# Identification of Versions in Databases of Software Components *

Pavol Návrat, Mária Bieliková
Slovak University of Technology, Ilkovičova 3, 812 19 Bratislava, Slovakia
E:mail: {navrat, bielik}@elf.stuba.sk
WWW: http://www.elf.stuba.sk/~navrat
http://www.dcs.elf.stuba.sk/~bielik

## 1 Introduction

Evolution of software systems during all stages of development and maintenance requires some sort of management of different versions (variants or revisions) of software components as well as of whole systems. Software configuration management (SCM) involves applying various methods, possibly supported by specialized tools. One special, but very frequent situation is that the development of a software system is supported by some CASE tool. In such a case, all kinds of documents (not only source text units, but also their descriptions, specification diagrams etc.) are stored in a database (repository) that the CASE tool operates upon. On the one hand, the fact that all the components are available in a database (i.e., in electronic form with a defined organization) creates a very favourable condition for managing their versions in an automated way. On the other hand, application of SCM methods and tools in a CASE supported development is connected with several problems.

In the paper, we analyze some of the problems connected with integrating SCM with CASE. We concentrated mainly on configuration identification as one of the most fundamental SCM processes [1]. We propose a technique of identifying versions of software system process models as described by data flow diagrams. We report on a first experience with applying the technique.

## 2 Versioning

CASE tools support various activities during the software life cycle, but the analysis and design are probably the ones that receive assistance most often. Here, most notably the creation of diagrams (techniques of semiformal description) that form various views of a system (e.g., functional model, data model, time model) is supported by a tool. Most of the tools use abstraction i.e., they allow forming hierachies of diagrams. For each kind of diagrams (e.g., data flow diagrams, data model diagrams, data structure diagrams, entity life history diagrams, state transition diagrams), there are various typical kinds of entities that are to be identified in each particular diagram. For example, to complete a data flow diagram (DFD) in the Systems Engineer CASE tool, it is necessary to work with the following kinds of entities: `DFD Set` (Process Model), `DFD Process`, `DFD Store`, `DFD External Source`, `DFD Flow`, `DFD Object Instance` (Process Store or External Source), `DFD Flow Instance`, `DFD Diagram (Picture)` [8].

If we stay with the example, working with diagrams frequently means changing them i.e., changing one or more entities participating in it. Technically, each modification of any of the entities involved could be considered as forming a new version of the diagram. Practically, adopting this view would lead to a dramatic explosion of the number of versions, managing of which would become nearly impossible. Moreover, most of them would not be meaningful versions deserving being recorded from the point of view of the user. From this example we see that in any approach to SCM it is very important first to deliberate carefully what entities are going to be considered as configuration items (granularity version control).

We can envisage three rough levels of granularity. When maximal granularity is chosen, configuration item corresponds to an entity in the CASE database. In particular, when working with a process model described by data flow diagrams, processes, data stores, data flows, external entities, but also diagrams and hierarchies of diagrams all would be configuration items.

When medium granularity is chosen, configuration item corresponds to single diagrams, screens, modules. A version includes also entities at hierarchically lower levels. It is important that a version can be identified at any level of the hierarchy.

When minimal granularity is chosen, configuration item corresponds to an entity at the top level of the hierarchy. In particular, when working with a process model described by data flow diagrams, a complete hierarchy of data flow diagrams starting from the top (i.e., a context diagram along with all the diagrams and entities included in it) is one configuration item.

With no strict and precise rules to formulate and subsequently to follow, generally the medium granularity is often the most appropriate choice. It offer the biggest flexibility contrary to the minimal granularity, when it is necessary to form after every single modifi-

*The work reported here was partially supported by Slovak Science Grant Agency, No. G1/4289/97.

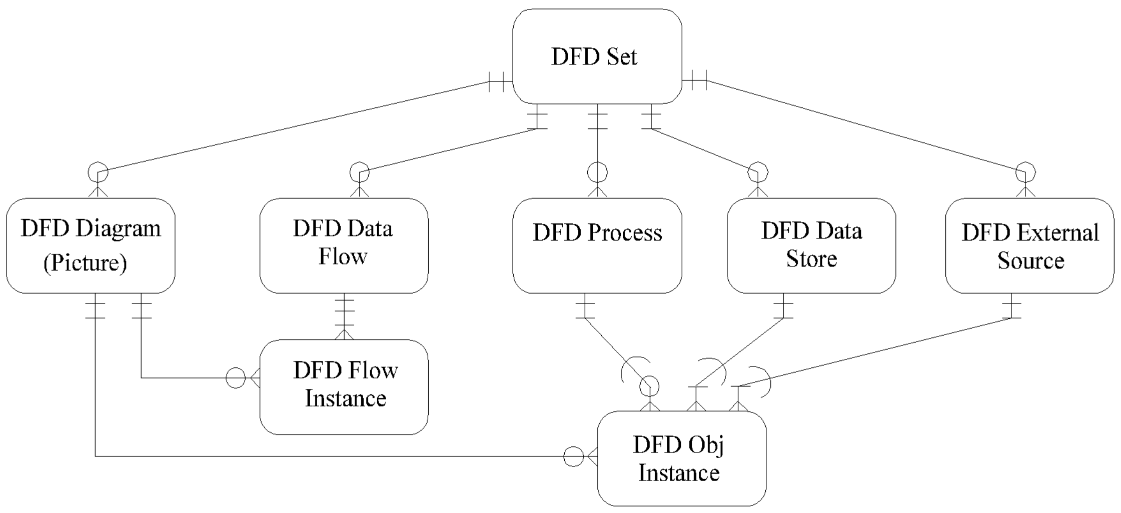

Figure 1: Systems Engineer Model Database.

cation of a single entity a new version of the complete hierarchy of components. At the same time, medium granularity does not introduce so many components as maximal granularity would. Their management is therefore easier.

When the great variety of entities used in CASE tools is taken into account, even the maximal granularity – which looks useless at the first glance – can be an appropriate solution for some kinds of entities. For example, entities that denote modules in the system being designed, or data models which do not allow hierarchies of data entities can be versioned at the maximum level of granularity.

It is reasonable to form a new version in a CASE tool database only after all transformations will have been completed that are consequences of the particular modification and moreover that can be performed by the tool. For example, a modification of a data flow diagram at some level in the hierarchy may bear as a consequence the necessity to reestablish a balance at all the remaining levels. The change should be propagated within the whole hierarchy. Obviously, these transformations can be automated so that the CASE tool should at least support them. Only after the transformations are completed a new version should be identified.

## 3 Versioning of data flow diagrams

We take data flow diagrams as an suitable example for discussing our approach to versioning of software components. Whenever we refer to a specific CASE tool, we have in mind the Systems Engineer that we have a fairly long experience with [3]. However, the ideas are applicable also when other tools are used.

Systems Engineer supports the minimal granularity in version management. When modelling a process, the whole process model as depicted in Fig. 1 is a configuration item.

When applying the technique of data flow diagrams, it is often the case that several versions of data flow diagram hierarchies have been formed. Among the reasons to form alternatives, let us mention e.g., desire to elaborate an alternative decomposition of the functional model, or need to consider different interface (context) of the system being analyzed. Frequently, the alternatives differ in just one diagram, or even just a few data flows or data stores. In all these cases, a new version of the whole process model is formed. In addition, if we take into account the development progressing in time and giving birth to ever new revisions, we must conclude that the minimal granularity is not satisfactory.

To sum up, we propose to elevate the level of granularity based on the following considerations:

- The functional model of the system is usually very complex (of course, it depends on the complexity of the problem being solved). We could equate it to a set of modules forming a subsystem. Considering such a complex object to be an atomic entity for the purpose of SCM results inevitably in a big number of versions. To manage them is difficult.

  On the other hand to consider each entity to be a configuration item (i.e., maximal granularity) is also difficult due to its complexity and could be compared to considering lines of code as configuration items.

- Many parts of versions formed in the above described way are identical. Unless an efficient storage technique (such as the delta technique) is employed, the minimal granularity option would lead to an unfavourable waste of space.

- Even when the delta or other technique for efficient storage of versions of the functional model is employed, there will occur problems whenever it is necessary to modify identical parts of the versions (alternatives) of the functional model.

We stress the last point because of its practical importance. Let us assume a situation when, during the analysis phase, a customer is to be presented a functional model of the system. Sometimes, it is useful to modify the functional modelling technique in order to simplify the view at the system. One such modification often used is to group all external entities that are linked to a particular data store or a process by the same

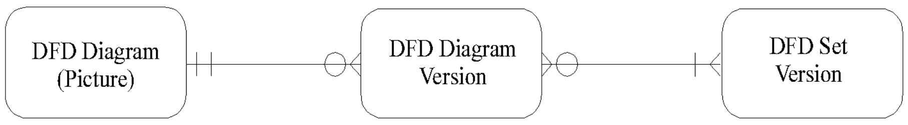

Figure 2: Extended Model Database.

data flows and to link the group just by one instance of the data flow instead [2]. Obviously, the resulting diagram is not consistent according to the standard rules for data flow diagrams. It could become consistent, however, if the modification is properly interpreted. The consequence for versioning with minimal granularity selection, however, is that a new version of the whole functional model is formed so from now these two versions are to be maintained consistently whenever a change in one of them is made.

We propose not only to elevate the level of granularity to the next one i.e., to medium granularity, but also to use the following technique to record and maintain versions of data flow diagrams. Instead of the original solution according to which the data entity `Version` is connected to the `DFD Set` as a high level object in data repository, our proposal is that an association to `DFD Diagram (Picture)` entity is chosen (Fig. 2) i.e., data flow diagram represents a configuration item. Data flow diagrams are combined to form a set. In an extended model of Systems Engineer repository, a new entity `DFD Set Version` associated to `DFD Diagram Version` serves this purpose. Finally, `DFD Set Version` is an item which goes to baseline.

Now we formulate our proposed method of version identification for CASE databases of software components. It consists of two steps:

1. Select a granularity version control for a particular formalism (technique) supported by a CASE tool i.e., for data flow diagrams, or state transition diagrams etc.:
   (a) *medium granularity* if the technique allows forming hierarchies of entities (e.g., hierarchies of diagrams),
   (b) *maximal granularity* if high level entities represent a single form,
   (c) *minimal granularity* otherwise.
2. Design operations with configuration items identified as above:
   (a) "standard" version operations such as create, get, check-in, check-out, branch, history,... as they appear in SCM tools [7] (could be adopted by way of integration the CASE tool with a SCM tool [9]),
   (b) operations which capture semantics of the versioned entities such as compare, merge [4].
   The compare and merge function should consider all the attributes of the entities. They should work with three levels of comparing the entities:
      - *equal* (name and all other attributes are equal),
      - *similar* (all the attributes are equal provided they have a defined value; names are different),
      - *different* (name and all other attributes are different).

   Moreover some operations from the "standard" group can be enhanced by the technique of rules checking e.g., level balancing control during check-out operation together with automated propagating of change through the hierarchy of data flow diagrams.

## 4 Experimentation

To verify experimentally the idea of the solution that we have proposed for the problem of identifying configuration items in functional modelling using data flow diagrams, we have created a prototype. We were constrained by the fact that direct modification of the repository structure of the CASE tool Systems Engineer is not possible (but the similar holds for most of the tools that are available commercially). Therefore, we implemented the cooperation between the tool and our prototype by export and import operations on the database in the design interchange format (DIF).

In implementing the merging of versions we considered only entities which are present in the respective diagrams (cf. Fig. 1) regardless their positioning. Technically, a difference in positioning of an entity in a diagram can lead to a new version. Comparing them, however, reports equality.

We decided to stay with the standard way of identifying entities (in particular, processes) within the functional model. By not introducing any enhancements, we want to endorse simplicity of the proposed technique. As a consequence, there may exist in different versions several processes with equal identification numbers. In our prototype, we implemented the distinction between the objects by establishing associations with the corresponding version of the diagram.

## 5 Conclusions

The difficulties with version control of software analysis and design related entities in databases of CASE tools derive mainly from their specific properties. When a proper level of granularity is chosen, the rules for configuration forming need almost no adjustment to cooperate smoothly with the exisitng SCM tools. Integration of CASE and SCM tools as described in e.g., [9, 6, 5] can in such a way be enhanced by the understanding of the semantics of software components.

Our proposed solution which is based on the medium granularity is suitable chiefly for those techniques which allow forming hierarchies of entities. A typical example of such a technique is the functional modelling technique by data flow diagrams which was used for experimenting with the proposed solution. Other example would be a hierarchy of requirements, where whole group of requirements at a particular level could be chosen as the configuration item. In this case, the maximal granularity as adopted by the Systems Engineer tool does not appear to be suitable for even moderately larger problems because the number of requirements tends to be rather big.

Another possibility of applying the proposed approach could be in object-oriented modelling techniques.

## References

[1] E.H. Bersoff. *Elements of software configuration management.* IEEE Transactions on Software Engineering, SE-10(1):79–87, 1984.

[2] M. Bieliková, M. Galbavý, I. Kapustík, Ľ. Molnár, and P. Návrat. *Using a CASE tool in developing an information system for Slovak Telecom.* In Information Tools and Technologies, volume 1, pages 159–164, Moscow, 1996. International Academy of Automation.

[3] M. Bieliková and P. Návrat. *An experience with the use of Systems Engineer CASE tool.* Software and Knowledge Engineering, to appear.

[4] J. Buffenbarger. *Syntactic software merging.* In J. Estublier, editor, Software Configuration Management, ICSE SCM-4 and SCM-5 Workshops, pages 153–172. Springer, 1995.

[5] J. Estublier and R. Casallas. *The Adele configuration manager.* In W.F. Tichy, editor, Configuration Management, pages 35. John Wiley & Son Ltd, 1994.

[6] P.A. Favre. *Succesfull configuration management for CASE.* In EOQ-SC SAQ, editor, Software Quality - Concern for People, pages 394–399. Hochschulverlag AG an der ETH Zürich, 1994.

[7] A. Lobba. *PC-based version control.* In WWW: http://www.silcom.com/~alobba/pc_vc.html, May, 1996.

[8] Systems Engineer. *DIF mapping guide.* Learmonth & Burchett Management Systems Plc., 1995.

[9] K.C. Wallnau. *Issues and techniques of CASE integration with configuration management.* Technical Report CMU/SEI-92-TR-5, March, 1992.