EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN - PS DIVISION

LL

CERN/PS 93-48 (CO)

CERN LIBRARIES, GENEVA

CM-P00059542

849348

9

**EXPERIENCE WITH CASE TOOLS IN THE DESIGN OF PROCESS-ORIENTED
SOFTWARE**

Ognian Novakov*, C.-H. Sicard**

* Now at Logatronix, Sofia (Bulgaria)

** PS Division

* Now at Logatronix, Sofia (Bulgaria)

** PS Division

International Conference on Accelerator and Large Experimental Physics Control
Systems

Berlin, Germany, October 18-22, 1993

Geneva, Switzerland

15/11/93

EXPERIENCE WITH CASE TOOLS IN THE DESIGN OF PROCESS-ORIENTED SOFTWARE

Ognian Novakov*, Claude-Henri Sicard**

* Now at Logatronix, Sofia (Bulgaria), ** PS Division, CERN, Geneva, (Switzerland)

In Accelerator systems such as the CERN PS complex, process equipment has a life time which may exceed the typical life cycle of its related software. Taking in account the variety of such equipment, it is important to keep the analysis and design of the software in a system- independent form.

This paper discusses the experience gathered in using commercial CASE tools for analysis, design and reverse engineering of different process- oriented software modules, with a principal emphasis on maintaining the initial analysis in a standardized form.

Such tools are in existence for several years, but this paper shows that they are not entirely adapted to our needs. In particular, the paper stresses the problems of integrating such a tool in an existing data-base-dependent development chain, the lack of real-time simulation tools and of object-oriented concepts in existing commercial packages.

Finally, the paper attempts to show a broader view of software engineering needs in our particular context.

1. The software development context at CERN PS/CO

The present situation shows an increasing part of the development manpower made of young professionals, with varying previous background, staying for a duration of 1 to 2 years. The remainder of the staff is mainly devoted to managing and maintaining existing software.

Practically all software is written in C code. This amounts, in the present state of the project, to about 60,000 lines of code for the process-oriented part of the project.

As much as possible of the static information is entered in a relational database, with associated file/code generation [1], and a set of software development procedures.

Projects are divided among several groups, outside the Controls group, and the actual software development is often done in a late stage of the project.

The absence of profit making incentives and the research orientation of CERN also have an important effect on software development process, in the sense that:

- Economic figures for initial estimates and actual project costs are approximate (when they exist). In this context, the reasons to use tools which may provide productivity improvements must be found outside pure economics, and the cost of such tools cannot be justified by hard figures.
- It is difficult to impose to all developers the use

of any given method, because there is a well-known reticence to accept changes in work habits, and CERN culture will rather encourage individuals to find by themselves the methods best suited to them than employ hierarchical enforcement.

The consequences of this situation are:

- Training investments on temporary staff are necessarily limited (compare with an estimated 150 hours of training need for software engineers in companies [2]).
- Difficulty of realizing a good system design, in particular concerning the hardware/software tradeoffs.
- CERN groups which develop similar styles of software spend little effort to use a uniform set of development procedures, to enforce design and code reviews, or to provide a central quality assurance team which has the authority to enforce the usage of rules or standards. On the other hand, many of the latest tools may be found, for evaluation or use, in different places.
- In the absence of profit-related measures, we must obtain meaningful numerical figures for other measures of the overall product quality (reliability, maintainability,...).

2. Use of CASE tools for Analysis

We have licenses for Software through Pictures (StP) from IDE, and also evaluated RTEE (Westmount Technology, Netherland) and DecDesign (Digital Equipment).

2.1 Functionality:

All tools support essentially the same basic methods for structured analysis and structured design, namely:

- Data/control flow modeling (Yourdon/De Marco)
- Entity relationship modeling (Chen)
- Data structure modeling (Jackson)
- State Transition modeling (Yourdon)

We used with StP the Hatley/Pirbhai [3] real-time extensions.

Most tools ignore the problem of distributed systems and architectural constraints (an exception is RTEE, which provides System Architecture Diagrams as an intermediate phase between analysis and design, allowing to map defined data and processes to actual computational tasks and processors), although this type of model is mentioned in several books (Hatley/Pirbhai).

Consistency and completeness checks are provided by these tools; however, although passing from analysis to design is mostly an engineering activity which cannot be reduced to an automatized process, the lack of traceability between analysis and design is a defect of all such methods.

All such tools use the term of 'real-time', but none of them take in account timing constraints coming from internal architecture or hardware physical properties.

2.2 Integration

As mentioned previously, our system uses a relational database as a repository for part of the design information: function list, call sequences, data structures. This information is part of the analysis or design, from which it needs to be exported in the database. Interface standards for CASE tool information interchange are still in a draft state (current CDIF interim standards as from July 91), and few tool builders subscribe to them but rather provide their own set of access routines. Thus any code to integrate CASE tools is supplier-dependant and probably short-lived.

2.3 Document preparation

StP offers a sophisticated document preparation tool, which allows to produce proper system specification documents.

2.4 Customisation and openness

StP offers wide customisation of the environment and editors via template files, provides file formats and offers library access to its data dictionary (but, as mentioned above, via non-portable specialised routines).

2.5 User-interface, ergonomics

All tools are quite similar, and navigation from one to another is relatively simple, with the restriction of StP whose current version(4.2D) is based on X but doesn't provide a Motif interface. (This should change in the next version, but has been a factor of rejection by users).

The main benefits we gather from these tools are an increase in documentation quality and a stronger emphasis on the requirements analysis phase. We do not think these tools can bring a dramatic increase in productivity, as advanced by some of their advocates (or vendors...).

3. Reverse engineering facilities

The interest in reverse engineering tools comes from the difficulty of guaranteeing a permanent adequacy of actual code with the design documents which were elaborated at the time of its conception. Several tools can be found on the market, which claim to automatically keep design, code and documentation consistent during the life cycle of a software system.

A trial of 2 such tools (C Development Environment from IDE, and Ensemble, from Cadre) has been realized on an existing part of the project, in order to compare the facilities provided.

- The design documents consist of structure charts, call graphs, data-structure diagrams and some additional analysis information, in particular metrics (cyclomatic complexity, data complexity,...), cross-references and on-line queries (functions which call a specific function, or using a specific global...) The structure charts show the call sequences of each procedure, the globals used inside each procedure (although it does not provide information of possible side-effects acting on these globals, which would be very useful)
- From the design documents, it is possible to automatically generate code frames (function interfaces and data definitions), possibly augmented with comments or code coming from the annotations entered in the design documents.

As main conclusion on the usage of such tools, we believe that they must be seen as part of an interactive development environment, integrated with code production tools using a common repository. It is only in such a context that their dynamic features (navigation, browsing), which are their most interesting feature, can be used at best.

4. Reliability measurements

Reliability is the most important feature in the process-oriented part of a control system. It is therefore very important to obtain figures (in some quantified numerical form) for this reliability if we want to measure progress. In our context, we do not think that extremely exhaustive tests as done on critical embedded systems or widely commercialised programs are cost-effective. These types of tests can gather statistical information on the number of defects found per thousand hours of tests, and thus give estimates of the remaining percentage of defects, before declaring a system operational.

A less ambitious approach makes use of test-coverage tools, which give some confidence on the percentage of code covered during testing, and of test suites for each equipment-interface software which can be used in particular after any change in the software.

A major difficulty in testing accelerator controls is the need to cope with a wide variety of external conditions, hardware faults or malfunctions, some leading to subtle changes (such as for timing problems). Our experience shows that most of the additional delay for repair of hardware faults are due to errors or incomplete diagnostics in the accompanying software.

Not much training is provided on this subject in the academic field, and unless emphasised enough, many programmers assume their code is correct once its main path works according to specifications. As a consequence, the treatment of abnormal situations must be specified as thoroughly as possible in the analysis phase.

We have gathered a long history of errors on the operational system, which are in the process of being classified. This should provide evolutive estimates of Mean Time Between Failures and Mean Time To Repair which are of most interest to our clients.

5. Code standards and style

The fast growing usage of the C language should not hide the fact that C code is intrinsically more prone to errors than other high-level languages such as Ada, and C programmers may be tempted by the ease of quickly producing software.

Adding to this the sometimes low level of knowledge of the young professionals, it is extremely important to provide them with a set of style rules and standards before they start implementation.

There exist now a variety of published standards from which to choose, with varied degree of details.

We consider that the main features must concern portability (word sizes,...), source readability, naming conventions, symbolic numerical constants and external declarations.

All code, before being integrated in the operational environment, goes through an acceptance procedure which tries to enforce the above standards as much as possible (including use of the 'indent' program, check via 'lint', check of all globals for code integrated in libraries, proper documentation...) However, much care must be devoted to explain the reasons behind such standards, and the advantages the programmer may reap from their usage, in order to reduce the changes imposed by this acceptance test to a minimum.

6. Other investigations

Object-oriented design and languages: At present, no well-supported object-oriented languages are available on the front-end platform (LynxOS), and only prototyping work has been engaged on languages (C++) and associated tools (ObjectCenter for development, Objecteering/OMTool for design).

We expect an improvement in program quality, due to code reuse, better type checking and encapsulation. Their use should also improve maintainability, from the closer match between real-world objects and program objects. However, understanding object-oriented programs is not necessary trivial, mainly when multiple inheritance and polymorphism features are used. Adequate tools must be provided from the start to help navigating in existing programs.

7. Future directions

Most of the above-mentioned activities need a strong consolidation, which depends on available manpower, evolution of standards, and a software process development plan which needs to be endorsed by the management of several groups involved in similar tasks.

References

- [1] J.Cuperus, F.di Maio and C.H.Sicard, The operator interface to the Equipment of the CERN PS accelerators, these Proceedings.
- [2] R.Dion, Process improvement and the Corporate balance sheet, IEEE Software, July 1993 pp.28-35.
- [3] D.J.Hatley and I.A.Pirbhai, Strategies for real-time system specification Dorset House, 1987.