Chapter 6. THE FINE PLANNING SUPPORT

6.1. Introduction

The previous chapters provided insight in the current work practices at the Cabofa solid board mill. We have demonstrated that the planning problems are complex and the production planners demand an increased level of planning support. Introduction of a DSS may provide a way to increase planning support at Cabofa. In order to test this assumption, we developed a DSS with restricted functionality. During the development stage, particular attention was given to production planners’ requirements and the current planning task execution, ideally resulting in a DSS that is both user-friendly and supports the planning tasks.

In order to develop a DSS, two options are generally possible: either a sequential flow of actions is performed, or a more cyclical and iterative flow is performed. The sequential flow refers to methodologies such as System Development Methodology (SDM), which is promoted by researchers such as Eilers et al. (1991); the iterative flow refers to methodologies that apply to prototyping (Carroll et al., 1995 and Moran et al., 1996). The distinction is not as strict as suggested here, but we would like to further develop this point. SDM provides a manual to control, organize and carry out projects with respect to information services. System development with the help of SDM is a top-down approach based on four projections: project control, actual system development, change, and validation. Six stages are considered: 0) information planning, 1) definition study, 2) base design, 3) detail design consisting of 3a) functional design and 3b) technical design, 4) realization, 5) introduction and 6) use and management.

Prototyping is more iterative and cyclical, which indicates that the functionality of the DSS is evolutionary in development. Prototyping is a method; prototype refers to the product of development. In this case, the prototype refers to a DSS with restricted functionality. Generally, a prototype is an artifact with restricted functionality. The development is restricted to a particular area of interest. The main purpose of a prototype is not to offer a complete system with complete functionality, but rather to test
the suitability of a particular conceptual framework in terms of adequate support. In our case, merging approaches from CS and OR in order to improve the planning support level within a solid board factory.

Criticism regarding ‘prototyping as a method’ was mentioned by Vicente (1999): i) strong device-dependence and ii) incompleteness. Because prototyping and testing do not avoid the task-artifact cycle, the requirements identified by analysts are limited by the choice of prototypes tested. However, there is no systematic way to move from the results of testing to prototype attributes. A crucial point is that current practice is device-dependent, because it represents the strategies and tasks that production planners using to complete the job using various tools. Thus, prototyping focuses on current work practices with the possibility of being incomplete. Moreover, there is strong device-dependence. Although the arguments of Vicente against prototyping are valid, we believe that for an explorative case study, prototyping is more suitable than a sequential method like SDM, because it considers the feedback of the users more interactively. In addition to this argument, prototyping is also more related to knowledge acquisition methodologies, like commonKADS (see chapter 3) which have been used in research. On the other hand, SDM is more related to classical system development with limited attention to user-friendliness and the support of the current planning tasks.

The DSS development product is called Paperclip: an acronym for Prototyping Advanced PapER & Cardboard Logistics In Planning. This name was inspired by a production planner’s request to have ‘paperclip’ functionality, like post-it notes on a document, in the new planning and scheduling system. At the Cabofa facility, we applied prototyping in the following way. First, we performed a logistical diagnosis and analyzed the logistical and planning concept. In this stage, a functional design was also constructed, specifying the objects within the planning and scheduling domain and their properties. Second, we presented sample screenshots to the planners showing the potential look of the system. Third, following the screenshots and the previous functional design, we enhanced the prototype by providing functionality. In this stage, the original functional design had not been tested; adjustments were performed for pragmatic reasons. Fourth, the prototype with limited functionality was discussed with the production planners, and their feedback eventually resulted in an improved and enhanced prototype version.

The prototype was developed to answer the research questions introduced in chapter 1, with a focus on the third research question. In order to develop a prototype, we specified several functional and technical requirements. Among others, the requirements were derived from the aspects noted in chapters 2-5, and the design criteria were derived from principles observed in literature data. These requirements will be discussed in section 6.2; it comprises a description of both functional and technical design. The system was developed based on the functional and technical design. Section 6.3 discusses the use of the prototype. In section 6.4, we reflect on the most important aspects found in this chapter.
6.2. Functional and technical design

A detailed design is required to develop a planning support system. According to Eilers et al. (1991), detail design consists of two parts: the functional design and the technical design. The functional design specifies what the system should do, whereas the technical design specifies how the system performs the required tasks. Before we begin analyzing the functional and technical design, we will summarize the system boundaries of the Paperclip prototype. In the previous chapters, we analyzed the whole order flow at Cabofa. We concluded that the production planning and scheduling domain was very broad. From a planning support perspective, too many relationships and disruptions exist, which may complicate the development of planning support. We have established system boundaries to the fine planning domain because of pragmatic reasons and the relative autonomy of the fine planning task. We also expect the content of the fine planning tasks to remain relatively stable. These arguments were previously discussed in section 1.3. Thus, planning support is only provided for the fine planning domain by means of a DSS with restricted functionality; this type of planning support is called a prototype.

We developed an overall prototype at Cabofa for the fine planning process, starting with a specification of the functional and technical design. Functional requirements relate to the tasks and processes within the (fine) planning and scheduling stage, while technical requirements convert these specifications such that they can be implemented within Paperclip, the fine planning support prototype. Note that we only discuss the main aspects of functional and technical design.

6.2.1. Functional design

In general, the emphasis within the functional design is on the user and interaction with the user (in our case the production planner). Among other aspects, functional design specifies the functions or tasks supported by the system, the information needs, the data structure, the performance requirements, and user-friendliness. Many of these aspects were discussed in previous chapters. However, in the functional design these requirements are generally specified more explicitly. The structure of this section is as follows. First, we consider several ways to support the human decision maker in an abstract way. We apply a simplified version of the framework (Jorna et al., 1996 and Mietus et al., 1996), which is implemented within a system architecture by van Wezel (2001). Another way to look at planning support is by means of providing a manual or automatic mode. After this discussion, we continue to present relevant design criteria, which are directly derived from pitfalls identified in the literature. Given the fine planning process, the supported tasks within the Paperclip prototype were specified. The current execution of these tasks was discussed in chapter 4. Previously identified restrictions are also discussed. Finally, the tasks are worked out in more detail, and we conclude with an overview of the functional design.
There are several ways to support the human decision maker (John & Kieras, 1996a; John & Kieras, 1996b; Johnson, 1992). van Wezel (2001) proposed a system architecture to support it. Of these, we restrict ourselves to the following: inspector and evaluator, editor, generator, and blackboard. The blackboard stores multiple and partial solutions for later retrieval. Everyone working on the planning problem can view solutions proposed by others. Theoretically, this allows the production planner as well as an algorithm to work simultaneously on solutions to a problem (van Wezel 2001). The inspector and evaluator check the value validity of properties, constraints and goal functions. The editor provides the production planner with an editing facility to eliminate or easily change property values. The editor is the natural interface with the production planner. The generator manipulates new objects or properties that were demanded. More specifically, it provides calculations and proposes new solutions to specific problems. Of these elements, the inspector and evaluator, the editor, and the generator are implemented in the Paperclip prototype. The blackboard is not implemented in the prototype, although it is relatively easy to extend the prototype with this type of functionality. Although the properties of the orders determine to a large degree the complexity of the plan, it is very difficult (if not possible) to use a previous plan to construct the new production schedule. After all, the composition of orders at a particular moment in time is unique. Thus, it can be argued that the parameters of a production plan generally differ; this makes a blackboard less valuable.

On the other hand, the structure of the production plans has certain similarities. In case of repetitive actions and planning schemes that return in time, this type of functionality may help. For our particular application at Cabofa, a blackboard functionality is of minor importance.

Within the Paperclip prototype, two different modes of planning support were implemented: automatic and manual. In the manual mode the production planner creates a (partial) solution for a planning problem in a particular stage of the fine planning process. The system may only provide consistency checks if invalid data is entered. On the other hand, in automatic mode the system applies scheduling rules or algorithms to propose a (partial) solution for a planning problem in a particular stage. The proposed solution is not necessarily feasible. If we relate the automatic and manual mode to the ways of planning support by means of editor, inspector and evaluator, and generator, we can sketch the following situations, which may clarify the relationships.

At first sight, the editor and the inspector and evaluator are related to the manual mode and the generator to the automatic mode. However, if the inspector and evaluator assess the validity of the input, followed by a suggestion to improve the solution or to remove infeasibility, this is also considered a part of the automatic mode. If no suggestion is provided, it is considered part of the manual mode. Apart from the mode types in planning support and the ways it can be implemented by means of the editor, generator, and evaluator and inspector, the sequence of activities within the fine planning domain is important. We argue that the formulation of a feasible production schedule cannot be performed with only stages in the automatic mode. The first stage of the planning process always consists of a manual mode, like selecting which orders
are suitable to be part of a production plan. Later stages can be performed either manually or automatically. Thus, within the manual mode, the planner creates a solution, while within the automatic mode the program provides a solution. The planning process can be divided into several stages, which can be performed automatically or manually; however, only the first stage is always performed manually. The type of planning support can generally be linked to ways of planning support like editor, generator, and evaluator and inspector.

Another aspect that may complicate functional design is the layered structure of functionality. Even within the editor, simple algorithms can be implemented that apply tasks to sort, select, or count values in case of changes. Some of these elements can be considered as an automatic mode of planning support. However, we only consider elements in the automatic mode if the program provides a solution. This is not the case in these above-mentioned examples.

In order to support the fine planning task of formulating new production plans within the automatic mode, we propose a simple scheduling heuristic that follows the partitioning in sets concept. The partitioning in sets concept is based on the idea that human problem solvers start the solution process with the most difficult orders and finish the process with the easiest orders. More concretely, this means that within the fine planning process, the orders that cannot run alone are first planned, ending with the orders that are ‘easy’ to plan. Thus, the concept of partitioning in sets follows the current practices by production planners at Cabofa. For more information about the concept of partitioning in sets, refer to chapter 4 page 68 to page 72.

One of the purposes of the research is to achieve an interaction between human and machine. Particularly in the automatic mode, this is an important issue because automation may reduce the impact of decisions made by human problem solvers and may ultimately complicate the interaction between human and machine. Hence, special attention is required. The way in which the simple scheduling heuristic is implemented in the Paperclip prototype is discussed in section 6.2.2.

The simple scheduling heuristic does not offer optimized solutions with respect to trim losses. However, this is also not its purpose. We introduce the simple scheduling heuristic in order to mimic the production planners’ behavior. It was prompted by an assumption from CS that the content of a planning task crystallizes in time. The current content shows a sensible way to handle the problem. If minimal trim loss is the target, the implementation of more sophisticated heuristics or algorithms can be considered as an extension, and is a subject left for further research.

The manual and automatic mode show strict differences. However, if these differences are viewed in perspective, it is evident that in practice they may blend into each other. This is also motivated by the example presented above which shows that the fine planning process can be divided into several stages, and for each stage (except for the first one) the planner may decide either to apply the manual or the automatic mode of planning support. Apart from the formulating aspect, there is another inter-
estating issue about making final decisions. Our stance is that even in the automatic mode, the production planner should always have the opportunity to reject proposed schedules and start from the beginning or to revise proposed schedules. The planner (human) is always decisive in the planning process.

Human-Computer Interaction (HCI) literature discusses several aspects that may complicate the interface between human and computer. We present some of these observations here, which may help us to avoid some pitfalls during the development of the DSS. We translate these observations into principles and corresponding design criteria that should be satisfied by the system. Note that it is not our purpose to present a complete outline, but to illustrate some dominant observations that may affect the performance of a DSS. The following observations have been identified in the literature:

1. The human is (legally) responsible for the whole human-machine system performance. This necessitates him to monitor the system in case something goes wrong. (Astrom, 1985; Fox & Smith, 1984; Sanderson, 1989; Hoc, 2000 and Barthélemy et al., 2002).

2. Human loss of expertise is the consequence of designing machines that play autonomous roles. This is one of the dangers associated with the human becoming too remote from the process: humans may not have an up-to-date mental picture of current process states; humans may even lose their long-term mental model of system functioning and structure (Bainbridge, 1983; Mitchell & Miller, 1986; Sharit et al., 1987; Endsley, 1996; and Ward, 2000).

3. Human-computer interactive systems often outperform humans or computers individually (Bergeron, 1981; Dunkler et al., 1988; Haider et al., 1981; Sorkin & Woods, 1985 and Barthélemy et al., 2002). Several studies indicate that computers often outperform humans on particular types of task. However, algorithms within a computer system could not always find an optimal schedule (Smith et al. 1997). Benefits of mutual control have been demonstrated or suggested by Clark & Smith (1993) and Hoc (2000).

4. Loss of adaptability: there is often a lack of feedback from the computer system to the human (Hoc 2000). Loss of adaptability may reduce acceptance for the human to use the system within the task execution, which may reduce the accuracy of the data within the system.

5. Complacency / over reliance: the human decision maker may rely too much on the solution proposed by the system. In order to avoid complacency, Smith et al. (1996) suggested incorporating an output of several solutions in the system instead of one. Earlier research in a different application domain (Mietus 1994) incorporated this suggestion of having more solutions in the ZKR system. Generally, complacency may overcome by strictly following formulated working programs.

In order to apply these observations within the functional design, the observation has been translated into principles and corresponding design criteria. A principle denotes
a particular target that should be satisfied during the development process, while a
design criterion represents a way to make the principle operational.
Table 9 summarizes the principles and their corresponding design criteria.

Table 9: Principles and corresponding design criteria

<table>
<thead>
<tr>
<th>No.</th>
<th>Principle</th>
<th>Design criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Let the human have end-responsibility for the whole system performance</td>
<td>Ultimate decisions left to the production planner</td>
</tr>
<tr>
<td>2.</td>
<td>Prevent human loss of expertise</td>
<td>Provide both a manual and automatic mode</td>
</tr>
<tr>
<td>3.</td>
<td>Design an interactive system that outperforms any individual system</td>
<td>Provide both a manual and automatic mode, whereby planner may select among several planning stages</td>
</tr>
<tr>
<td>4.</td>
<td>Avoid loss of adaptability</td>
<td>Provide a predetermined level of feedback (e.g. inspector / evaluator).</td>
</tr>
<tr>
<td>5.</td>
<td>Avoid complacency / over reliance</td>
<td>Provide multiple solutions</td>
</tr>
</tbody>
</table>

The table presented above summarizes some observations from HCI literature relevant to the development of a DSS. These observations have been translated into principles and corresponding design criteria.

Having presented ways to implement the planning support, principles and corresponding design criteria, we continue with a more detailed description of the planning tasks that will be supported by the Paperclip prototype. In order to limit the task set, the current way of working should be noted, see chapter 4. With respect to the fine planning process, we will restrict the planning support system to support three fine planning stages. Note that a planning task refers to a collection of actions resulting in a particular product, for example the construction of a fine plan. Generally, a planning task is divided into planning stages.

To clarify the distinction between planning task and planning stage, Figure 49 shows the fine planning stages. These stages are performed if the fine planning task is used to formulate a production plan. The diagram shown in Figure 49 is similar to Figure 19 in chapter 4.
Figure 49: Fine planning stages in process flow diagram form

From the fine planning stages mentioned in Figure 49, we will support the gray shaded areas, representing determine separate runs (7a.4), determine separate positions (7a.5), and determine arrangement of positions (7a.6). These stages represent the core of the fine planning task when formulating or revising a production plan. Ultimately, these three stages may lead to an approved production plan (7a.7). For a detailed description about how these stages are performed in the current situation, refer to the pages 78 to 84 of Chapter 4. Note that we use production plan, production schedule, run, and fine plan interchangeably to refer to the final product of the fine planning process.

Figure 49 represents the fine planning stages in diagram form. From a planning support perspective, it is important to consider three distinct types of fine planning tasks that depend on the status of a production schedule to implement these stages within the prototype. If no production schedule is available, i) the construction of a new production schedule, the fine plan should be the target. The second type of planning task is ii) the revision of an existing production schedule, which is executed if new rush orders enter the planning process or disruptions on the production machine disturb the original assignment. Finally, the third type of planning task denotes iii) arranging the approved production plans on a particular production machine. It is strongly related to determine arrangement calipers (7a.3). The difference is that the task determine arrangement calipers is executed prior to starting the planning process with a provisional arrangement of the calipers, while after the production plans have been finished, the production planner considers the arrangement again in order to
make a definitive assignment. The main drives in the third type of task are the avail-
ability of lining paper and the due dates of underlying customer orders.

The first type of fine planning task, i) construct a new production schedule, is pre-
sented in Figure 49. However, for the purpose of the functional design, we are more
explicit; the stages to be implemented in the planning support are:

- Proposing the candidates from a set of orders that satisfy some criteria
- Combining the orders that run side-by-side, such that the trim loss within the
constructed run is minimal
- Determining the arrangement of positions within a run

The orders that have been selected are candidates for the run. Given the whole set of
orders that have been considered as candidates, the production planner combines
some orders to run side-by-side. The assigned orders refer to a position. In the third
stage, the arrangement of positions within a run can be altered.

The second type of fine planning task, revision of a production schedule often uses
the same elements as in the construction stage. However, it mainly depends on the
source of disturbances that necessitates the revision. Figure 50 shows the main flows
that we consider in the revision of a fine plan task. It also presents the stages that
belong to the construction of a fine plan.

Figure 50: The construction or revision of a fine plan

The three main sources that guide a revision:

- Entry of rush orders. A new rush order should be delivered in the short term. If
the demanded grade does not run on time after the fine plan is manufac-
tured, the production planner needs to insert the rush order into the pre-
existing production plan. Depending on the dimensions, the order may or
may not run alone. However, it may use all stages that are also applied during
the construction of a fine plan.
• **Change in due dates** for orders that are in the fine plan or revised arrangement of fine plans. If there is a change in due dates for a particular order in the run, it may imply a change in the arrangement of positions. It may also have been caused by the revised arrangement of fine plans. Thus, if an order demands cardboard of 2.0\text{mm}, but the fine plan responding to 2.25\text{mm} is manufactured before this run, it may lead to a need to rearrange the positions within the fine plan.

• **Disruptions on a particular production machine.** Depending on the severity of the disruption, a number of cases may arise: the fine plan should be completely revised; it impacts the arrangement of positions, or it just causes a split of the production plan into two parts. The latter case often happens with knife problems. In this case, orders that run alone are manufactured, and the orders that need to run side-by-side are placed in a new production plan that is manufactured later when the disruption or other problems have been solved. The ultimate result is a split in the production schedule.

The third type of task, *arranging the set of finished production schedules* can be solved in a similar way as the arrangement of positions. Only the abstraction level is different: in the construction stage, the objects to arrange are the positions, whereas in the sequencing stage, the objects to arrange are the finished production schedules. The goal of arranging the sequence of production schedules is to revise the sequence on a particular production machine, if required because of capacity problems or due date violations. The arrangement of fine plans is supported in the following way. The sequence of constructing new production plans denotes the default sequence. If necessary, the production planner may alter the sequence by hand. Additionally, an algorithm may support this stage.

The stages mentioned above show the main elements that are implemented within the prototype. For the fine planning task we consider another more detailed level. In case of constructing or revising a fine plan, the following options are available to perform the fine planning tasks. Figure 51 shows the menu options when constructing or revising a fine plan.

The options in Figure 51 represent methods of constructing or revising a production plan. These options should be implemented in the Paperclip prototype in order to provide the production planner with demand planning support plus an alternative process flow. From an object-oriented point of view, we consider three important objects within the fine planning tasks: fine plan, order, and (alternative) positions. These three objects are necessary, because within a fine plan a set of positions has been specified. Within a position, one or more orders run side-by-side. Thus, these three objects specify the building blocks within the fine planning tasks. With respect to the fine plan, the production planner has the possibility to accept the production plan as formulated thus far. A new position can be inserted or an existing position can be deleted.
Figure 51: Menu options during construction or revision of a fine plan
With respect to order, many options should be available. We present three options. First, we would like to have the opportunity to have a detailed view on the properties of a particular order. Second, the planner demands planning support for a specific order by means of proposing alternative orders to run side-by-side. Third, a manual selection should also be provided.

Planners are accustomed to manual planning by using the third option. If the planner applies the second option and the system proposes good solutions, then additional functionality is provided and this may reduce cognitive load because the system performs all the calculations. If the system proposes a solution, we would like to consider two opportunities: either proposing a solution on the whole set or on a subset of candidates. In addition to scheduling orders for insertion into a new position, we would like to have the opportunity to add appropriate orders to the set of candidates. We consider two options: orders with the same grade and orders with a different grade. Grade differences may only reflect differences in liquid percentage and differences in lining paper. Except for grade, we consider the option to select the same time window as that of the selected candidates, or to select an entirely differently time window. Finally, with respect to alternative positions, we would like to have the opportunity to add an alternative position proposed by the system or to remove a position from the current production plan. Additional functionality is provided under the edit menu. This functionality is related to undoing actions or searching for orders that satisfy particular criteria. Note that these criteria do not necessarily satisfy the properties of the current set of candidates. This is a major distinction with the add an order option to the set of candidates, which shows orders with similar properties.

Besides the fine planning tasks, the system should also provide planning support with respect to other planning tasks. It is not our intention to test the system on these planning tasks, but rather to enable the planner to distinguish among data of the various stages. It should therefore have its place within the Paperclip prototype. Figure 52 shows all tasks available within the Paperclip prototype.

The figure below shows the set of activities that enables the production planner to perform the fine planning tasks. The fine planning tasks within the white box, required to manipulate the set of (finished) fine plans, demands particular attention. These tasks were discussed earlier, and we therefore omit them here.
In order to construct a new fine plan (production schedule), additional information from other processes is also required. From the rough planning stage, we need information in order to provide an overview of all orders. The capacity utilized over the
production machines should also be known while formulating a new production plan. An overview of all orders may help to structure the stages in the fine planning process. As discussed in chapter 4, the production planner performs the rough and fine planning stages simultaneously. The capacity utilized indicates the utilization of the available capacity for the production machines. Bottleneck machines are easily identifiable. The knowledge about capacity utilization may help to objectively discriminate between production machines and to remove imbalances in machine loads.

Thus far we have discussed the tasks directly related to the planning and scheduling domain. In addition to these tasks, the production planner may demand information about properties used in the fine planning stages. For example, the recipe number, which shows the current assignments of recipe numbers to product, caliper, liquid percentage, lining paper (weight and type), and production machine.

Within the functional design, the emphasis was on functional requirements. We discussed several methods about how planning support can be implemented (editor, generator, inspector and evaluator). These aspects were linked to a manual or automatic mode of planning support. The literature suggested several principles and design criteria to avoid some common pitfalls. The core of the functional design discussed the types of (fine) planning stages supported and how it is implemented within the DSS. From the whole range of planning tasks, we have restricted the support to the following fine planning tasks: the construction and revision of production schedules. A planning task, such as formulating a new production schedule, applies to several stages discussed in chapter 4. Of the fine planning stages, the stages shaded in gray in Figure 49 are supported. We later emphasized these planning tasks on a more detailed level and discussed the types of activities that may help to relieve planning tasks. Finally, we discussed other planning tasks that do not belong to the fine planning tasks, but information from these stages is essential. We explained, among others aspects, the use of capacity utilized in production machines and gave an overview of orders to be manufactured within a specific time window. Fourth, we presented an overview of the tasks supported in the prototype.

6.2.2. Technical design and implementation aspects

The functional design specifies functional requirements and restrictions to be considered. We presented some important requirements and restrictions in the section above. The technical design represents the way the functional design will be realized (Eilers et al. 1991). The functional specifications are translated into technical requirements in order to implement them within the Paperclip prototype. Regarding the technical design and the implementation aspects, we discuss the following aspects. First, we start with an explanation of the selected development environment and discuss the guidelines applied during the development. We then present some sample code to implement a particular task within the prototype. We also consider the basic idea of partitioning in sets and discuss how this is implemented. We do not present the sample code here, because it uses several functions that have been collected in a distinct module. In this sense, it is far more complex than the code presented for determining
alternative positions. We conclude the section with a summary of the most important aspects discussed.

Based on the available Integrated Development Environment (IDE) of the Windows platform in 1999, two products dominate the market: Microsoft Visual Basic (VB) and Borland Delphi. Both IDEs can be considered visual Rapid Application Development (RAD) tools. The application of a RAD tool has several advantages: it enables a relatively easy development of a ‘quick and dirty’ sample to illustrate an applied concept, i.e. development is quick.

Given the shortlist of two competitive products, we decided to select Borland Delphi 5 as the IDE for the development of the prototype. This decision was based on the following arguments. First, Delphi follows the object-oriented principles more strictly than VB does. Second, the programming language Pascal, which is in Delphi, is strongly typed, which means that each variable has to be defined and initialized in the header of a module before it can be used. In this sense, Delphi forces more structure during the programming process. With respect to the user interface components, both IDEs heavily rely on Application Programming Interfaces (APIs) provided by the Windows platform.

The Paperclip prototype was therefore developed in Delphi. We now move to the development process and the feedback given by the production planners. The first version of the computer program only had a set number of screenshots, which were enhanced in later versions with additional functionality. In this way, the prototype grows in an evolutionary fashion. The intermediary results were discussed several times with the production planners. Their feedback provided important input for improving and enhancing the functionality within the fine planning domain. As previously mentioned, the iterative process during development consisted of three stages. At an earlier stage, a functional design was formulated based on findings of logistical diagnosis, logistical concept and planning concept. This functional design was used as input in the above-mentioned stages.

To illustrate the implementation, we present some sample code for demonstration purposes. In general, there are three distinct aspects of software development: the user interface to present results, the functions and procedures that code functionality, and the database to store values. It is theoretically possible to run the distinct areas on different machines. Thus, the GUI may be presented on one machine, the decision rules and functionality of the program on another machine, and the data on a third machine. Because our aim is to test the suitability of the proposed framework, the focus is on the user interface and functionality. The database environment, although important, is not considered in most versions of the prototype. This aspect is part of further implementation and research, which may enhance the prototype in such a way that it can communicate with other database management systems using ODBC, ADO, JDBC, native drivers or other interfaces. We therefore intended to separate the user interface from functionality during the development. If a particular action was demanded, it required a particular function or procedure within the respective module.
In the case of collecting separate modules with a clear specification as to what the function or procedure does, software re-use is easier to implement (van Wezel & Jorna, 1999; van Wezel, 2001). It is an important consequence of this type of system development. However, our target is to test the applied conceptual framework, not to test software re-use.

We discussed the choice of IDE, the development and prototyping process, and the distinction between GUI, functionality and database. We now present how the functionality is coded within Delphi. An example is provided for the list of alternative positions. The list of alternative positions shows a selected list of positions in which a selected order can run side-by-side, either completely, or partially, with other orders. A more detailed description is given in section 6.3. provides an abstract of the code for the calculation of alternative positions.

Note that several other functions and procedures in the code have been called. The main idea behind the sample code of Example 8 is the following. First, several initialization actions were performed by calculating and setting particular values based on a number of parameters. We then considered the number of stacks in which the selected order may run on the production machine. This represents a discrete set of options. Given the number of stacks of the selected order, the other candidates can run side-by-side. In the for-loop, we calculate the trim losses that may arise during combination. After the for-loop, the orders were sorted. Finally, in the repeat-until-loop alternative positions were calculated and prioritized. In addition, the corresponding quantities were demanded. The repeat-until-loop stops if the number of alternative options is larger than a particular value or the trim loss is above 15%.

Similarly, the concept of partitioning in sets (See pages 68 to 72 of Chapter 4) was coded. It discriminates among the orders based on their dimensions compared to their trim on a particular production machine. In the prototype, the due date is not a fixed criterion, because the system is not yet operational. We therefore have two distinct sets: orders that are easy to plan and orders that are difficult. The system starts with the orders that are difficult to plan and tries to combine them with another order that is difficult to plan. If this is not possible, another order from the set of easy orders will be selected. The heuristic continues until all orders have been planned against another order; orders that cannot be combined run alone.
Example 8: Sample code for the calculation of alternative positions

```pascal
procedure CalculateAlternatives(Inputs: AllOrders, Restrictions, SelectedOrder; Outputs: SetOfResults);
var {Set of help variables};
begin {Initialization}
  for i:=0 to MaxNStrokes-1 do
    begin
      Performance[i]:=TRelativePerformance.Create;
      CalculateBeginInterval(IndexKM,
        Restrictions.NLKnife[IndexKM]-(i+1),
        Restrictions.MaxPlanWidth[IndexKM,Restrictions.
        NDKnife[IndexKM]]-(i+1)*AllOrders.Format[0,SelOrder],
        Performance[i].BeginInterval,Restrictions);
      GiveOrderingOrders
        (IndexKM,Restrictions.NLKnife
        [IndexKM]-(i+1),
        Restrictions.MaxPlanWidth[IndexKM, Restrictions.
        NDKnife[IndexKM]]-(i+1)*AllOrders.Format[0,SelOrder],
        Performance[i],Restrictions);
    end;
  SuperSort(Performance,Outcome,Restrictions,True);
  i:=0;
  repeat
    StillToPlan:=TotalToPlanQuantity(SelOrder,Restrictions,Init
      Solution);
    SetLength(AltOpst,Length(AltOpst)+1);
    AltOpst[Length(AltOpst)-1]:=TPosition.Create;
    repeat
      ChooseAlternative(IndexKM,SelOrder,RelativePerformance,
        Performance,AltOpst[Length(AltOpst)-1],
        i,Outcome,StillToPlan,Restrictions,True);
      inc(i);
    until (StillToPlan=0)or(StillToPlan<0.15)
  until (i>=Length(Outcome))or(AltOpst[i-1].TrimLoss>0.15)
end;
```

Regarding the technical design and implementation aspects, we explained our choice for Borland Delphi 5. We also sketched the development and prototyping process. The technical design translates functionality described by the functional design such that it can be embedded in the future system. We presented sample code to determine a list of alternative positions and a conceptual outline concerning the way the concept of partitioning in sets has been implemented in the planning support. For other functional design aspects, a technical design was also specified. However, our goal was to
6.3. The use of the Paperclip-DSS

The previous section discussed the functional and technical design. We applied the whole logistical order flow (see Chapter 2), the execution of the planning tasks (see Chapter 4), and the important objects within the planning and scheduling domain (see Chapter 5) into a coherent framework. The functional and technical design (see Section 6.2) specify and restrict the system boundaries considered within the Paperclip prototype.

In order to use the Paperclip prototype, several stages are considered. With respect to the menu structure implemented in Paperclip, we refer to Figure 51 and Figure 52. As previously mentioned, the main emphasis is on the planning support for the fine planning stage, especially for the construction or revision of production schedules. We therefore start by presenting the main activities during the production schedule construction and showing how the planning tool provides planning support to the production planner. The first stage in the fine plan construction consists of proposing the candidates to participate in the run. The production planner looks for candidate orders for the run. Paperclip supports this functionality by means of selection criteria. Figure 53 shows a screenshot of how the selection criteria are presented in the prototype. Note that the selection process is always performed manually and cannot be supported automatically.

![Selection of orders](Figure 53: Selection criteria for the construction of a new run)

The candidates for the run are selected by means of two selection criteria: recipe number and delivery time. Production planners usually apply a weekly time horizon. However, market trends show a tendency to smaller order sizes, shorter due dates, and restricted variety mix. These trends place more emphasis on flexibility, in additional to planning and scheduling (see Chapter 2). Therefore, the production planner may
select the delivery time for the candidates within a specific time interval of several days instead of weeks. Both types of time intervals are supported. The recipe number embeds five parameters: caliper, cardboard machine, liquid percentage, weight and type of lining paper. For the current prototype, we omit the weight and type of lining paper as selection criteria, because in practice these criteria are used less frequently compared to the first three parameters. It should not be difficult to enhance the prototype with these selection criteria; it will not significantly alter the applied conceptual framework. A set of orders that satisfies the selection criteria is created. It denotes the orders that are candidates for selection in the newly constructed run. Given this set of candidates, the production schedule can be formulated.

Paperclip offers planning support for the formulation or revision of a production schedule in one fine planning screen. We would like to discuss the overall functionality provided by the system and relate it to how planning support can be implemented (editor, generator, inspector and evaluator). After the general functionality discussion in Paperclip, we continue to describe in more detail the second and third stages of the implemented fine planning tasks. Note that the stages were: i) proposing, selecting or revising candidates, ii) combining orders that run side-by-side (also called ‘trimming’), and iii) determining the arrangement of positions. However, we start with a description of the overall functionality within Paperclip. Figure 54 shows the fine planning screen.

![Figure 54: Fine planning screen](image)

**Figure 54: Fine planning screen**
The fine planning screen of Paperclip consists of three data blocks (in technical terms: ‘frames’). The menu offers menu options with particular functionality as previously discussed in section 6.2.1 (see Figure 51). The first frame in the upper-left quadrant is labeled ‘Orderset’. It consists of all orders that satisfy the selection criteria specified in Figure 53. These orders are the proposed candidates for implementation in the production schedule. If the planner demands additional orders, he may insert them in the set of proposed candidates. In this frame, general properties of the set of orders are presented, such as demanded quantity, planned quantity, customer name, dimensions and the direction of manufacturing (short grain versus long grain).

The second frame in the upper-right quadrant is labeled ‘Alternatieve opstellingen’, which represents the alternative positions proposed by the system. Given a highlighted order (in Figure 54 the order with order number 44482-1), the alternative positions have been calculated. The list of alternative positions consists of positions, which according to the system, result in an acceptable trim loss if that particular position is inserted in the production schedule. Note that a position consists of a set of orders that run side-by-side. The technical design in section 6.2.2 showed the sample code in Delphi that supports the functionality of providing a list of alternative positions. The system also specifies the slitter knife on which the order runs, the corresponding quantities of sheets or weights for the order. In this way, complexity is reduced.

The third frame in the lower quadrant is labeled ‘Productieplan’. It shows the current status of the production schedule. Here, the positions that should run on the cardboard machine are specified. It specifies on which knife the order runs, the number of stacks that the order occupies on the cardboard machine, the quantity to be manufactured, and the sequence of the positions. Several manipulation actions are possible.

The quantities are measured in two dimensions. The production planner may select a preference. The calculation units that can be selected are kilograms and number of sheets. In practice, production planners still think in terms of tones, a multiple of kilograms. This originates from the common standard in the paper and cardboard industry where tones are still the common denominator. However, from a practical point of view controlling the planning process in number of sheets or square meters of product may lead to fewer disturbances.

In the functional design, we have already discussed several ways to support the planner. They can be implemented by means of a generator, editor, or inspector and evaluator. The generator returns in two ways: 1) the construction of a complete initial production schedule based on a simple heuristic; the initial production schedule generator, and 2) the formulation of the alternative position list, given a selected order. For a more detailed description of the content of the generator component, refer to section 6.2.2 where both the sample code of determining the list of alternative positions and an outline of the initial production schedule generator is presented by means of applying the partitioning in sets concept. The evaluator / inspector comes to the
fore if a particular value within the production schedule is altered. It evaluates the
goal functions and restrictions. If in Figure 54 the number of stacks of order 44482-1
is altered from 1 to 2, the planning width becomes larger than allowed. The evaluator
provides a warning message ‘Aantal gewenste stroken kan niet i.v.m. overschrijding
planbreedte’ and reverses the change. The editor is in the lower quadrant labeled
‘productie plan’. Here, the parameters can be changed to other values. Additional
functionalities, such as sorting and calculating within the production schedule, are
considered a part of the editor. The editor cannot be applied within the other quad-
rants. All we can do here is select a particular order or alternative position from which
we may decide to insert or remove it from the current production schedule.

Having discussed the overall functionality by means of the fine planning screen, we
would like to return to the second and third stages in the production plan formulation.
Below, we will discuss these in more detail. In the second stage, the production plan-
ner combines one or more orders side-by-side, such that trim loss is acceptable. Sup-
pose we start from scratch to construct a new production schedule. Then, we have a
set of proposed candidates. Given a highlighted order in the upper-left quadrant, the
system proposes several alternative positions in the upper-right quadrant. Figure 55
presents such a situation.

![Figure 55: Second stage: order side-by-side](image-url)
In Figure 55, we present a highlighted order with order number 44058-3 and the corresponding alternative positions. The number of kilograms is the selected calculation unit. According to the system, order 44058-3 may run side-by-side with order 44801-1. Then, the total number of stacks on the cardboard machine is equal to 3; 1 stack for order 44058-3 and 2 stacks for order 44801-1. According to the program, 3566 kg of the first order correspond to 8338 kg of the second order. In total, the selection of this order will result in a trim loss of 6.9%. If this position satisfies the production planner’s criteria, it may be inserted into the production schedule; the position will be listed as first in the sequence of positions. We may also want to delete the proposal from the list of alternatives. Both the insertion in the production schedule and the deletion from the list of alternative positions are provided by the system. Another option in this stage is to call the initial production scheduler, which demands the system to plan all orders automatically and return with a proposed initial production schedule. Figure 56 shows the result of this option.

![Figure 55](image)

**Figure 55: Second stage: plan all orders automatically**

All orders have now been planned. This can be seen on the sixth column (‘Nog te plannen’) of the upper-left quadrant (‘Order set’). All proposed candidates show a value equal to zero, which means that they have been planned completely. In contrast, Figure 55 shows all values excluding zero, meaning that all proposed candidates should be planned. This is logical, because the production schedule is empty. The insertion of positions to the production plan from the list of alternative positions and
the initial production schedule generator show two possible options in the second stage to determine which orders should run side-by-side.

The last stage in the fine planning construction consists of the determination of the arrangement of positions within a run. In the fine plan, Paperclip offers the ability to rearrange the sequence of positions. As we have explained in chapter 4, the mill production department prefers a sequence of positions in which the orders are manufactured without interruption. This constraint is taken into account in the initial production schedule generator. If the production planner constructs the production plan by selecting appropriate alternative positions, he is responsible for satisfying this constraint. Additional checks can be implemented, but have not yet been implemented. The sequence of positions within Paperclip can be altered manually. Thus, if position 1 and 10 should be exchanged, the system will perform this action. In addition, it is possible to delete a particular position within the production plan.

After combining several orders side-by-side in positions and arranging the sequence of positions, the fine plan usually consists of a number of positions. If the size of the run is above a specific value, the production schedule may be finalized. Proposed candidates that are not (partially) in the fine plan must be assigned (partially) to another run. The production schedule has now been finalized. The planner may formulate a new production schedule or revise an existing production schedule, for example the previously accepted production schedule.

Besides methods to support the fine planning tasks by offering a generator, editor, or inspector /evaluator, we also consider two modes of planning support: manual and automatic. The definition of these concepts was presented in section 6.2.1. What does manual or automatic mode mean in the Paperclip prototype? In automatic mode, the program provides a (partial) solution. The program either formulates a feasible production schedule, such that all proposed candidates are planned in the fine plan (initial production schedule generator), or presents a list of alternative positions that may plan the selected order (partially) side-by-side with other orders. The lower quadrant, labeled ‘Productieplan’, shows the constructed production schedule. When calling the initial production schedule generator, it consists of several positions for which all proposed candidates have been planned. If all orders are planned and they show a zero value in the sixth column ‘Nog te plannen’ of the upper-left quadrant, then the list of alternative options (upper-right quadrant) will not show any entries.

In manual mode, the production planner creates the solution. The starting point is either an empty production schedule or a proposed initial production schedule that is to be revised. In manual mode the DSS does not provide any support by means of proposing a (partial) solution to a particular problem. The DSS validates the entered or modified data in both modes; this functionality does not depend on the type of mode that has been used. If the system proposes a solution to remove infeasibility in case of infeasible data, we deal with the automatic mode.
In practice, the planner will use the automatic and manual mode alternately. For clarification purposes, we present a list of possible activity flows in Figure 57.

(1) Automatic

(2) Manual

(3) Manual → Automatic

(4) Automatic → Manual

Figure 57: Activity flows in the DSS

Note: if the activity flow consists of more than two stages, the flow represented by (3) or (4) can be followed by (1) or (2), etc. The first stage, selecting which orders are considered as candidates, is always performed manually. Given various criteria, the list of proposed candidates is constructed. The planner may add or remove additional orders. Construction of the production schedule then starts. The initial production schedule generator constructs a production schedule automatically by applying a particular algorithm. This proposal may be accepted. Then, the schedule is finalized. This corresponds to the following flow: manual (selection criteria) – automatic (initial production schedule generator) – manual (finalize). Another way to continue after the selection of proposed candidates is to select an order and evaluate the list of alternative positions. The selection process is manual, but the alternative list of positions is constructed automatically. An alternative list may be added to the production schedule. This corresponds to the following flow: manual (selection criteria) – manual (selection order to plan) – automatic (alternative list of positions) – manual (select alternative position) – etc.

The production planner may then alter the production schedule by altering values for a particular order position. These activities are manual. The DSS supports the planner by recalculating the values when changed. During production schedule formulation, the planner may alternately apply the manual and automatic mode, which was illustrated by the two examples presented above. Figure 57 presents a possible sequence of activity flows during planning task execution. In general, the first stage is always manual. The next stages can be either automatic or manual. The list of activity flows presented in Figure 57 always starts with either (2) or (3); if more than two stages are required, it can be followed by (1), (2), (3), or (4).

In summary, Paperclip supports the fine planning task in the following way. First, the production planner reduces the list of candidates by specifying the selection criteria.
Then, based on this set of candidates, the planner continues to construct a production plan. The fine planning screen provides the production planner with a worksheet that helps him during the three stages of the formulation of a fine plan. The system provides automatic and manual modes to support the production planner during the fine planning task execution. The system also evaluates entered or modified values in the production plan and rejects inconsistent data.

6.4. Reflections on the development and prototyping process
The main topic of this chapter deals with the development and prototyping process of a DSS. It includes a functional and technical design specification, the way the design is implemented in the DSS, and how the DSS can be used in practice. We will now assess the development and prototyping process. The structure of this section is as follows. We will start by discussing the functional and technical design criteria in relation to the DSS. We continue with a discussion as to what extent the DSS supports current work practices. What elements from the applied conceptual framework have proven useful and what elements are not useful? Thereafter, we will evaluate the development process as a whole and the way the feedback of the production planners has been used. We will conclude by drawing general conclusions based on the applied conceptual framework.

6.4.1. Functional and technical design: criteria and implementation
In section 1.3 we introduced two starting points for system development: ‘system follows user’ and ‘user follows system’ (see page 13). We demonstrated that the ‘system follows user’ approach should be used during development. We discussed the extent to which this approach is satisfied. Several design criteria were specified in section 6.2. After the completion of the prototype, we will consider whether the DSS satisfies the specified criteria. We use the same structure as the above-mentioned sections.

With respect to the adopted framework, we demonstrated the ‘system follows user’ approach. This means that we attempted to uncover and enhance the existing decision practice by supporting the decision maker in his attempt to formalize decision intentions (see Wanders et al., 1999). In our opinion, the prototype offers sufficient flexibility to use the DSS in a way that the planner demands. Besides the current way of planning task execution, additional functionality was also implemented. In practice, the additional functionality offered was appreciated because it provides new opportunities to evaluate planning problems. Tasks are dynamic to a certain level. This leads to the application of new strategies that may affect the use of the DSS and ultimately may result in added demand for information.

Three ways of planning support were described in the development process: editor, inspector, evaluator, and generator. All of these elements have been implemented in the DSS, as discussed in section 6.3. In the current version of the DSS, only the
evaluator and inspector check validity; they do not propose alternatives, which may remove the invalidity. These types of extensions are left for further research. The editor is only present at the production schedule frame. The generator exists in two types: i) construct the initial production schedule and ii) construct a list of alternative positions. Of these three elements, most emphasis during development was on the generator, although in practice the demarcation lines are not that strict.

Based on the five principles and their corresponding design criteria derived from literature (see Table 9), we conclude the following:

1. *Allow the final responsibility be directed to the human.* This means that the DSS always leaves the ultimate decision to the production planner. The DSS only provides support by relieving the calculation tasks within the formulation of a production plan stage. The implementation in the DSS is consistent with the specified design criterion. The monitoring task in case something goes wrong lies primarily with the production department, where the operator should reschedule. If problems are structural, feedback reverts to the production planner, and fine planning rescheduling becomes relevant. The DSS supports several stages by proposing a (partial) solution or checking the validity of the data.

2. *Preventing the human loss of expertise* is satisfied if the planners regularly perform all stages of the planning process. A way to reduce the probability of human loss of expertise can be achieved by providing both a manual and an automatic mode. However, if the production planner only applies the automatic mode, this principle is not satisfied. In practice, people would like to hold final responsibility and possess many options in performing the human task. Although slightly contradictory, people also heavily rely on checklists, which ultimately may reduce the human loss of expertise. On the other hand, the introduction of planning support generally enables the planner to do more in the same amount of time; several scenarios may be considered when expanding the human expertise (Mietus 1994).

3. *Designing an interactive system* is satisfied in the following way. If both a manual and automatic mode are provided and the planner may choose among planning stages, then the system is more or less interactive, depending on the complexity of the planning problem and the preferences of the production planner. This principle is somewhat related to the second principle. The concepts mentioned there also apply here.

4. *The loss of adaptability is avoided* by providing a sufficient level of feedback. Particularly in the manual mode, if the planner constructs his own schedule the system may verify the validity. This type of task is directed to the evaluator and inspector. The system provides feedback in case of violations, and it also indicates the source of the violation. If an initial schedule is automatically generated, the production planner must always have the opportunity to rearrange and manipulate the data or to restart the scheduling process from scratch. However, this aspect may be improved in later research.

5. The principle *avoiding complacency or over reliance* denotes the production planners’ behavior to overly trust the proposals without modification based
on relevant arguments. This element is particularly relevant to persons with restricted planning experience, as observed at Cabofa’s Sand site. The design criterion proposing additional solutions could reduce complacency (Smith et al. 1997). We reduced complacency by offering several ways to formulate the production schedule in two modes: automatic and manual. In the case of manual mode, the production planner creates the solution; complacency is not an issue. In the case of automatic mode, the DSS provides a solution. It is implemented in two ways: either a complete initial production schedule or a list of alternative positions. The system only provides one complete production schedule. The complacency argument therefore remains unsolved. However, as an extension it can be extended relatively easy.

On the whole, Paperclip satisfies all principles derived from observations in literature data (except for complacency in automatic mode when the initial production schedule generator is called). In our opinion, this violation is not problematic, because the proposed production schedule is considered as an initial schedule, rather than being considered a definite schedule.

With respect to the technical design, we outlined procedures applied during the development process. The DSS was developed in Delphi, a RAD tool. The development of the prototype followed an evolutionary approach. The first version of the computer program only consists of a number of screenshots, and it was enhanced by additional functionality in later versions. The intermediary results were discussed at various times with the production planners. The prototype was adjusted using their feedback. Finally, all three selected fine planning tasks were implemented. Although only these elements were discussed in detail, the final version of the prototype provides a stable basis to enhance the program in a direction and move it towards application in an actual production environment. Note that the tool also provides additional functionality, such as proposing lists of alternative positions, calculating quantities, and proposing a (partial) solution for the production schedule. This additional functionality was not available in the previous situation (old programs).

The combination of supporting the current planning tasks and offering new additional functionality makes it hard to judge these elements solely on their value. The lack of supporting the production planner in real life situations makes it even harder to test the validity. We demonstrated that the prototype supports the current planning task, and there is no need to alter the content of the planning task. However, the introduction of new functionality and a reduction in time used making calculations may ultimately lead to a new organization of the (fine) planning tasks, an aspect that is often neglected in system development with methods from CS. This is called the paradox of support (Jorna & van Wezel 2001). Another aspect that often applies is the reduction of time necessary to perform the same type of tasks, resulting in a dynamic change in the content of the planning tasks. Introducing a DSS may result in more time available to perform additional scenarios or alternative evaluations.
6.4.2. Concluding remarks

In this chapter, we formulated the functional and technical design requirements in section 6.2. Within the functional design, we restricted the system boundaries of the Paperclip prototype to the fine planning stage on the cardboard machine; other tasks were not considered. The technical design translates the functional design criteria such that they can be coded. Section 6.3 continues to present the functionality of Paperclip by discussing how it can be used to perform the planning tasks. We concluded that many of the current fine planning tasks are supported by the prototype. Besides the current planning task, the system also provides additional functionality. Section 6.4 deals with the development process, and we concluded that most design principles have been satisfied. When violations occur, it should be relatively easy to cancel the violation. Overall, we concluded that the level of planning support was improved by the introduction of a planning tool like Paperclip. Current planning tasks are supported sufficiently, and the tool provides additional value by introducing new functionality and new means to perform the planning tasks with more sophisticated strategies.