CASE FOR COMMUNICATIONS SOFTWARE

J. T. Krist
Ameritech Services Science and Technology
2850 Golf Road, Rolling Meadows, IL 60008

EXTENDED ABSTRACT

Computer Aided Software Engineering (CASE) has been heralded as answering the need for software development productivity. The allure of using CASE for communications software is understandable: there is tremendous demand for more software; sooner, cheaper, with higher quality. Software has become a critical element in the development and operation of communications systems. These systems require hundreds of people and years to build. Evolving their capabilities is expensive, tedious, and error prone. The programs have many, many assumptions deeply imbedded in them. When there are changes in user needs, regulation, technology, or other factors, there is no easy way to unwind the various design optimizations and restructure the program to fit its new environment. It is not a surprise then that software is increasingly seen as an impediment to offering new communications products and services.

Automating the development of software may appear to be the remedy to this situation. However, expectations regarding the potential for automatic programming are idealized and probably unachievable.[1] Any automated tool relies on a formal model to represent information in some systematic way. Most CASE tools provide only a very weak formal system so that the mechanical analysis and transformations they perform are limited. When customers and managers ask for complete and consistent specifications, they are expecting a good deal more than the proof that all data flow diagrams are correctly structured with a non-empty entry in the dictionary for each term. Likewise, automatic code generation from end-user specifications is possible today only for very narrow domains. And automatic reverse-engineering of code to deduce higher level architecture is only little help when huge programs have been modified over the years by many people who themselves had no explicit, clear, consistent view of such a structure.

Why are these limitations not widely understood? First, software development remains a mystery to non-practitioners. Even among specialists, there is difficulty in developing a common understanding of the problem because of great fragmentation of expertise. There is also the wide gap between the research and development communities. Few people fully understand the technology, the issues of scale, and the non-technical aspects of the software profession.

Scale, that is the size of the systems and the number of people involved, is perhaps the most salient feature that limits the use of CASE tools today in communications system software. Why are so many people needed to build and maintain these systems? Lacking an obvious answer to that question, it seems prudent to address this issue, rather than to postulate that it will disappear with the next generation of tools.

CASE models of the requirements and design for large-scale systems will themselves be large, elaborate software objects. These models will require support capabilities similar to those proposed for software development environments (SDEs) for large systems. Perry and Kaiser [2] have proposed a classification of SDEs from the viewpoint of scale using a sociological metaphor; the individual, family, city, and state model. Few CASE tools offer any support for coordination, cooperation, or commonality. This mismatch of need relative to
The facilities available add more complexity to the development process. Inadequate CASE tools, by imposing overhead and coupling, add bottlenecks and churn to the process. The result is decreased productivity, repeated false starts, and "shelfware." [3]

To make CASE tools and formal methods more suitable for the development of communications software, improvements need to be made in the following areas:

1) **Incremental, Evolvable Environments:** Models and specifications for communications software will be large. Tools need to naturally support incremental change and analysis of the models. Many people should be able to work in an incremental fashion without long delays or too tight coupling with others. Support for multiple versions of specifications and tools to analyze differences between versions should also be provided.

2) **Compatibility with Existing Methods:** No current CASE tool will cover the entire life cycle of a large system. At some point, the CASE environment will give way to conventional techniques. There must be a solid reason for people to use (not circumvent) the higher level tools. CASE tools should replace effort, not just add activity. For example, report and diagram generation from a CASE model should replace documentation required by current development policies. To do so, the tools must provide flexible, customizable report generation facilities.

3) **Ease of Learning:** Many people, with differing backgrounds and expertise, need to read and understand the model created using a CASE tool. Natural language paraphrases along with schematic graphical displays can permit people to understand the ideas that are being presented with a minimum of special training. The designers who create the models however, will need extensive training and experience to successfully use the power of higher level formal systems.

4) **Application Specific Building Blocks:** Almost all new communication services build incrementally on existing products or must deal with an existing complex environment. CASE environments need to supply application specific building blocks so that users can quickly begin to work on their problem and not spend weeks building scaffolding.

The capabilities described above are derived from an analysis of the technology transfer process rather than from focusing on automation.[4] [5] This is important because overall, software development is a people intensive activity. Attention to technology transfer can increase the success of new technology trials. This is particularly important because, given the current state of the art, the benefits from CASE tools are modest and the barriers to their application on large communications software systems are great.

**References**


