches of application logic. This disadvantage is vital for the future life of the developed application.

The fixed data structure method is suitable **only** for application domains, which have a relatively little diverse structure, which moreover practically does not change. But an application domain can be very large as to the data volume.

The general object method and the fixed data structure method represent two extreme alternatives of the implementation method. Numerous compromise alternatives exist between these two extremes, each of them can approach one or the other extreme and can resolve some of their main weaknesses and try to preserve their advantages to some extent. **The dynamic relational data storing method** (Vrána 2003) is such a compromise alternative, which tries to achieve maximum flexibility (with respect to changes of the application domain) whilst keeping a high operational efficiency taking advantage of the maximum utilisation of properties of the host relational database engine. Analogically as in the case of the general object method, the logical model describes the data structure of the application domain. This logical model is transformed to a similar form to the logical model of the application domain used by the fixed data structure method, i.e. to the system of entities and relations. The transformed logical model is stored in a general form of metadata in a special separated part of the host database and becomes a substantial element of the application behaviour. It is its meta-description or meta-

programme. Based on the stored meta-description, a relational structure of specific tables and associations is dynamically generated in the host database. This structure practically corresponds to the structure of specific tables and associations firmly created by the fixed data structure method. The dynamically created specific relational structure is further queried by SQL queries, which are dynamically composed from the given request and the meta-description. Processing of the SQL query can fully utilize properties of the host relational database engine.

In this way, the dynamic relational data storing method combines advantages of both extreme approaches in such a sense that in every moment data is stored in the fixed "native" relational data structure in which they can be effectively processed but it is always possible to change the data structure and thus the behaviour of all application simply by changing the meta-description without any intervention to the programme code. Therefore, the main objective of this method is:

– To store the logical model of the subject system in full generality
– To enable maximum independence of applications on the concrete application domain
– To enable maximum flexibility against changes of the application domain;

and simultaneously:

– To store data of managed system utilizing all possibilities of relational technology of the host database engine
– To enable maximum operational efficiency of applications utilizing relational processing of the stored data.

Application based on the dynamic relational data storing has a structure, which is similar to applications based on the general object method. Also application logic dominates over data management in programme facilities and the boarder between these two parts is sharp (Figure 1). Interface between them is created by general functions of the type:

– Select values of certain attributes of all objects, which satisfy the given criteria
– Select all objects, which are neighbours to objects satisfying given criteria by associations with the given properties
– Modify values of selected attributes of the specified objects
– Modify associations between objects
– Determine history of selected associations
– Etc.

General operations of the SQL language provide the mentioned basic functions to the superstructure application logic, i.e. an arbitrary entity filtering, entity projection, entity joining, modification of values, etc.

The dynamic relational data storing combines the main advantages of both extreme methods and eliminates their disadvantages. Its generality enables to create applications, which are very general and independent on the concrete application domain, which also have very little requirements on further maintenance and extension. These features predetermine utilisation of the dynamic relational data storing method in application domains, which rapidly and often change their structure. Once developed applications are still capable of a relatively easy implementation in other application domain.

By principle, relational data storing and processing moreover gives to these applications excellent properties from the point of view of operational efficiency and predetermines this method for utilisation in the data and structure very extensive large application domains.

The main disadvantage of the dynamic relational data storing is an excessive abstractness of algorithms of applications and necessity to develop and maintain metadata. Both these disadvantages manifest themselves in increased requirements on the qualification and skills of the development team.

The dynamic relational data storing is a method, which is very suitable for building an application dedicated to manage a large and quickly changing application domain. Once developed application can be with little costs implemented in much a smaller or less dynamic application domain.

COMPARISON OF DESCRIBED IMPLEMENTATION METHODS

Let us briefly summarise and compare the basic properties of the described three methods for data storing and management. For simplicity, we shall denote them as GOA, FDS and DRD, which stand for the General Object Method, the Fixed Data Structure Method and the Dynamic Relational Data Storing Method. Concrete properties of individual methods depend on character of an application domain. Therefore, for the purpose of this comparison, let us consider a management of large and complex technological network of a mobile phone operator as a uniform application domain for all considered methods. Besides aspects of data complexity, i.e. number of records in the host database needed to store a subject data system and a ratio of number of data records with respect to metadata records, we

shall also consider the aspects of operational efficiency and overall performance. We are particularly interested in time, during which the system based on a certain method is able to process a certain type of a query.

For comparison of the discussed implementation methods we shall use the following criteria, which are split into three groups:

- Aspects related to application development
  - Initial development costs (IDC)
  - Maintenance and further development costs (MDC)
- Aspect, which express static complexity of the application
  - Metadata overhead (MDO)
- Aspects, which reflect dynamic complexity of application (runtime efficiency)
  - Complexity of selection of one value of one attribute (SASV)
  - Complexity of selection of all historical values of one attribute (SAMV)
  - Complexity of selection of actual values of all attributes of one row (MASV)
  - Complexity of selection of all valid combinations of attribute values in one row (VCA)
  - Complexity of selection of actual values of all attributes of the given parameter table (AVA)

This list contains still further aspects in addition to the basic ones described in the charter Relative assessment of quality of computer applications. The overall overview of properties of individual methods in their mutual relationships is depicted in the radar diagram in Figure 2.

The semantic of individual axes of the graph in Figure 2 (or individual assessment aspects) is such that the smaller the value is (closer to the centre) the better is the given implementation method in this aspect. Since individual aspects are not equally important, the bold axes indicate the most important aspects.

It clearly follows from the graph, that the GOA method has the best properties in IDC and MDC aspects, which express the development complexity of applications (i.e. complexity of the initial functionality development and of maintenance). In all aspects SASV, MASV, SAMV, VCA and AVA, which express the runtime efficiency (particularly in cases of the most frequently used operations) the GOA method exhibits fatal imperfections, which practically exclude it from any real use.

In contrast to the GOA method, the second compared FDS method exhibits excellent properties mainly in runtime efficiency. For the FDS method, the vast majority of operations could be performed in an easy and efficient way. But this high operational efficiency is paid for by a high development complexity IDC and particularly a high complexity of maintenance and of further evolvement MDC. The FDS method is also practically disqualified due to these shortcomings.

It is clearly seen from the graph that the DRD method is just slightly worse than the FDS method in runtime efficiency and at the same time, the DRD method is better by orders than the GOA method in these aspects. In aspects of development complexity, the DRD method is just slightly worse than the GOA method (which is the best in these aspects), but the DRD is far better than the FDS method. The DRD method asymptotically keeps excellent properties from both

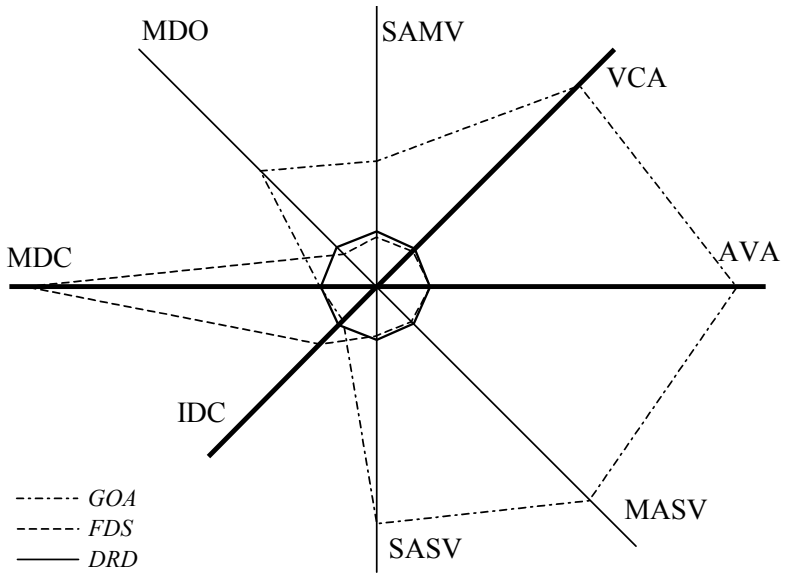

Figure 2. Radar diagram graphically describes properties of individual implementation methods

compared extreme methods and also eliminates their very poor properties, which practically disqualify these extreme methods from real utilisation in the considered type of critical applications.

CONCLUSION

The described method of assessment is suitable for evaluation of basic qualitative properties of several alternatives of solution to the software applications. The good solution can be distinguished from the poor solution in this way. Some parameters could be only a qualified estimate. Similarly to other methods, this method does not provide an exact quantitative expression of properties of individual alternatives. Therefore, it does not suit for a fine recognition of the quality of similar solutions. But the accuracy of our assessment method and also properties of the DRD method itself and related theoretical results of the comparison were proven in commercially successful large applications including data warehouse for the statistics of pension assurance in the Czech Republic and the application supporting configuration management of the mobile phone network of the Czech greatest mobile operator Eurotel (Vrána 2002). The same approach can be used in many other application domains, e.g. in:

– Management of large and complex technological networks
– Management of a distributed controlling system
– Management of large trading and food supply chains
– Modelling and management of distributional and pipelining networks
– Modelling and management of trading and logistic networks
– Modelling, management and exploring dependencies in economic and other processes
– Etc.

REFERENCES

Aho V. (1974): The design and analysis of computer algorithms. Addison-Wesley.

Aho V. (1983): Data structures and algorithms. Addison-Wesley.

Flaatten P.O. (1991): Foundation of Bussiness Systems. Andersen Consulting, The Dryden Press; ISBN 0-03-076481-5.

Greene D.H. (1981): Mathematics for the analysis of algorithms. Boston, Birkhauser.

Horowitz E. (1978): Fundamentals of computer algorithms. Potomac, Md, Computer Science Press.

Kingston J.H. (1990): Algorithms and data structures: Design, correctness, analysis. Sydney, Addison-Wesley.

Vrána J. (2002): Methods of employing metadata for managing large systems. DATESO'02 –Proceedings of Workshop on Databases. Ostrava: 44–56; ISBN 80-248-0080-2.

Vrána J. (2003): Dynamic relational data storing method for management of large data systems. [Doctoral dissertation.] Czech University of Technology, Prague.

Vrana I., Vrána J. (2004): Metody porovnávání vlastností informačních systémů (Methods of comparison of properties of information systems). Agrarian perspectives, CUA Prague.

Wirth N. (1976): Algorithms + data structures = programs. Englewood Cliffs, N.J., Prentice-Hall.

Arrived on 3rd January 2005

Contact address:

Prof. Ing. Ivan Vrana, DrSc., Česká zemědělská univerzita v Praze, Kamýcká 129 00, 16521 Praha 6-Suchdol, Česká republika
tel.: +420 605 474 137, e-mail: vrana@pef.czu.cz
Dr. Ing. Jan Vrána, Komix a.s., Holubova 1, 150 00 Praha, Česká republika
tel.: +420 604 366 463, e-mail: vrana@komix.cz