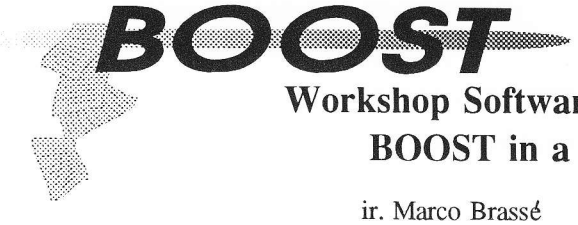


Workshop Software Engineering 1994: BOOST in a bird's eye view

ir. Marco Brassé ir. Ton van den Berg

One of the highlights of the OOTI programme is the workshop Software Engineering. During spring 1994, two teams have developed software for an embedded copier system from Océ . An overview of the experiences of one of the teams, called BOOST, are described here.

BOOST had been assigned to work on the specification, design, and implementation of a complex control system for an Océ high-volume copier. Two restrictions were imposed on the BOOST project, namely the design should be done in an object-oriented manner, and the implementation should be done in the object-oriented language EIFFEL. To emphasize the realistic setting of the project, Ruud Vermeulen, a project manager from BSO/Origin, Ruud Vermeulen, played the rôle of customer; he was supported by several critical experts, guided by Peter van der Stok.

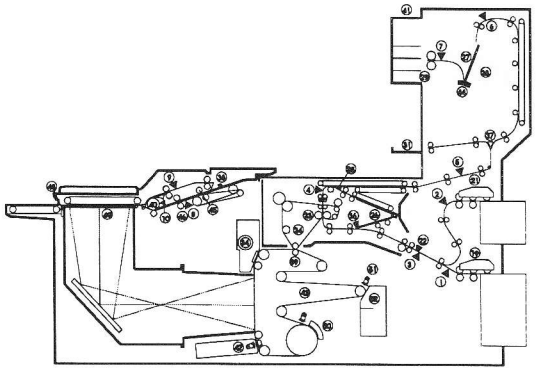


Figure 1: Model of the Océ high-volume copier

So, BOOST was facing a number of challenges. Especially developing software for embedded systems was a rather unfamiliar field. Several new aspects related to embedded systems had to be mastered; the hardware of the copier must be modeled, and the software should satisfy real-time constraints. Moreover, although we had followed the course *Object-Oriented Design* (OOD), we did not have much experience with object-oriented techniques, at that time.

Apart from these technical problems, we had to become a well-organized team, with proper agreements, clear responsibilities for individual persons, and an efficient way of working. To ensure this, several team building activities had to be organized. Furthermore, for customer-driven production holds that a satisfying product is good enough. Hence, the experts needed to be convinced that our products (and documentation) sufficed. This was also considered a team effort.

Organizational aspects

The team first chose a project manager from several candidates (also team members). Subsequently, under his supervision a project plan was agreed on, describing the different phases of the project. Each phase required a certain amount of resources, for which an estimate was given in terms of number of people, available time, and necessary equipment. Based upon these estimates, it was decided when a phase started, ended and consequently, which phases had to be done in parallel. In addition to this information, an estimated budget for each phase was given. Figure 2 shows the time schedule used in our BOOST project.

For each phase, several sub-teams were working in parallel at the activities of the phase plan. As it was our first major project, one team was preparing a Software Quality Plan (SQP) and a Software Verification & Validation Plan (SVVP). Since it is not useful to do this *preparation* phase with fifteen people, the *specification*-phase took place in parallel.

Because we did not have an actual copier to test our copier control system, the development of a software simulator was included in the project description as well.

In the design phase, the BOOST team was split into two teams: one team specified and designed the copier control system, and the second team

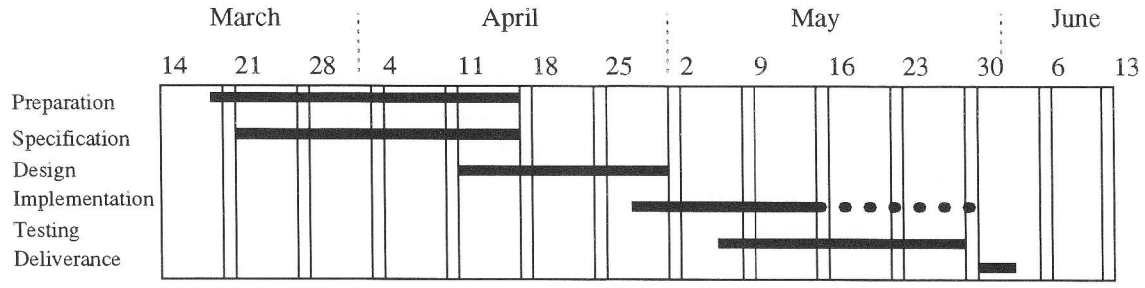


Figure 2: Planning BOOST project

specified and designed the copier simulator system. The team which designed a certain part of the system, was also responsible for the implementation and debugging of that part (debugging actually took place in the testing phase). In general, a team member that specified and designed a certain part of the copier, also took care of the implementation.

The planning and deadlines were very useful as an indication of the progress we made. Several deadlines were not met, and werepostponed by one or two days. Eventually, the project was successfully completed at June 10th, 1994.

Object-oriented approach

As a principal software engineering tool, we used Object Modeling Technique (OMT, cf. [1]). This technique is well-documented and comprises most stages of the software life cycle. OMT is based on the well-known entity relationship paradigm, extended with several concepts to support object-oriented designs, such as inheritance. Several (other) arguments in favor of using OMT are listed below.

- OMT has an object-oriented style, starting from the specification phase.
- OMT's design principles and procedures are clear.
- OMT does not require much training in applying the method.
- OMT is supported by a software tool, called OMTool.

Despite all these promising features, a few remarks are in order. OMT is a general framework for object-oriented designs, so we only applied those steps that we considered relevant for our project. A drawback in our view is the bottom-up approach in the specification phase to identify all relevant real-world objects (or entities) of the copier. Further, OMT does not provide special facilities to analyze real-time constraints. This is still a topic of research. Finally, we would like to mention that we had merely access to a PC version of OMTool. Noticing that several members of the design team should have access to the tool simultaneously, we decided not to use OMTool.

An alternative to OMT that was presented in the course OOD was a graphical representation technique of object-oriented classes (cf. [3]). However, this technique only covers a part of the design trajectory, and at the time of our workshop, it was not supported by software.

Principal design decisions

In order to model the complete copier system, we first split the copier system into five subsystems, which are briefly mentioned below (Figure 1).

- Original handler: transports originals from the original tray to the glass plate.
- Process part: flashes the original to generate an electrical image on the photo conductor belt. The electrical field is used to hold the ink (toner) for the copy.
- Paper handler: transports a sheet of paper from the paper tray to the place where the ink on the photo conductor belt is transposed on the sheet.
- Finisher: transports a copy to the output tray; it also collects and staples sets of copies.
- User interface: handles user interaction and displays warnings.

Apart from these subsystems, in which the hardware is modeled, we also identify the following subsystems: originals, sheets (or copies), and a

subsystem to control the process part of the copier.

A first design decision is to choose which subsystems are considered active, and which subsystems are passive. An active subsystem eventually corresponds to a (software) process; all active subsystems together define the dynamic behavior of the system. A passive subsystem can be seen as a model of the hardware, which provides services for the active processes. In principle, two approaches exist for choosing which subsystems take care of the dynamic behavior of the system.

1. *Structure oriented.* In this approach, the active subsystems are the original handler, paper handler, etc. The passive subsystems are the originals, sheets, etc.

2. *Process/data oriented.* In this approach, the active subsystems are the originals and sheets (the data objects), whereas the passive subsystems are the original handler, paper handler, etc.

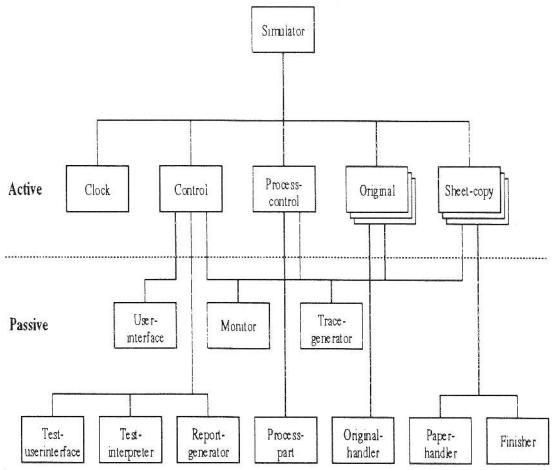


Figure 3: Simulator Architecture

With respect to extendability and reusability properties of object-oriented designs, we note that a structure-oriented approach is preferred if the hardware is likely to change a lot over time, whereas the data-oriented approach is preferred if the way of making products (viz. the copying process) is likely to change a lot over time (e.g., adding new paper sizes to the requirements of the copier). It will come as no surprise that we have chosen for the second approach; it is also a very natural way to model the copier, as the data (originals/sheets) actually 'flows' through the copier, and thereby determines the behavior of the copier. Figure 3 shows the architecture of the simulator, showing the active and passive systems.

Passive subsystems can be modeled quite well in OMT. However, active subsystems have to satisfy real-time constraints. A dynamic behavior should be described for each active subsystem (as well as its interaction with other subsystems). For example, a *sheet* subsystem reads sensors and activates actuators at specific times. This can be written down in OMT using state charts, but it requires a detailed *schedule* of all actions in time. Figure 4 shows a small example of such an OMT state-chart for a photo-conductor-belt.

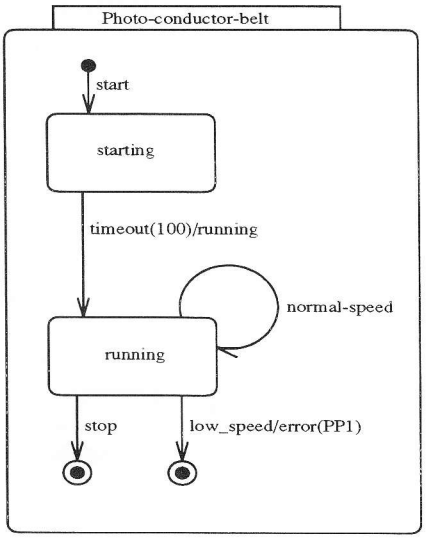


Figure 4: Example of an OMT state-chart

Constructing such a schedule is done in a rather pragmatic way. Based on the performance requirements of the copier, a schedule is constructed by determining at which moments in time a process should perform a certain action in order to meet the performance requirements of the copier. However, this does not guarantee that two actions from different processes do not occur at the same time slot (with a typical duration of 2 ms). Thus, if more than one action is scheduled in a certain time slot, it could happen for these actions to consume more CPU time than the length of the time slot, especially if the software overhead is large.

Nevertheless, we convinced ourselves and the experts that the design could be implemented using only one processor, while all actions could be scheduled without introducing conflicts (with a maximum CPU utilization rate of 8.73%, including user interface polling). Although we did not use a

spreadsheet tool for detecting and solving scheduling conflicts, the competing TORNADO team has shown its benefits and effectiveness.

After having found a feasible schedule for an active subsystem (or process), its detailed dynamic behavior can be described well in OMT. Apart from the analysis and design of the subsystems themselves, the interaction between the subsystems is often very intricate. Initially, we used several synchronization messages between the subsystems (using the message passing paradigm between processes), but eventually we have replaced most of these messages by a single synchronization point for all subsystems in time, namely the time-slot in which the control system detects a special physical hole in the photo conductor belt.

Other design aspects

The previous section was devoted to several principal approaches to cope with problems regarding embedded systems. Apart from these problems, we also came across other serious design issues. A simulator must be built to test the control system. Furthermore, parallelism had to be implemented, or simulated, in the sequential language EIFFEL. Finally, the problem of error recovery had to be solved. If, for example, a sheet gets jammed inside the copier, the user should be able to remove the sheet, after which the copying proceeds. These issues are addressed in the remainder of this section.

The simulator system consists of a monitor that graphically displays the position of all originals and sheets, and the state of all relevant hardware components (see Figure 1). In order to determine the position of a sheet inside the copier, the actions of the sheet process are monitored by a similar process controlled by the simulator (a so-called *simulated sheet process*). When, for example, a sheet process reads a sensor or activates an actuator, it is noticed by the corresponding simulated sheet-process, and thus knows the position of the real sheet inside the copier. The simulator processes also take care of the setting and resetting of sensors to be read by the control software.

Using this concept, the copier control system did not have to be changed in order to test it within the simulator software. The only communication between the processes of the two systems occurs at the level of low-level hardware components, whose actual code is shared in the test environment.

A consequence of this technique is the creation of a lot of UNIX processes in the test environment. In fact, an implementation on our UNIX system would give problems with respect to the memory and speed of the UNIX machines. To avoid these UNIX problems, we decided to build a scheduler in EIFFEL to simulate parallel execution of the processes. The scheduler is based on the ideas used in the programming language SIMULA-67, in which a similar scheduler is used to simulate parallelism.

In order to use such a scheduler, the processes (i.e., active subsystems) had to be encoded in EIFFEL as *state machines*. This can be seen as follows. A process can in this way send a request to the scheduler to enter a certain state of its state diagram at a certain time slot in the future (to perform a certain action). The scheduler acts on this request by placing the process in a waiting queue, and avoiding busy waiting. When the time slot is reached, the process is awaken and will find itself reactivated in its predefined state.

By encoding all processes (including the scheduler) in EIFFEL, we obtain one UNIX executable process, apart from a C-program which takes care of the communication between the simulator test software, the (test) user interfaces and the graphical monitor.

The scheduler proved to be very useful, for processes could now be easily and efficiently activated, deactivated, and reactivated. Furthermore, the scheduler supports rescheduling of processes in the waiting queue. This last fact appeared to be crucial for a clear implementation of the error recovery routines. A drawback of the method was that the concept of state machines did not make the EIFFEL code very elegant.

Conclusion

OMT cannot be applied directly to design software for embedded systems (it also does not support real-time concepts). As an alternative the method in [3] can be used, but this method is a technique rather than a design strategy.

The language EIFFEL is easy to learn and simple to use. Because EIFFEL has no functionality to deal with parallelism, state machines were implemented. This combination worked fine and we

circumvented a lot of rarities and implementation restrictions of UNIX processes (e.g., the restricted amount of processes as well as the problem of sharing data between several processes).

Although object-oriented methods are used, extendibility and reusability properties are not totally met. May be this is due to the lack of experience we had with object-oriented methods, but also the short lead time of the BOOST project must be taken into account. Being the first OOTI team who has developed a working copier system with error recovery for all copying modes, has had some effects on those quality aspects.

The members of the BOOST team can all look back at a successfully completed project. Not only from a technical point of view, a working copier system has been developed, but also the atmosphere in the team was very good. Everyone participated in a very active way, which eventually resulted in the realization of the copier system. □

References

[1] Rumbaugh et al., Object-Oriented Modeling and Design, Prentice Hall International Editions, 1991

[2] BOOST team, BOOST project documentation, OOTI, Eindhoven University of Technology, 1994

[3] O.S. van Roosmalen, A Hierarchical Diagrammatic Representation of Class Structure, Department of Computing Science, Eindhoven University of Technology

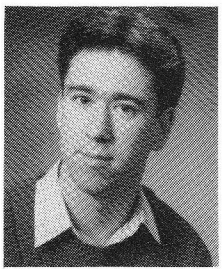
The Workshop Experience

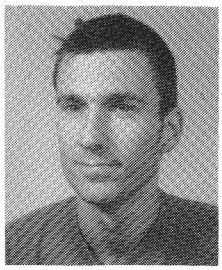
Continuation of page 16

the culmination of the 15 months cursory part. The utilization of a real-life exampleis important for the motivation of the students and helps them to place the course in the context of their later professional careers. The engagement of a professional consultant for the project management course makes the students more acutely aware that they are facing real-life problems and are not suffering artificially created circumstances. His experience is essential for the student's grasp on the progress of the project. The presentation of their results to the staff of Océ , who provide the case, shows them the possible place of their technical skills within the organization of a future employer.

The more important lessons which have been learned over a four year period are: (1) the presentation of the technical courses before the workshop takes place is essential, (2) the project management courses must be given before and during the workshop for a maximum effect, (3) the roles of client and technical advisors when filled by the same staff members are delicate, and (4) the assessment of the student's individual performance is difficult.

This form of teaching is extremely useful: the theoretical knowledge can be put into practice in a rather natural way, and students realize that project management methods, when properly applied, help to solve the social and planning problems. □





Ir. Marco Brassè (left) and ir. Ton van den Berg (right) are students of the post-masters programme Software Technology. Marco Brassè was team leader Design & Implementation of the control system in the BOOST project. Ton van den Berg was BOOST quality manager and member of the simulator system. Both authors are member of XOOTIC.

Dr. Peter van der Stok (left) and dr.ir. Marloes van Lierop (right) are staff members at the Department of Mathematics and Computing Science of Eindhoven University of Technology. Peter van der Stok is the organizer of the workshop and plays the role of expert in it. Marloes van Lierop is the Programme Manager of OOTI and supervisor of all OOTIs.