Flexibility of standard software packages for production/inventory control

Citation for published version (APA):

Wortmann, J. C. (1992). Flexibility of standard software packages for production/inventory control. *Production Planning & Control*, 3, 290-299.

Document status and date:

Published: 01/01/1992

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Flexibility of standard software packages for production/inventory control

J. C. WORTMANN

Abstract. This paper investigates first why flexibility is needed. Continuous organizational change appears to be a major driving force for changing information systems. Two main types of flexibility requirements are identified, namely, changes in operational procedures and changes in decision making. Three types of flexibility requirements are discussed, namely, syntactical, semantical and pragmatical flexibility. The following conclusions are drawn:

- *Syntactical flexibility* is in principle obtainable by available software engineering principles
- *Semantical flexibility* is difficult to obtain and is constrained by the investment in application software
- *Pragmatical flexibility* is in practice constrained by investments in existing data and in organizational learning, and not by application software. This holds especially for changes in decision making

1. Introduction

Control systems for production and logistics are usually implemented by means of information systems. Software packages such as MRP systems and OPT are information systems where a hard-wired control approach is built-in. But even JIT, which is primarily a manufacturing philosophy and not a control system, is sold nowadays as a software module. Thus, there is a close interconnection between control systems and information systems. Many practitioners could not even tell the difference between control systems and information

systems. Because of this close interconnection, it is fruitful to study information systems and control systems together.

Many information systems in production control and related areas are nowadays based on standard software packages. The reason for this state of affairs is simple: standard software packages are cheap in comparison to home grown applications, they are usually well documented, immediately available, and contain the accumulated knowledge of many preceding implementations. For this reason, the study of information systems is largely focusing on standard software packages.

Furthermore, some standard software packages can be used as a platform for further home grown developments within a particular company. Such a strategy to expand, modify, or improve a standard software package requires *flexibility* of such a package. This paper investigates the potential flexibility of standard software packages in more detail.

First of all, the question why flexibility is needed is addressed in Section 2. Next, a taxonomy of types of flexibility is given in Section 3, this section also presents a general architecture of standard software packages. Four types of flexibility are distinguished, namely syntactical flexibility, semantical flexibility, pragmatical flexibility and portability. Portability is briefly discussed in Section 3. The other three types of flexibility are discussed in Sections 4 to 6. The paper's conclusions are summarized in Section 7.

Author: J. C. Wortmann, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

JOHAN C. WORTMANN studied industrial engineering and management science at Eindhoven University of Technology. He has been active in the development of information systems for production/inventory control since 1973 and wrote a doctoral thesis on the subject. He has been involved in a number of practical applications, both in component manufacturing and in assembly operations. He worked in the first half of 1985 as visiting professor at Rutgers University, New Jersey on databases. Dr Wortmann is currently employed at Eindhoven University of Technology as professor of information systems for production management.

2. Why flexibility?

Perhaps the most quoted disadvantage of standard software packages is the statement, that a particular package cannot be tailored to the needs of a particular company. Why do companies want to tailor information systems to their particular needs?

Before discussing this question in more detail, it should be noted that flexibility is also a desired feature of home grown information systems. Stated differently, companies tend to *keep changing* their information systems. Ironically, this fact often turns out to be an argument in favour of standard software packages, because these packages have more built-in flexibility than home grown systems.

The fact that information systems are being changed continuously is well known in the field of information systems development. It emerges at many places as the so-called 'application software maintenance backlog'. In practice, it is sometimes tried to avoid information system's changes by improved communication with users in the early phases of a systems development project. To some extent this is a useful and sensible effort, because low user involvement in early phases will result in high costs in later phases.

However, it is an illusion to aim at elimination of all information system's changes. This is basically due to the fact that information systems should reflect the actual organizational behaviour as much as possible. Consequently, organizational change should be mirrored by information system's change. This point will be reiterated several times throughout this paper.

There are two main types of organizational change which should be mirrored by information system's change.

- changes in the operational procedures,
- changes in management and decision making.

Apart from these two sources of *user-driven* change requests, there are also two sources of *supplier-driven* change requests.

- change requests due to the wish to improve functionality,
- change requests due to improved software technology.

The first type of supplier-driven change requests is in many respects similar to user-driven change requests. The second type of supplier-driven change requests relies largely on the modularity and the internal architecture of the reusable software components. This topic is beyond the scope of this paper, therefore we will concentrate on user-driven change requests.

2.1 Changes in operational procedures

The operational procedures refer to the way in which basic logistics actions are implemented organizationally and technically. The term 'basic logistics actions' refers to such activities as customer-order entry, work order release, material issue, purchase order release, and engineering-change control. Operational procedures are subject to a continuing improvement process, because these procedures reflect the adaptation of the production system to its environment. The change of operational procedures can be primarily driven by the business itself, or it can be driven by technical innovation. An example of a business-driven change is given in the appendix. An example of a technical innovation is electronic data interchange (EDI). Quite often, technical innovation and business-driven changes go hand-in-hand, e.g. JIT and EDI.

2.2 Changes in management and decision making

There are several reasons why changes in the way of management and decision making should be reflected by changes in the information system. The most important reasons are.

- evolution of a particular decision making job,
- increased experience of a particular decision maker,
- improvement of a decision-support algorithm.

A proper example of evolution of a particular job is given by Den Boer (1990). In this paper, the job of a material planner is analysed. The material planner is supposed to be the primary user of MRP I, because (s)he should release orders planned by MRP I, and react to exception messages for scheduled receipts. The paper points out that this job consists of two parts, namely

- setting parameter values for lot sizes, safety stocks, etc
- making sure that materials are available when *really* needed.

Usually, the emphasis in the job definition is on the second task, which leads to a lot of decision support for handling exception messages. In most standard software packages there is hardly any systematic support for the first part of the job. Den Boer argues that due attention for this first part (parameter setting) might improve the efficiency and effectiveness of the second task (reacting to exception messages). If companies agree to this view, they would require that the information system gives support to parameter setting.

The increase of experience of a particular decision maker often leads to other requirements for information. As argued by Bertrand and Wortmann (1981), a new decision maker wishes much redundancy in his reports, in order to be able to make cross-checks and in order to improve his understanding of the process to be controlled. An experienced decision maker wants reports on request. If we proceed with the example of the material planner from the previous paragraph, an inexperienced planner would like to rely on general operational research techniques for, say, safety stock levels. An experienced planner would like to have signalled the out-of-stock history of comparable items from different suppliers plotted against the historical values of the safety stock norms. (By the way, most MRP packages do not keep track of the out-of-stock history of an item, and do not keep track of the historical values of parameters!)

Changes in decision-support algorithms are also an important source of change requests for information systems. However, these change requests will normally occur only in those cases where advanced users of decision support systems understand the shortcomings of standard algorithms. Interesting examples can be found in the papers of Van Donselaar (1992) and Wijngaard and Zijlstra (1992) in this issue. We will not elaborate on this matter here.

Concluding the discussion on the reasons why flexibility is needed, we should make one final remark. A distinction was made between operational procedures and decision making in this section. This distinction is somewhat arbitrary. For example, work order release can be seen as a decision, but there is usually also an operational procedure associated, because of the clerical procedures and transaction processing required for work order release (especially in MRP).

3. Types of flexibility

3.1 Introduction

Flexibility of information systems for production/inventory control is a complex matter for two reasons. First of all, these information systems tend to be rather large, in terms of software size. Therefore, a simple scheme for a general architecture of these information systems will be presented in this section.

Secondly, there are several types of flexibility to be distinguished. We will introduce four types of flexibility in this section, namely.

- portability
- syntactical flexibility
- semantical flexibility

- pragmatical flexibility

These types of flexibility will be discussed in depth in the remainder of this paper.

3.2 General architecture of production control information systems

The architecture to be described in this subsection is derived from Bertrand *et al* (1990), Chapter 6. The general architecture of production/inventory control information systems is depicted in Figure 1. This figure shows four concentric circles.

Systems software The innermost circle of Figure 1 contains the run-time environment of the information system. This environment contains all application-independent software packages which should be available before application programs can run: an operating system, a database management system, an input/output monitor, a query language facility, etc. The systems software is important here, because the portability of the application software is determined by the systems software together with the hardware.

State-independent transaction processing systems The second layer in Figure 1 constitutes the state-independent transaction processing systems. The term 'state-independent' refers to the state of orders, materials and capacities in the production system. Therefore, the 'state-independent transactions' are concerned with recording of products, technology, manufacturing equipment and

Figure 1 The four concentric circles of software

personnel, which is supportive to recording and planning of orders and materials. State-independent transaction processing systems allow the recording of standard products, bills of material, routings, capacity types, standard lead times, and so on.

In this layer, finally, we also locate transaction processing systems which enter or change all kinds of parameters. These parameters should be initialized to get a properly working information system. Similarly, a Gregorian calendar is needed together with a working days calendar and a periodic reporting calendar.

In Section 2, two main causes for flexibility requirements were identified, namely.

- changes in operational procedures, and
- changes in decision making

It will be clear, that flexibility in state-independent transaction processing is mainly triggered by changes in operational procedures around engineering data. These procedures are part of *engineering data management*. They are concerned with such issues as the control of engineering changes, or the control of documentation of an installed base of sold products (sometimes called configuration management).

State-dependent transaction processing systems The third layer of Figure 1 contains application programs which monitor the state and state-transitions of materials, orders, and capacities. Materials are received, inspected, stored, issued, transformed, consumed by assembly, restored, shipped, and so on. Customer orders may start as prospects, and are transformed into confirmed orders, shipped orders, invoiced orders, and finished orders. Internal work orders and orders for suppliers have similar life cycles. Equipment is acquired, installed, maintained, and disposed. Operators are hired, they are trained, they may become ill, they may have holidays, and they leave or retire. All these facts are recorded by state-dependent transaction processing systems.

This layer also contains software for storage and retrieval of forecasting and planning figures. Of course, these data are often generated by decision support systems, located in the next layer. However, these data have to be maintained in a database for later reference, regardless of their source. Whether these data are computed by a formal model or entered manually, is irrelevant here.

Flexibility requirements for state-dependent transaction processing software stem primarily from changes in operational procedures around customer orders, work orders, or materials. These flexibility requirements should not be underestimated. Many companies which move from batch production to just-in-time do change

from work order driven operations to operations without work orders. This requires a significant adaptation of transaction processing software. In addition, changes in decision making usually imply changes in state-dependent transaction processing software.

Decision-support systems and structured decision systems The fourth layer in Figure 1, finally, is the layer of decision support systems, which should support a human decision maker. Furthermore, this layer contains the structured decision systems which are able to perform a decision-making process without human intervention (but usually on behalf of a human decision maker, for example a lot-sizing computation). This fourth layer is the focus of interest of most researchers when discussing for example (beyond) MRP. The requirements for flexibility in this fourth layer are usually triggered by changes in decision making, such as discussed in Section 2.

Although decision-making procedures are quite important for information systems, their weight tends to be overemphasized in textbooks and courses on production control and logistics. Both in software development and in practical implementation, most effort is put in transaction processing systems. From an information systems point of view, therefore, transactions processing systems (and especially the underlying data structures) are the key to understanding the system. It is nevertheless wise to consider also the decision-support algorithms, because these tend to influence nature of transaction processing.

These relationships are explored more thoroughly in Bertrand *et al.* (1990). We will explain the above concepts now for the case of MRP packages.

3.3 Architecture of standard software packages based on MRP II

When describing standard software packages based on MRP II, there is a considerable risk of misunderstanding. On the one hand, there are many standard software packages available nowadays, which offer much more functionality than what would be included in MRP II according to the APICS understanding of the term. Such packages might support multi-plant operations, customer-order driven operations, distribution planning and control, extended shopfloor control, support of just-in-time approaches, and many other topics.

On the other hand, there are packages aiming at special branches of industry, such as process industry, subcontracting business, construction work, or transportation, where the functionality of MRP II is not offered although it is advertised. Note, by the way, that usually the functionality of MRP II is simply not applicable in

Figure 2 MRP packages depicted in the circles of Figure 1

these branches (see the paper by Bertrand and Wortmann (1992) in this issue)

In order to avoid this misunderstanding, the treatment of MRP packages below will be confined to topics with follow straightforwardly from the MRP II approach

Figure 2 depicts the architecture of MRP II packages in terms of the circles of Figure 1. Note, this figure is split up into two areas. the upper part represents the goods flow control for a factory as a whole, whereas the lower part represents ‘production activity control (PAC)’ The latter area is also known as ‘shopfloor control (SFC)’ or ‘production unit control (PUC)’ The last term is used here, in accordance with Bertrand and Wortmann (1992)

State-independent transaction processing in MRP systems State-independent processing in MRP systems deals first of all with product information. All products (‘items’) are considered to be completely defined standard products A standard bill of material is known for each product. These data are mandatory for running the MRP I algorithm, as is well known Apart from these standard products, it is possible in MRP systems to use several types of standard, non-physically existing product types, such as product families, kits, and pseudo-items For a more comprehensive description of products and bills of material, see the paper by Van Veen and Wortmann (1992) in this same issue

In addition to product information, MRP II systems employ information about capacities and capacity usage

This occurs usually at two levels of detail, namely, the master production schedule (MPS) level and the capacity requirements planning (CRP) level

At the MPS level, the description of available capacity and of required capacity per unit of output is given by rough-cut capacity planning (RCCP) data (Unfortunately, these RCCP-data are depicted in Figure 2 in the state-dependent circle, in order to stress the connection to the corresponding decision support module)

At the CRP level, the description of available capacity is given in much more detail The description of required capacity uses the routing information which will also be used at the shopfloor

Finally, the state-independent layer of transaction processing is also concerned with storage of parameter data (PAR in Figure 2) and with calendar data, relating the Gregorian calendar to the working days in factories and in transportation (CAL in Figure 2)

State-dependent transaction processing in MRP systems State-dependent transaction processing covers, in MRP systems, usually the following categories of data.

- sales forecasts at various levels of aggregation,
- accepted customer orders and prospects;
- plans and schedules (including the actual MPS and alternatives),
- storage and inventory data,
- workorders (planned, firm planned, and scheduled receipts),
- workorder details (routing, actual vs planned execution data),
- status of personnel, equipment, tooling,
- quality information (measuring data, tracing and tracking data)

Decision support systems and structured decision systems This last layer covers the well known decision support in MRP II such as mater production scheduling (MPS), rough-cut capacity planning (RCCP), material requirements planning (MRP I), capacity requirements planning (CRP), and input–output planning (IOP)

The first two modules, by the way, provide support for the MPS decision, whereas the last three modules provide support to the work order release decision Thus, there is no one-to-one mapping between software modules for decision support and decision types in practice.

3 4 Types of flexibility

There are several types of flexibility which are distinguished in the remainder of this paper Before intro-

ducing these types, we should say a few words on *portability*.

Portability refers to the question whether a software package and the associated data can be transferred from one processing environment to another one. Until about five years ago, most standard software packages for production/inventory control had a very low portability: these packages were rooted in a particular operating system and consequently they were tied to a particular line of hardware.

In recent years this situation has changed. On the one hand, software suppliers have obtained the technology to build their applications more independently of the operating systems involved. One major help in this respect has been the emergence of database management systems, and their appearance in different processing environments. On the other hand, operating systems such as UNIX have become available on hardware from different suppliers (especially workstations and minicomputers) which have made it possible to develop software independently of hardware suppliers.

Apart from portability, three types of flexibility are distinguished here:

- *syntactical flexibility*
This concept refers to the possibility to change the form in which data are acquired, stored, processed, communicated or displayed.
- *semantical flexibility*
This concept refers to the changeability in the meaning of attributes, entity types, messages, or data processing procedures.
- *pragmatical flexibility*
This concept refers to the changeability of the usage of data, in terms of actions which are allowable or compulsory and in terms of the level of detail which is appropriate.

These three types of flexibility will be discussed in more detail below. However, it seems worthwhile to state at this point in very broad terms the main messages with respect to each type of flexibility.

- *Syntactical flexibility* is in principle obtainable by available software engineering principles.
- *Semantical flexibility* is difficult to obtain and is constrained by the investment in application software.
- *Pragmatical flexibility* is constrained by investments in existing data and in organizational learning, and not by application software.

4. Syntactical flexibility

The syntactical flexibility of a software package refers

to the question whether this package can handle its data in any form which is wished by the user.

A classical example of syntactical changeability is the representation of calendar dates. Are dates represented in the format MM/DD/YY, or in DD/MM/YY, or in YY/MM/DD? Are months specified by their names or by numbers? and so on. A professional software package has the flexibility to handle all these situations. Reading a particular type of date input according to American standards and displaying a particular type of date output according to French conventions should also be supported.

Almost just as classical is the requirement for flexibility in using product-identification strings. About a decade ago, standard software packages would have a maximum length for product-IDs of, say, 25 characters. It has been the experience of each successful package that there is no theoretical limit to some customers' wishes with respect to the length of product-IDs. Consequently, the package should support whatever the customer requires.

Basically, these requirements hold for all other data as well. Differences in languages, in supplementary information on screens or in messages, differences in codes and many other considerations lead to a continuous request for changes in the forms in which data is acquired, displayed, and communicated. In turn, these requirements also have some impact on the way in which data is processed and stored. A product-ID with a length of 40 characters has (slightly) different processing and storage requirements compared to a product-ID of 25 characters.

In Section 2 we distinguished between flexibility requirements due to changes in operational procedures and flexibility requirements due to changes in decision making.

First, consider changes in operational procedures (see the appendix for an example). Changes in operational procedures will usually lead to differences in transaction processing systems, because the sequence is changed in which various steps of a transaction are executed. This will normally lead to various changes in transaction processing systems involved, together with changes in input/output screens (or other I/O media). Thus, syntactical flexibility is needed.

Next, consider changes in decision making which affect information system. Whatever the reason is for a change (change in the job, change in the decision maker, improved algorithm) this will always result in requests for other human-computer interaction, and therefore for at least syntactical flexibility.

Is it possible to maintain this level of support for syntactical flexibility? Fortunately, it is indeed. The basic idea is, to have all input/output procedures designed in such a way that they refer to a predefined format for each

field. This format should be predefined in a 'format database' which is accessible and changeable by the end user. In this way, the form of an input field or output field can be specified completely independent of the application software. A similar solution can be used for storage problems. This technology is well known in database management systems. The so-called three-level architecture of databases distinguishes between an internal schema, a conceptual schema and several external schemes (Pels and Wortmann 1990, Date 1989).

Also, in the context of EDI, it is nowadays possible to obtain a high level of syntactical flexibility: there exist standard tools which extract logistical messages from a database in any form and translate these messages in the format required. For automated communication, this means quite a relief.

For human-computer communication, the above technology is not always satisfactory. For example, a screen which has been designed for human-computer interaction can be notably affected if fields with a length of 12 characters are extended to 40 characters, because of a format change. The application software should notify the user in case of such consequences. This type of notification logic requires considerably more functionality than the normal 'format-database' technology. However, the required technology is being developed rapidly, for example in connection with executive information systems.

5. Semantical flexibility

5.1. Introduction

The semantical flexibility of a software package refers to the question, whether the *meaning* of attributes, entity types, messages and computational procedures can be changed easily by the users of such a package. Let us first give some examples.

In the early days of MRP packages, there was hardly more functionality in these packages than the support of MRP I itself. Consequently, many companies started to use MRP I for all kinds of functions for which it was not meant. In order to be able to aggregate planning numbers of end items into meaningful totals for families, some companies defined family items as *components* of end items. In order to be able to disaggregate planning numbers for families into planning numbers for end items, other companies defined family items as *parents* of end items (using planning bills). In both cases the *meaning* of the entity type ITEM is extended. The question is not: is this possible? Rather, the question is: what disadvantages are encountered in such an unanticipated usage?

A more realistic example in today's situation is the fol-

lowing. Suppose a standard software package has defined its forecasting and order-entry data as entities which are uniquely related to items. Now a particular company, using this package, wants to enter forecasts per item-client combination. The reason could be, that each client has its own peculiarity, or that clients are willing and able to enter their time-phased requirements by EDI.

The solution seems simple: the company should define the item-client combination as a new item (see Figure 3).

In fact, this 'trick' changes the meaning of the entity type ITEM. In the view of the user this entity type might, under specific circumstances, not just represent an item but represent an item for a particular client. Again, the question is: what are the disadvantages of such an unanticipated usage?

Let us consider this case in somewhat more detail. The disadvantages become clear for example in releasing an order for shipment to customer 1. Probably, customer 1 expects item 211. However, item 211 is not in stock, so it cannot be shipped. Thus, after using some human intelligence, the company will ship item 210. More human intelligence is required to make sure that client 1 receives a forwarding notice which specifies item 211. Still more human intelligence is required to make sure that the invoice does also specify item 211 instead of item 210. And so on.

This example can be generalized into a general rule. If software is used in a tricky way, which implies a change of the meaning of entity types and attributes in the conceptual schema of the database, there is a continuous human effort required to compensate for the unexpected side effects.

Is it easily possible in the above example to change the software? Perhaps in such a way that forecasts are always related to client-item combinations, and not to items? The answer to this question is dependent on the amount of software available which makes use of the relationship between forecasts and items. If only one or two software modules do use this relationship, modification is easy.

Figure 3. Changes in the meaning of the entity type ITEM by introducing items for specific clients

However, the more sophisticated the standard software package is, the higher the probability of many modules being involved.

5.2 What is semantical flexibility?

We are now in a position to give a more precise treatment of semantical flexibility. A distinction is needed here between transaction processing and decision support.

For transaction processing, semantical flexibility refers to the possibility to change database constraints. More specifically, semantical flexibility is concerned with the constraints which should be specified in the conceptual schema (in contrast with syntactical constraints which should be specified in the internal and external schemas).

These constraints mostly take the form of so-called functional dependencies, which specify that for each occurrence x of an entity type X , there exists a unique occurrence y of an entity type Y . For example, 'each work order is meant for precisely one item' is a well known functional dependency in most MRP systems.

The requirement that a bill-of-material structure should not contain cycles, is also a constraint which *should be* specified in the conceptual schema. This constraint should be specified, because many application programs are critically dependent on the acyclic nature of the BOM. Such an implicit assumption about the semantics of a database should preferably be made explicit. Although this requirement is not realized in current practice, it is possible to use the phrase 'a constraint which should be specified', because it simply refers to implicit or explicit assumptions about these constraints in application programs.

What makes it difficult to change these constraints? When is it easy to change these constraints? Basically, it is easy to change these constraints if there is no application program which employs them. (We neglect the fact here, that the conversion of an actual database is a sometimes a major problem also.) Changing a constraint is therefore more difficult if more application software has been developed which relies on its existence. In other words, *semantical flexibility in transaction processing is constrained by the investment in software.*

Semantical flexibility in decision support software is a different situation. Often, decision support software can be seen as extracting information from an operational database, performing some computations, and presenting the results to a decision maker. If this is indeed the situation, it is relatively easy to obtain a semantical change: this means the change of only one program. Unfortunately, the situation is often more complicated. In many cases, the result of the computations have to be

stored in the database again, and should therefore satisfy the constraints of the conceptual schema again. This is the case with the MRP I computations, for example. Consequently, the implementation of LRP (see Van Donselaar (1992)) in a standard MRP package is not so easy.

Concluding then, semantical flexibility is in general quite limited, as opposed to syntactical flexibility. However, one comment on this general rule should be mentioned. If software packages can be modified in such a way that a pure extension is realized (and not a modification), then there is considerable room for extensions in a well-designed package. This possibility is in practice quite often used.

6. Pragmatical flexibility

The term 'pragmatics' refers to the usage of a particular modelling language. A conceptual database schema can be interpreted as a modelling language. Such a language does not by itself provide guidelines about many questions, such as:

- what level of detail is needed?
- when are two objects identical?
- what actions are supposed to be taken in particular circumstances?

These questions show that there is quite some modelling freedom left after the semantics have been specified. Of course, if there is more richness and precision in semantics, the freedom in pragmatic is constrained somewhat. However, it is unlikely that this freedom will ever be replaced completely by semantics.

Before going into details, a few examples seem worthwhile again.

When using MRP systems in a particular factory, there is no guideline regarding the number of levels in the bill of material. In similar factories, someone might use 20 levels, and somebody else might use two levels, without any problem from the software point of view. Of course, there are pragmatical guidelines on how to handle these questions, both in theory and in each practical situation. The point is, that software does not impose any constraints.

Another example is the so-called *form-fit-function* rule. This is a pragmatical rule which states, that two different items which have the same form, fit and function in higher level assemblies, can have the same product-identification (= item number). Again this is a rule which is not imposed by software, but by its use.

The requirements for change in operational procedures, such as discussed in Section 2 and in the

appendix, is also largely a requirement for pragmatical flexibility. It is again a different way of using data and data processing, without changing the meaning of entity types, attributes, messages, or computed results.

Pragmatical flexibility is defined as the changeability of the usage of data, in terms of actions which are allowable or compulsory and in terms of the level of detail which is appropriate.

What are the factors by which pragmatical flexibility are constrained?

First, pragmatical flexibility is constrained by the semantical rigidity of the software. For example, if the form-fit-function rule is included in the conceptual schema of the underlying database, then the above-mentioned freedom vanishes.

Second, flexibility in transaction processing support of operational procedures is in many packages quite restricted, nowadays. In the long run, considerable improvement may be expected here. More and more packages are in a process of undertaking the effort to write generalized software for the support of operational procedures, which can be used to configure and generate the appropriate support. An elegant approach is the following: consider operational procedures as a production process in itself, for which again a routing and bill of material can be designed. Then standard 4GL-tools can be used for generating the support software components for operational procedures, and production control software can be used for combining these components into a complete routing.

Finally, the most important constraint for pragmatical flexibility is the organizational investment in existing databases, procedures, and education. Implementing a rule such as the form-fit-function rule leads to a complete database of products which satisfy that rule. Changing such a rule usually requires a huge organizational effort, which is seldom available. For this reason, pragmatical flexibility with respect to basic entity types in the database is in practice a mirage. This means that pragmatics should be given considerable attention at the initial implementation of standard software packages for production and logistics control.

7. Conclusion

In this paper we have first investigated why flexibility is needed. Continuous organizational change appears to be a major driving force for changing information systems. Next, two main types of flexibility requirements were identified, namely changes in operational procedures and changes in decision making. Operational procedures are usually supported by transaction processing software, and information systems' contribution

to decision making is by decision support systems (DSS) and by structured decision systems (SDS). Three types of flexibility requirements were discussed: syntactical, semantical and pragmatical flexibility. The following conclusions were drawn.

- *Syntactical flexibility* is in principle obtainable by available software engineering principles both for changes in operational procedures and for changes in decision making.
- *Semantical flexibility* is difficult to obtain and is constrained by the investment in application software. This holds especially for changes in operational procedures.
- *Pragmatical flexibility* is constrained first of all by semantical rigidity. Furthermore, pragmatical flexibility is in practice constrained by investments in existing data and in organizational learning, and not by application software. This holds especially for changes in decision making.

References

- BERTRAND, J. W. M., and WORTMANN, J. C., 1981, *Production Control and Information Systems for Component Manufacturing Shops* (Elsevier, Amsterdam).
- BERTRAND, J. W. M., and WORTMANN, J. C., 1992, Information systems for production planning and control: developments in perspective. *Production Planning and Control*, 3, 280-289.
- BERTRAND, J. W. M., WORTMANN, J. C., and WIJNGAARD, J., 1990, *Production Control—A Structural and Design-Oriented Approach* (Elsevier, Amsterdam).
- DEN BOER, A. A., 1991, Decision support for material requirements. Planners, working paper, Eindhoven University of Technology, 1991 (submitted to *Production Planning and Control*).
- DATE, C. J., 1989, *Introduction to Database Systems*, 4th ed (Addison-Wesley, Reading, Mass.).
- PELS, H. J., and WORTMANN, J. C., 1990, Modular decomposition of integrated databases. *Production Planning and Control*, 1 (3), 132-146.
- VAN DONSELAAR, K., 1992, The use of MRP and LRP in a stochastic environment. *Production Planning and Control*, 3, 239-246.
- VAN VEEN, E. A., and WORTMANN, J. C., 1992, Generative bill-of-material processing systems. *Production Planning and Control*, 3, 314-326.
- WIJNGAARD, J., and ZIJLSTRA, P., 1992, MRP application in the batch process industry. *Production Planning and Control*, 3, 264-270.

Appendix. An example of a simple change in operational procedures

Let us illustrate the consequences of changes in operational procedures for information systems by a simple

example This example is concerned with changes in the operational procedures of the customer order entry process in a particular company

Example customer order entry

Consider a company which is producing and selling professional measuring equipment. Customer order entry proceeds as follows:

- plan a shipment for all items at the earliest possible date where these items can all be made available to the customer,
- enter the due date requested by the customer for this order;
- plan an earlier shipment for all items which are available before the requested date (and change the earlier plan),
- check the creditability of the customer; if satisfactory, then.
promise a shipment for all items which are available before the requested date, and make reservations for these items (in accordance with the plan),

- promise a second shipment for all other items at the earliest possible date where these items can all be made available to the customer, and make reservations for these items (in accordance with the plan)

Suppose the company decides to change this procedure for customer-order entry as follows.

- check the creditability of the customer; if satisfactory, then:
- enter the due date requested by the customer for this order,
- promise a shipment for all items which are available before the requested date, and make reservations for these items;
- promise a second shipment for all other items at the earliest possible date where these items can all be made available to the customer, and make reservations for these items

This is only a small change in the procedures, but it is a fairly good example of day-to-day requests for changes in information systems.