# IMPROVING PROGRAM COMPREHENSION OF OBJECT-ORIENTED SOFTWARE SYSTEMS WITH OBJECT-ORIENTED DOCUMENTATION

Johannes Sametinger

Institut für Wirtschaftsinformatik
CD Laboratory for Software Engineering
Johannes Kepler University of Linz
A-4040 Linz, Austria

Object-oriented programming has brought many advantages to the software engineering community. Especially, the reuse of existing software components and application frameworks has improved the productivity in software development considerably. Now, the object-oriented programming paradigm has advanced in years and increasingly object-oriented software systems have to be maintained. Program comprehension plays a major role in software maintenance. Additionally, the increased reuse of software components, which is propagated and supported by object-oriented programming, necessitates the understanding of existing software during development and, thus, program comprehension becomes even more important.

Very often the only information a maintenance programmer can trust is the source code. It is the only accurate, complete and up-to-date representation of a program. However, source code listings are hardly suited to representing design decisions, the global system structure, or the interactions among different system components. System documentation is necessary to enable reuse and maintenance of software components. It should remain valid as long as the software is being used. Nevertheless, system documentation is often inadequate and out of date, and therefore unreliable and misleading.

Good (system) documentation should be complete, current, and consistent in style. We apply object-oriented technology to documentation in order to improve its quality by better reflecting the logical structure of a system. This way of organizing software documentation eases program comprehension of object-oriented systems.

## Class Libraries and Application Frameworks

Typically, object-oriented software systems are extensions to class libraries or application frameworks. This characterization should become true for the documentation as well. Hence, such documentation should not describe an entire system from scratch; instead, it should contain a description of all extensions and modifications of the reused components and describe all system-specific parts as well. It is assumed that separate library documentation is available which—similar to the code—should build the base for the entire documentation.

With the object-oriented concepts of inheritance, information hiding, polymorphism, and dynamic binding software components have become reusable and extensible without the need to make any changes in the source code of these components. The reuse of whole collections of classes, called class libraries is a major step in increasing the productivity of software engineers. However, class libraries and application frameworks have strong impacts on the comprehension process. Comprehension of a software system being based on a class library depends on the documentation of the class library itself and the documentation of the application specific source code. In order to guarantee complete and consistent documentation of the whole software system, the documentation—similar to the code—has to be easily extended and modified without making changes to the original documentation.

## Inheritance of Documentation

Inheritance can be viewed as both extension and specialization (see [Mey88]). A class X inherits from one or more superclasses A,B,C. The features of the superclasses are a subset of the features of class X, i.e., X heirs and thus provides whatever A, B, and C provide plus its own (extension). On the other hand, inheritance is used to realize an is-a relation. For example, a rectangle (X) is a special visual object (A) with the features of a visual object but specialized behavior (specialization). Inheritance is a means of better organizing the source code of a software system, because the logical structure of the software is getting closer to the structure of the part of the real world to be modeled.

In order to apply the inheritance mechanism to documentation, we divide the description of classes into subsections that can be modified and extended in subclasses. Thus, the documentation of a class is a combination of class specific descriptions plus the inherited subsections of the superclasses.

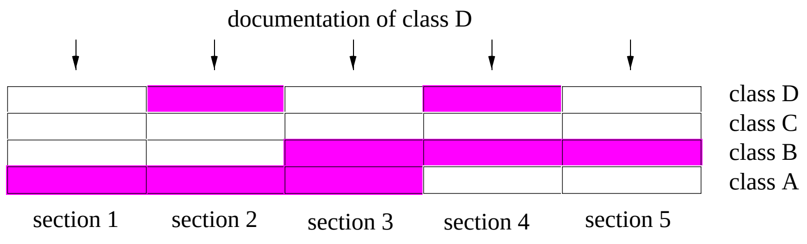

Fig. 1: Inheritance in the documentation of a class

Figure 1 contains the structure of the documentation of classes A, B, C, and D. The documentation of class A consists of 3 sections, and classes B, C and D have five documentation sections. Class D inherits section 1 from class A, sections 3 and 5 from class B, and has sections 2 and 4 of its own. Please note that the documentation of class C consists of five parts, though not an extra line of documentation has been written for this class.

The documentation of methods is organized the same way as that of classes. It is worth mentioning that there might be classes that do not implement a certain method. Naturally, they do not contain any documentation for this method. However, both the method and its documentation are available in these classes through inheritance.

## Conclusion

In our research projects we use the public domain application framework ET++ (see [Wei89]). Detailed documentation for the most important classes and methods of this class library is available. Unfortunately, it is rather cumbersome to get relevant information because the data is usually spread over the descriptions of several classes (superclasses). Therefore, we divided the documentation into sections (e.g., short description, instance variables, methods, example) to be used with our programming environment DOgMA, that supports object-oriented documentation (see [Sam92]).

Although the documentation of ET++ had not been written with inheritance in mind, the benefits of applying this mechanism has been enormous. The possibility to get the part of the documentation that is relevant for using a special class or method, even when it is spread over many superclasses, made reusing a complex class library much easier, and was especially esteemed by our students.

## References

[Mey88] Meyer Bertrand: Object-oriented Software Construction, Prentice Hall, 1988.

[Sam92] Sametinger J.: Object-oriented Documentation, submitted for publication, 1992.

[Wei89] Weinand A., Gamma E., Marty R.: Design and Implementation of ET++, a Seamless Object-Oriented Application Framework. Structured Programming Vol. 10, No. 2, 1989.