Title: Development of a Knowledge-Based Scheduling and Quality Monitoring Software

Course:
Year: 1993
Author(s): P. Byrne

Report No: P93041

Abstract: Here we detail the development of an intelligent scheduling and quality monitoring software package for a law firm. The scheduling aspect of the software forecasts preparation time of a batch of legal documents. The quality monitoring aspect determines the probability that documents within batches will be returned for corrections. The software has been termed intelligent because the system updates its rules to reflect changes in the values of particular variables.

Development of a Knowledge-Based Scheduling and Quality Monitoring Software

P. Byrne

EMP-P9341

DEVELOPMENT OF A KNOWLEDGE-BASED SCHEDULING AND QUALITY MONITORING SOFTWARE

P. Byrne

EMGT 506 - Spring 1993

ABSTRACT

This project involves the development of an intelligent scheduling and quality monitoring software for a law firm. The scheduling aspect of the software forecasts the time to prepare a batch of legal documents. The quality monitoring aspect determines the probability that some documents within a particular batch will be returned for corrections. The software has been termed intelligent because it is possible for the system to update its rules to reflect changes in the values of particular variables.

Preliminary results are encouraging. The software predicts a slightly longer time to completion, but is sensitive to the batch size and the document complexity. Additional testing must be carried out to determine how accurate the software is in predicting the possibility of errors.

TABLE OF CONTENTS

INTRODUCTION

Identification of the Problem

A small, local law firm is involved in the preparation of legal documents for an estate planning agency. The firm expressed interest in determining (to a more exact degree) the time needed to process a batch of documents, and in improving the quality of the documents. A large amount of information was readily available in the form of old client files, but the firm lacked the staff needed to continually monitor and update the information that could be obtained from these files. A computer program was proposed as a possible solution. The ideal program would be highly graphical in nature and user friendly, as the primary user is somewhat uneasy with computers. The program would need to be able to update itself, preferably in a manner transparent to the user.

Scheduling problems lend themselves very well to computerization [1]. There are a vast array of popular software packages devoted to project management, and scheduling in particular, and many offices take advantage of these tools. Quality control has also been implemented with the assistance of computers [2, 3, 4]. Due to the amounts of data generated by the information gathering for quality control and statistical process control, computers are almost indispensable for the analysis of the data generated.

Law firms traditionally make use of software of a different nature (specialized word processors, etc.) and popular opinion tends toward the feeling that lawyers are reluctant to implement total quality. The computer program developed for this project presents a combination of scheduling and quality monitoring.

Background Information

The firm receives the applications for the documents, processes them, prints them, makes final preparations, and ships them. All of the documents are similar, but the preferences of some clients make it necessary to spend additional time with document preparation. In summary, the steps of the process are as follows:

1. Applications for the documents are received.
2. Once client names are known, personalized binders are ordered.
3. Secondary merge files for each client are prepared.
4. The documents are merged, edited, and printed.
5. The documents are shipped.

Delays have been known to occur during any of these steps. There have been occasions when the firm has failed to receive applications that are shown on the packing list. There have been printing errors with the binders. A client may have special needs and the file preparation could become a lengthy process. The printing process may be delayed due to a lack of paper or toner cartridges. There have been misdirected and lost shipments. Some of these delays, such as the problems with shipping, are infrequent. Others, such as printing problems, are easily prevented through planning. The most frustrating delay is caused by complex documents, which are not avoidable, but are where a major amount of time is consumed.

The agents involved in the initial sale of the documents are eager to receive and deliver them to the clients. At present, they are instructed to inform the clients that delivery can be expected from four to six weeks after the firm receives the applications. As previously mentioned, the firm would like to be able to give the agents a more accurate (and shorter) time estimate for delivery.

Unfortunately, there are times when a document contains errors. The majority of these errors are in the spellings of names and addresses. Other errors have occurred where the wrong individual has been named to a position of authority by the document. When the firm is informed of an error, a correction is printed and delivered. In such situations, it is important to track down the source of the error. There are three possible sources:

1. The Client - Sometimes the clients decide that they want to designate different people to positions of authority in their document. This is not considered to be an error, but rather a change. At the present time the firm does not charge the clients for these changes.
2. The Agent - Some agents are notorious for careless spelling errors and poor penmanship. On occasion they omit vital information which must be known prior to the document preparation.
3. The Firm - Although it is rare, the firm has been known to make errors in the preparation of the secondary merge files.

The firm determines whether the correction is due to a client change or an error by examining the secondary merge file and the original application.

An intelligent software would be able to forecast the time needed to complete a batch of documents and to forecast the possibility of receiving corrections. The factors that affect the amount of time to complete a batch of documents include the number of documents in the batch, and the complexity of those documents. The chance that corrections will be needed in a particular batch depends on which agents are represented. The past record of the agents must be examined. After an actual batch has been completed, additional data (the actual time needed) will serve to modify the knowledge base, making it "intelligent". This modification reflects the learning process.

Information Gathering

The information that is needed for the scheduling aspect of this project includes the size of the batch (the total number of documents per batch), the number of complex documents per batch, and the time to complete the various tasks (this may vary depending on the batch size and the complexity of the documents). For the quality monitoring aspect of the project, the necessary information pertains to the individual agents' propensity for errors. A major portion of this information was available from a data base that was being constructed for record keeping purposes. Not all of the information in the data base was fit for use because it included some very old, incomplete records. The data base that was used to obtain initial time estimates and agent information contained 730 records. This is believed to be an adequate sample, as our typical volume at the time the data was being gathered was from 15 to 30 documents per week. Volume

has increased to an average of 40 documents per week. Some agents were not well represented by the sample because of their own low volume, and were excluded from this study. Additional time estimates were obtained by asking the individual responsible for document preparation. Those estimates were remarkably close to the value for the total expected time obtained by manipulating the data base, but the variance obtained using the information in the data base was much larger. Information from that individual was indispensable, as no obvious relation could be deduced between the document complexity and the task completion time.

To begin the scheduling aspect of the software, a simple network diagram of the tasks was constructed (Figure 1.). The optimistic, mean, and pessimistic times for the tasks that were obtained from the human expert are shown in Table 1. The table also lists the expected times and the variances for the tasks. The total expected project time and the total variance are also shown compared to the values obtained using only data base information.

The time estimates shown in the table reflect the times required for an average batch of 40 documents, with no complex documents. Complex documents do not affect the time estimates for the ordering of binders, the merging, editing, and printing, the packaging, or the shipping. In the expert's opinion, the total time for secondary merge file (SMF) preparation is increased by one day for each complex document.

The numbered tasks shown in the figure and table are as follows:

1. Applications received - this was not really a task, and shouldn't take up any time, but it may be considered to if there is some delay in sending out the binder imprinting list or starting the secondary merge file preparation, so it was included for clarity.

2. Ordering of binders.
3. Preparation of secondary merge files (SMF).
4. Merging, editing, and printing of files.
5. Packaging of documents.
6. Shipping of documents.

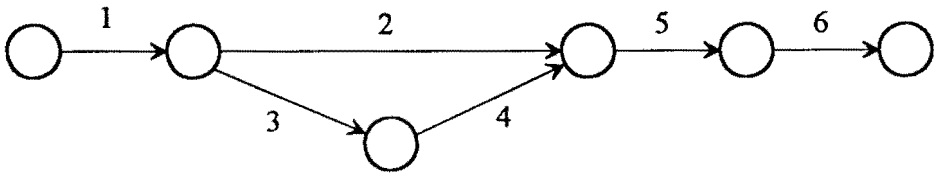

Figure 1. Network diagram of tasks.

Task	a	m	b	t_e	V
1	-	-	-	-	-
2	2	3	5	3.167	0.25
3	1	3	10	3.833	2.25
4	14	25	30	24	7.111
5	1	1	2	1.167	0.278
6	2	5	7	4.833	0.694
	Total values:			33.833	10.333
	Data base values:			31.139	117.4

Table 1. Time estimates (values are in days).

The quality monitoring aspect of the project involved analyzing the existing data base to obtain a "probability for making errors" level for each agent. This level was simply the ratio of the number of corrected documents to the total number of documents for each agent. As can be expected, only a few of the agents were shown to be habitual offenders, and most of them had very low probabilities for errors. However, it is important to note that the data base used for this analysis was the same one used to obtain the time estimates. Figuratively, it was a very clean data base. The true error probability levels are suspected to be somewhat higher for all of the agents, which means that the bad ones are much worse than they appear. The main reason for this is that many corrections are not properly filed or recorded.

SYSTEM DEVELOPMENT

Expert System Development

Since this project involved collecting and manipulating expert knowledge, it seemed appropriate to follow some sort of expert system design. The software was developed through the use of an expert system development shell. The particular shell selected was LEVEL5 OBJECT. This software operates under the MS Windows operating environment. A shell was selected for several reasons. First, it speeds up the development process because the inference engine (the manner in which the rules are processed) is built-in. Second, many shells are object oriented softwares, and the development of knowledge bases proceeds smoothly due to the organization that is forced upon the developer (classes, attributes, instances, and so forth). The third reason - relating to the brand of the shell - was that a swift development process was expected due to the developer's previous experience with the shell.

Several types of inference engines are used in expert systems. The most popular are forward chaining, backward chaining, point-to-point hypertext, procedural, and hybrid [5]. In forward chaining, or data-driven inference engines, the data is known beforehand and the goal or answer is found by obtaining answers to questions presented to the user at execution time. This is an appropriate method to use when there are many possibilities or outcomes depending on the data. Backward chaining or goal-driven inferencing begins with an assumed goal and works backwards in an effort to determine if the proper goal was assumed. This is appropriate when there are few goals or answers to investigate. Point-to-point hypertext is a user-driven type of inferencing which is often employed in the construction of interactive point-and-click graphical

(user-friendly) interfaces. Procedural engines are very similar to computer code, and are good for numerical applications. A hybrid engine would be a combination of these [5].

For this project, point-to-point hypertext was used for the user interface, and a combination of forward chaining and procedural inference engines was used to construct the knowledge base. The point-to-point hypertext was selected because it is unintimidating - the user simply selects choices that are offered and rarely needs to use the keyboard to enter special commands. The procedural type of engine was used for processing the mathematical aspects of the program, and forward chaining was used for directing the user interface based on user response.

The objects of the knowledge base represent packages of information. Objects are grouped into classes, attributes, and instances. An instance is the lowest or most specific level, and is used to represent a current value for an attribute [5]. An instance can hold one value at any time. An attribute may be better described as a variable. If it is assigned a value, that value is an instance of the attribute, but the attribute may possess more than one instance at one time. A class collects attributes beneath it. It is really more of a structure, and does not hold any values by itself.

One advantage of programming in this manner is the ability to use class inheritance. It is best used when a large number of objects have a similar structure. For example, in the initial rule base of this software, 39 agent classes are represented. Each one of these classes needed attributes to represent the agent data. Instead of creating identical attributes beneath each agent class, a separate, generic class was created to hold these attributes. Each of the 39 classes then

inherited the one generic class. One point of interest - if separate attributes were created under each of the 39 classes, they would need different names.

Rules and the Learning Process

After the information gathering and the initial data analysis was completed, a major task was the incorporation of a learning capability. It was desired that the learning process would not have to be initiated by the user, but would be an automatic function of the system. This was because the intended user does not have the necessary knowledge to update the system. The first approach to developing a learning process was to manipulate the rules after detecting whether or not modifications were necessary. This method of rule manipulation can be illustrated by starting with a simple IF...THEN... rule:

IF modifications to rules are needed = TRUE
THEN begin modifications := TRUE

The rule illustrates a check (=) of the value for *modification to rules are needed*. The antecedent "asks" if the object is equal to TRUE. If the value for this variable is TRUE, the conclusion of the rule is fired, and in this case, another variable, *begin modifications*, is assigned (:=) the value of TRUE. Another sort of rule commonly used, a WHEN CHANGED rule, is more procedural in nature. This type of rule fires whenever its reference attribute has been given any assignment [5]. The firing occurs even if the attribute has been assigned with an identical value, so the name WHEN CHANGED is somewhat misleading. It is preferable to use a separate rule to modify another rule because some instability can occur if the value of an attribute changes during the

execution of a rule in which it is referenced. In the example that follows, a WHEN CHANGED for *begin modifications* is shown. In this rule, there are checks for the type of modification that is needed. This rule will branch to other rules (to carry out time modification or confidence modification) when it needs to, then it will return and conclude itself.

```
WHEN CHANGED begin modifications
    BEGIN
        IF time span modification is needed = TRUE
        THEN modify time := TRUE
        IF confidence level modification is needed = TRUE
        THEN modify confidence := TRUE
    END
```

The preceding examples of rules illustrated one way of modifying the knowledge base. The method that was actually used in the project was not as elegant but was more reliable. The method involved converting all relevant instance values into strings and writing them to a text file at the end of a session. At the start of a subsequent session all of the data was converted back to its original value type; intervals (time), numbers, simple (T/F). This data was written back to the knowledge base after the program had gone through its own initialization process.

The initialization unfortunately would set all instances to the last value provided *during development*. That is acceptable if the rule base is not expected to change, but a primary objective of this project was to make the rule base modifiable. Since the rules are referenced to specific instances, the outcome of the rules depends on the values of those instances. By writing back the last value of an instance obtained *during execution*, the rule base has been effectively modified.

There are several reasons that this method is perceived to be more dependable than writing rules to modify other rules. The first is that there are fewer rules overall and less can go wrong during execution. Secondly, the debugging process is simplified because the developer does not have to determine if problems exist due to the modification process or due to the rules that were modified. Finally, and probably the most binding reason, was that the developer determined that the software development shell constrained the ability to modify rules once they were written. This was no doubt considered to be a benefit to developers, to prevent corruption of their knowledge bases.

SAMPLE RUN

To demonstrate the program, a sample run is included here in the form of screen shots.

The first screen to appear is a standard title screen and is shown in Figure 2. The software name, company name and restrictions on the use of the software are shown. The button located in the lower right hand corner of the screen has two purposes. First, it activates a command for the system to read the data file containing the "old" knowledge. After all of the instances have been reinitialized with their previous values, the user is routed to a subsequent screen. Presently, if the system is unable to locate the data file, the user is routed to a null screen. It is possible to incorporate a foolproof file searching method, but the programming overhead is substantial.

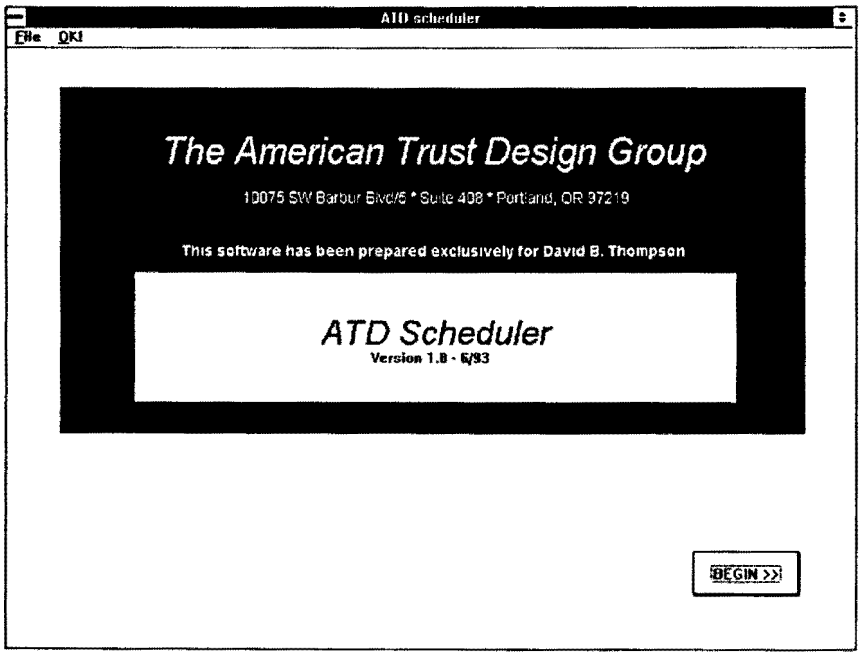

Figure 2.

The second screen of the system is shown in Figure 3. There are two options as to how the information will be entered into the system. The graphical tool used is called a radiobutton,

and only one of its items can be selected at a time. When a second item is selected, the first is deselected. The selection made at this point determines which screen appears later.

It is preferable to have the data loaded into the system through a file instead of manually entering it. The reason for this is that the manual information entry screen shown on the next page is rather busy - it contains a great deal of detail. The alternative was to split the screen into three or four separate screens. This alternative was not selected because a cumbersome data entry process defeats the benefit of a swift point-to-point hypertext user interface. Another reason that manual entry is not preferred is the possibility of error. If a number is accidentally entered for the wrong agent, the value calculated for the error probability will be affected as different probabilities are determined for each agent. In addition, the data file written at the end of the session will be corrupted with this information and the error will propagate each time the program is executed. By using a checked and edited file, these errors can be avoided.

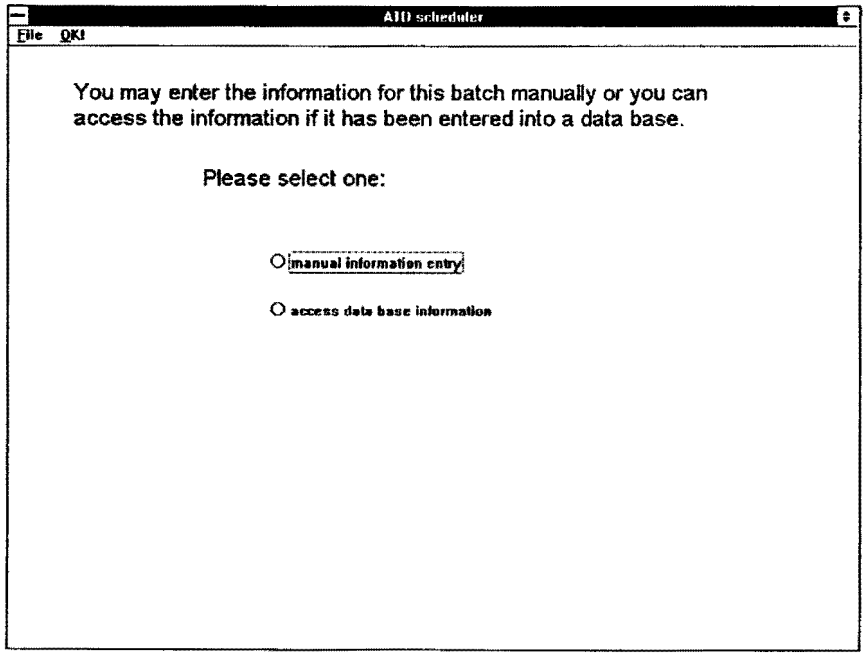

Figure 3.

The manual information entry screen is shown in Figure 4. As was previously mentioned, it is crowded. This screen prompts the user for the total number of documents in the batch, the number of documents that are considered complex, and the number of documents that each agent has in the current batch. Not all agents will submit documents for each batch. Once all of the information in entered, the OK! in the upper left hand corner is selected to begin processing. If additional agents are hired, an additional screen would be necessary.

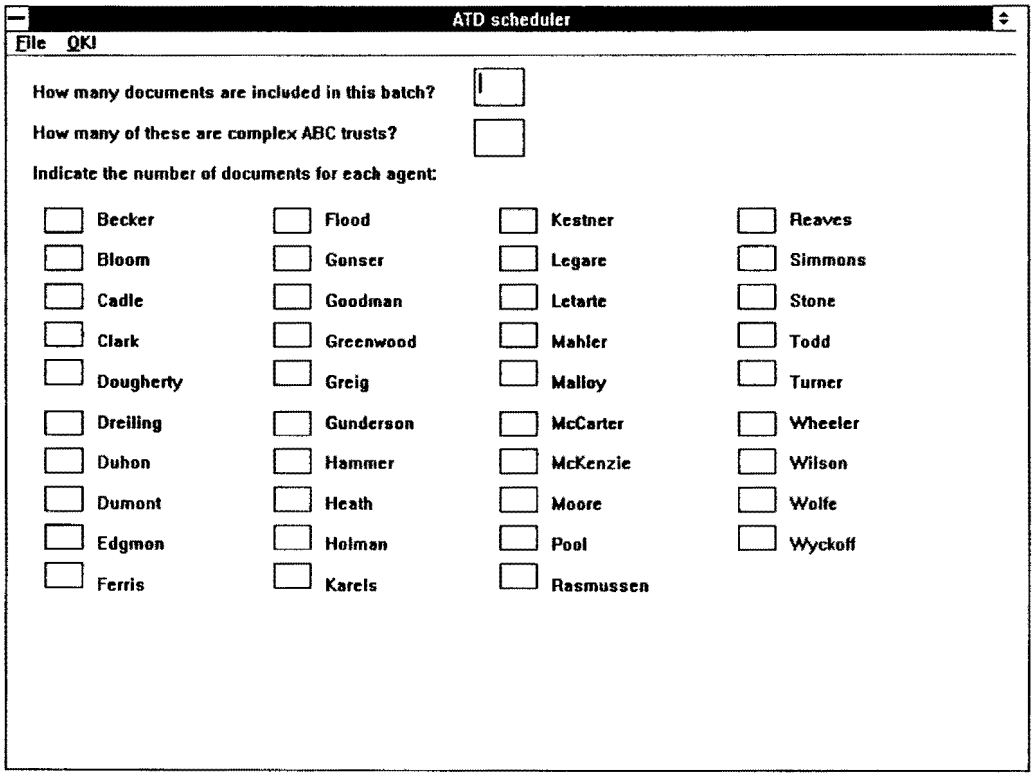

Figure 4.

If a data base contains the information that the system needs, input is less tedious for the user. The screen on the next page shown in Figure 5 shows the screen that appears if the user selected the option to access the data base information. The name of the file is typed into the

space provided. It is important to include the path of the file, or the software will only look in the directory in which it resides. The system will search for the file after the OK button in the center of the screen has been selected. If the file cannot be found, the user is again sent to a null screen. Once the file is located, the system proceeds to open and read the file. The information is assigned as instance values, but this new information occupies a different instance than the old data that was loaded at the start of the session. The file that was loaded was for an imaginary batch dated June 1, 1993. There were 30 documents and 12 of them were considered complex. Three agents were represented, each with 10 documents. One had a high error rate (about 25%), but the other two had reasonable rates (one was around 10%, and one was zero).

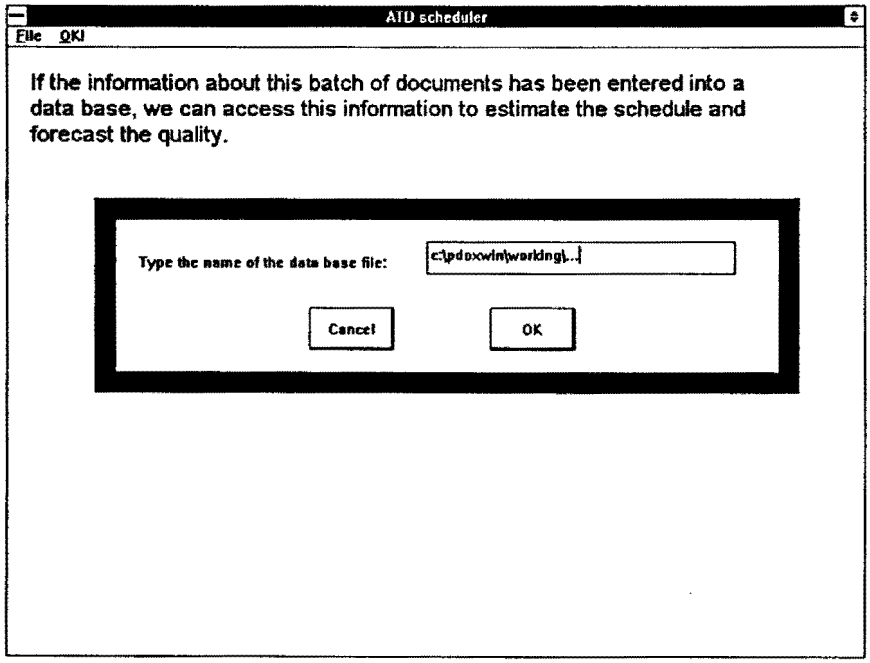

Figure 5.

The screen shown in Figure 6 is the final screen of the program. If shows the estimated times for the tasks. The lengths of the bars are specified by defining pixel locations for the four

corners of a box. These pixel locations have been determined by the intervals needed to complete the tasks. The top and bottom borders of the boxes do not change; only the left and right borders. The dates written along the top of the screen are determined by the date given for applications received. The actual dates that were produced are as follows:

Applications received on June 1, 1993

Binders ordered and SMF preparation begin on June 1, 1993

Binders received June 4, 1993

SMF preparation completed June 15, 1993 (takes 15 days)

Document preparation begins June 16, 1993

Document preparation concludes July 14, 1993 (takes 29 days)

Packaging takes place on July 15, 1993

Shipping begins July 16, 1993

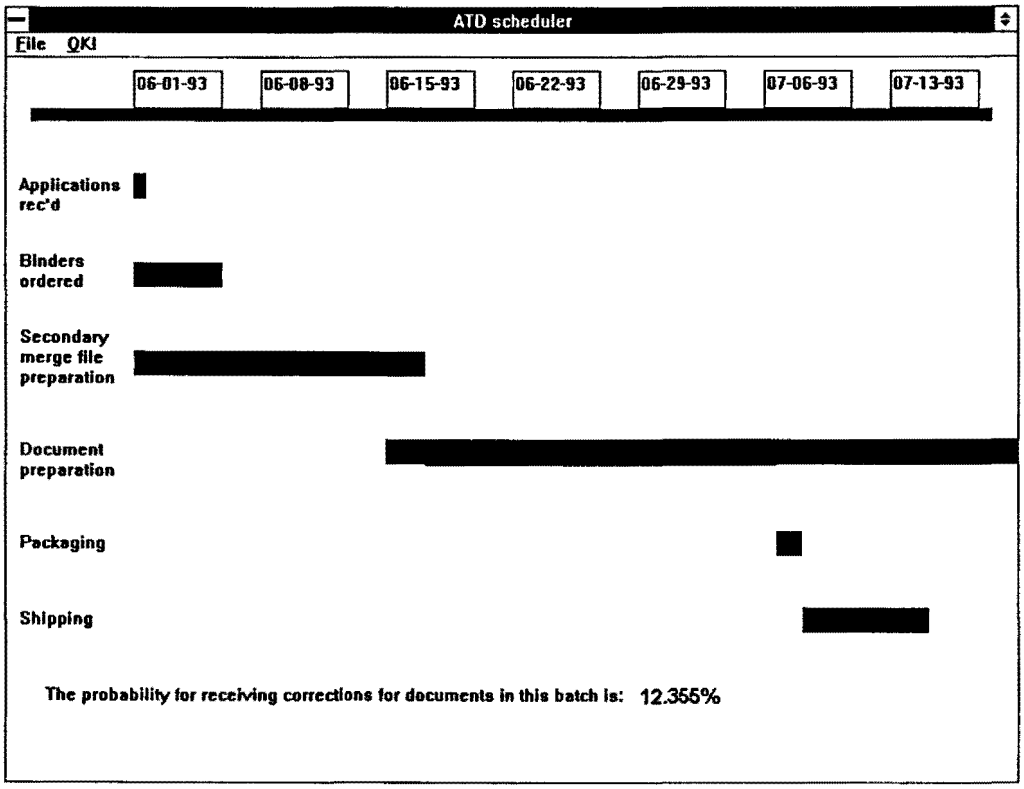

Figure 6.

The developer discovered that the software needs to be adjusted to account for tasks that extend beyond the right hand side of the screen. The user can always scroll to see the remainder of the screen, but it was decided that this is not appropriate.

The relevant task times were calculated as follows:

$$\text{Task } 3 = \left[\frac{x}{10}\right] + y \qquad \text{Task } 4 = 7 + \left[\frac{x}{20}\right] + y$$

The variable x represents the total number of documents in the shipment, and the variable y represents the number of complex documents. The result of this equation is the number of days to complete the task and is rounded to the nearest integer. Other tasks are not considered dependent upon the number of documents, and are not calculated separately for each batch.

At the present time, the software does not account for holidays. In developing the system, the effects of overlapping batches was not considered. However, the time estimates from the human expert and the data base must have accounted for this, since our batches always overlap.

VALIDATION

Testing and Refinement

Testing and refinement is an essential part of any software development [6, 7, 8]. It is appropriate to begin testing as early as possible, since a large untested program may slow down system development. Testing for this program involved both the user interface and the rule base. The user interface was tested to determine if it behaved correctly given the system input. This process was carried out simply by viewing the sequence of the screens. The testing of the rule base involved running a *history* file. The history file shows every step that the knowledge base takes, and tracks every single line of the rules that are fired. It can also be used to track the user interface. Conducting a session with a history file running simultaneously dramatically stretches the execution time, but it is invaluable when it comes to debugging.

Refinement is appropriate at a later point in the development process, but it should not occur so late that it generates severe changes in development. Refinement is a type of clean up work that is mostly cosmetic. Refinement on this project involved revising the layout of the screens. An appropriate color scheme was also selected. On screens that prompted the user for information, the prompts needed to be clear and not too crowded. For this project, one of the screens was unavoidably crowded. On screens that provided information, the font had to be large enough to read.

Verification

Closely related to testing is verification. While testing can be considered successful if a program is free of bugs and relatively foolproof, verification answers the more important question, "Does it work?" To verify this program, several batches of documents that were not included in the data base that was used for initial analysis were used as "guinea pigs". The relevant information was presented to the knowledge base via a separate data base file. The results shown in the final screen were compared to the known time for batch completion. The probability for receiving corrections was to be compared to the latest information available on corrections received, but at this point nothing has been received. The results are shown in Table 2. Run #1 was for a batch of 40 documents with 3 complex documents. Run #2 was for 20 documents (no complex), and Run #3 was for 35 documents (no complex).

Task	Run #1	Run #2	Run #3
1	-	-	-
2	3.167	3.167	3.167
3	7	2	3
4	24	13	19
5	1.167	1.167	1.167
6	4.833	4.833	4.833
Total time:	37	21	28
Errors:	9%	16%	4%
Real time:	28	17	26
Real errors:	Unknown	Unknown	Unknown

Table 2. Time comparisons (values are in days).

The total time values reflect the fact that task 2 is carried out simultaneously with tasks 3 and 4, and the time for 3 and 4 is always longer than that for 2. The results seem to show that the software provides a slight overestimate of the time required for completion. This information can be used to modify the "old knowledge" data file. After comparing to the actual records, task 2 was found to have been both underestimated and overestimated, but that would not affect the total task time because it happens simultaneously with tasks that take a longer time to complete. Tasks 4 appears to be the culprit.

CONCLUSION

The results from the preliminary verification are encouraging. Previously, agents were informed that documents could be expected in 4 to 6 weeks. Although the software is predicting a similar time, it is sensitive to the batch size and the complexity. Further investigation must be carried out to determine how accurate the software is in predicting the possibility of errors. The file writing capability can be exploited to monitor progress in this area.

The software currently represents 39 agents. That is how many agents were represented in the data base used to assist in system development. That data base only represented about 65% of the total agents. At this time further development would be needed to represent all of the agents, but this would not severely affect the structure of the knowledge base.

Recently the firm implemented a policy concerning the document corrections that can be traced to agent error. Those corrections will now cost the agents $5.00 per page. We hope to make the agents more careful about information validation. The software will be used to monitor the overall probability of errors by tracking the progress of the individual agents.

REFERENCES

1. Wiest, Jerome D., and Levy, Ferdinand K. A Management Guide to PERT/CPM, Second Edition, Prentice-Hall, Inc., 1977.

2. Miller, L. "Computerized Quality Control System," Food Technology, July 1991 , Vol. 45, No. 7, p 102.

3. Duncan, Robert M. "Quality Forecasting Drives Quality Inventory at GE Silicones," Industrial Engineering, January 1992, Vol. 24, No. 1, pp 18-22.

4. Anonymous. "Software Facilitates TQM Program at Union Pacific Railroad," Industrial Engineering, April 1992, Vol. 24, No. 4, pp 25-26.

5. LEVEL5 OBJECT Reference Guide. 1990, Information Builders, Inc.

6. Gallagher, John P. Knowledge Systems for Business, Prentice-Hall, Inc., 1988.

7. Delgado, Rafael F. "Planning For Quality Software," SAM Advanced Management Journal, Spring 1992, Vol. 57, No. 2, pp 24-28.

8. Pitman, B. "Total Quality Management for Information Services," Journal of Systems Management, July 1992, Vol. 43, No. 7, p 18.