The evolution of a production planning system:  
A 10-year case study

Kenneth N. McKay a, Gary W. Black b,*

a Department of Management Sciences, University of Waterloo, Waterloo, Ont. N2L 3G1, Canada  
b College of Business, University of Southern Indiana, Evansville, IN 47712, USA

Received 13 April 2006; received in revised form 17 January 2007; accepted 5 February 2007
Available online 23 March 2007

Abstract

This paper describes the evolution of a production planning system (PPS) from a simple work sequence generation tool to a useful, sustained scheduling system. Three stages of evolution are described. In the first stage, a Gantt chart sequencing tool was converted to a scheduler’s information system. This change was driven by the need to support the scheduler’s daily task. The second stage of evolution was caused by an MRP–ERP conversion. The integration and conversion increased overhead and complexity in the job task and hence the tool, including the transformation of the previously integrated dispatching/scheduling task into separate dispatching and scheduling activities. The third stage of evolution has been small continuous improvements driven by management reporting requirements. PPS was developed in 1996 and has been fully operational since January 1997. Two major insights are discussed in this paper: the implications of supporting the scheduling task versus work sequence generation, and the software design requirements for evolutionary change as the software is used in an ever-changing situation.

#2007 Elsevier B.V. All rights reserved.

Keywords: Production planning system; Decision support system; Enterprise resource planning; Materials requirements planning; System evolution

1. Introduction

Production planning systems (PPS) are a specialized form of decision support system (DSS). There are over 100 commercial systems available [1]. The purpose of a PPS is to take manufacturing requirements, match them with a model of the factory or the supply chain and, using various algorithms and technology, and craft a work sequence either automatically or with manual intervention. In this paper, the term scheduling will be reserved for what the human scheduler does as their daily job task—part of which is the creation of detailed work sequences. This differentiation is important since one of the major drivers of the software evolution described in this paper was the explicit recognition and support of the scheduling task, above and beyond the creation of detailed sequences. McKay and Wiers [2,7] describe the production control tasks of planning, scheduling, and dispatching. In this case study the individuals with a title of scheduler performed the tasks of scheduling and dispatching.

If detailed work sequencing is focused upon, the problem appears relatively simple and many sequencing tools have been quickly built with a Gantt chart interface and some form of sequencing engine. The concentration has been on the data representation, database, Gantt chart visualization and manipulation, and on the sophisticated mathematics used to create a recommended sequence. Unfortunately, there are few success stories and there are still many challenges that remain when creating planning and logistics systems that have to work in a real factory [3–6]. While there is not one sole issue or reason for PPS success or failure [7], one potential issue relating to the failure of planning systems is the distinction between generating a sequence and scheduling and the lack of information system support for the latter. This was the first major insight obtained from this longitudinal case study. Another possible reason for system failure is the rigid structure or inability to easily adapt to changing requirements in the factory environment—both information system and organization changes. Learning what aspects of the software should be easily changed was the second major insight.
This case study will describe three stages of evolution are described:

- moving from a basic work sequence generation tool to a scheduler’s information system—driven by the need to support the scheduler’s task;
- moving from a legacy MRP system to ERP (SAP)—driven by changes in the information system infrastructure;
- continuous evolution—driven by management and organizational requirements.

The first evolution occurred during a 4-month period following the initial live test of the software in the Fall of 1996. This evolution saw the code triple in size and the functionality more than double. This basic system was then used for approximately 1.5 years before the ERP migration. When the plant converted to the ERP, the scheduler’s tool doubled in complexity to deal with the added functionality required by the ERP interface and associated task changes. The decision support system continues to be used and has been fully operational and in daily use for over 10 years. The factory using the software has been participating in ongoing research on production control throughout this time period. This relationship has presented a relatively unique research opportunity to research, develop, deploy, and evolve a live decision support system over an extended time horizon.

The first phase of evolution is perhaps of the most interest to researchers who are contemplating the creation of decision support tools for a scheduler or dispatcher task. The lessons and insights gained relate to the understanding of the task and software architecture. The second phase of evolution is more interesting to individuals integrating existing production control tools with legacy or ERP systems. The insights in this context relate more to the changes and impositions created by the larger and more rigid systems. The third phase of evolution provides lessons to both audiences—the need for change and adaptation to the organizational demands as a plant itself evolves.

This paper is organized as follows. Section 2 will present a review of DSS evolutionary literature followed by specific discussion of case studies addressing evolution in the production planning and scheduling field. Section 3 will provide background information for the case study. Section 4 will discuss the system development processes and underlying architecture. Sections 5–7 will discuss in-depth each of the three evolutionary stages. Section 8 will discuss and summarize the results. Section 9 will present some concluding thoughts.

2. Evolution of decision support systems

Research reporting case studies on the evolution of decision support systems is relatively sparse. Evolutionary development in decision support was first hinted at by Meador and Ness [8] and Ness [9] as part of their ‘middle-out’ design. This design was in response to the ‘top-down’ versus ‘bottom-up’ methodology debate at that time. Courbon et al. [10] provided the first general statement of DSS evolutionary development. He argued that development processes are not implemented in a linear or parallel fashion, but rather in continuous action cycles involving significant user interaction. As each cycle is completed, the system gets closer to its final state.

Keen [11] advanced Courbon’s work to develop a model for understanding the dynamics of DSS evolution. The approach was termed ‘adaptive design’ and was based upon cyclic interactions between each pair of three basic elements: the builder/systems analyst, the user and the system. Courbon [12] later described these cycles as sequences of ‘action’ (i.e., when the designer releases a new version and the user works with it) and ‘reflection’ (i.e., feedback to the designer). He concluded that DSS evolution is best conceptualized as a learning process.

In an analysis of system adaptation and evolution, Sprague and Carlson [13] identified four levels of DSS flexibility: the flexibility to solve a problem, the flexibility to modify the DSS to handle different problems, the flexibility to adapt to major changes and the flexibility to evolve with changes in technology. They believed these levels exist in a hierarchy with technology-based evolution at the top. They argued that “DSS must evolve or grow to reach a ‘final’ design because no one can predict or anticipate in advance what is required. The system can never be final; it must change frequently to track changes in the problem, user and environment because these factors are inherently volatile”.

Arnott [14] discussed the nature of the DSS evolutionary process and presented a framework for defining it based on tempo, lineage and etiology. He contended the dominant tempo of DSS evolution was punctuated equilibrium in which logic and operation alternated between periods of being relatively static and periods of rapid change. This finding was in contrast to previous theories citing continuous evolution as the dominant tempo [10–12]. Lineage was discussed in terms of within-application and between-application evolution. Etiology was discussed in terms of cognitive causal factors (e.g., system use, training, interaction with analysts/peers/consultant) versus environmental causal factors (e.g., technology change, personnel change, industry change). By combining the various lineages and etiology, a framework was presented with four major classes of evolution along with the most likely tempo(s) for each class. A large DSS case study pertaining to a large semi-government building project was used to validate and expand the framework. Finally, the framework and case study findings were used to define a research agenda for evolutionary DSS development. This agenda was comprised of a set of 11 research questions within four different categories: evolutionary tempo, etiology, methodology and technology, and the nature and role of people involved.

The above approaches and theories illustrate the importance of the concept of evolutionary development to DSS theory and practice. The notion that DSS’s evolve through an iterative process of system design and usage has been central to the theory since the inception of the field [14]. While existing work on the DSS development describes the process in terms of a final system resulting from an adaptive process of user/analyst learning and system change, the real nature of DSS evolution may be even more complex. To this end, case study papers
discussing actual DSS implementations are crucial to gaining real insight for actual implementations.

No case studies have been found in the literature that describe the long-term evolution of production planning systems. There have been several studies that discussed design and usage factors for planning DSSs. In one, McKay and Buzacott [15] presented case studies of two different job shops—a high volume/low mix shop and a low volume/high mix shop. In the high volume/low mix shop, the human scheduler did an excellent job without the aid of any DSS. In the low volume/high mix shop, the human scheduler required a PPS to manage the situation. Although analytical and algorithmic aids had limited benefit in that type of shop, the appropriate use of computer technology did assist in addressing information overload, cue filtering and scheduler problem solving.

In another study, McKay and Wiers [16] presented the design of a PPS in a focused factory. In this factory, one person was responsible for planning, scheduling and dispatching. This “integrated planner” needed a seamless system with functionality ranging from performing daily dispatching to generating a 5-year plan. The paper presented the design of a DSS used to assist this planner. Specifically, DSS requirements and functions were specified and an empirical case study was used to illustrate the type of tasks that the planner performed and how the seamless DSS supported those activities.

There have also been few detailed, carefully researched studies of actual APS implementations—studies of the issues faced by the implementers. In the one such study found in the literature, Zoryk-Schalla [17] describes the modeling challenges at the planning level when using a tool such as i2 and relates the theories of Schneeweiss [18] and Bertrand et al. [19] to the problem. The research focused on matching the solution to the problem and highlighted the need for a thorough understanding of the factory production control situation prior to implementation. This research was limited in scope and focused on the higher levels of the control problem and not on the operational levels of final sequencing and dispatching. The research was also not concerned about how to design an operating tool to better model the problem, nor about how the tool would be used in final deployment.

In summary, although the McKay and Buzacott paper [15] and McKay and Wiers paper [16] presented valuable case studies, the primary focus was on describing the DSS components, features and usage, as well as human scheduler behavior and interaction with the DSS. The evolution and long-term use was not discussed.

3. Background and summary

The shop described in this case study is a press shop consisting of metal stamping and forming processes. There are roughly 65 major resource groupings possible and over 150 individual presses. Resource groupings are not static and presses can be coupled or decoupled from each other as required. Over 1000 dies exist and roughly 500 different parts are produced. The parts range in size from inches to over 15 ft. and from 1 stamping operation to over 10. Batch sizes range from dozens to thousands depending on part usage. Some dies accommodate only one part, others accommodate multiples of the same part, and still others accommodate multiples of different parts. One dozen blanking presses exist to feed the forming presses. In some ways, the job shop resembles a flexible or re-configurable flow shop since various machines are bolted together to create a line without intermediate inventory between operations. This physical reconfiguration of the machines implies that the number of parts is not scheduled per se, but groups of resources are assigned to the batch.

While there does exist some state-of-the-art technology, the sheer number of presses and machines have precluded massive and widespread investment. Thus, long setups have been prevalent (e.g., 1/2–1 shift) and batch sizes are usually large enough for the job to run two to three shifts. A cyclic schedule is attempted where the majority of parts are run in a 2-week cycle.

A preliminary study lasting 6 months was performed in early 1996 to document the scheduling requirements and determine the potential fit of commercial software packages to the problem. These commercial tools were ultimately ruled out for the following reasons:

(i) The crew-based orientation of the shop—a larger crew had general ‘ownership’ in an area, but the number of workers needed to build a part varied from part to part and the timing of allocation and freeing of crews was a major activity (e.g., completion of one part might free up 12 workers, but 14 workers are needed for the next part and it is impossible to build that part with just 12 workers).

(ii) Splitting of die setting and running of parts—setup crews are used and they can setup the presses independent of the running crew and the machines might be idle for a period of time between the setup and actual production.

(iii) Shift-oriented planning—from a personnel viewpoint, the beginning and end of shift timing was important, as was the ability to pause a job, move the crew to another set of machines, and then eventually resume production on the original resources.

(iv) The manner in which the current floor situation was described and analyzed—the shop, plant, and corporation used unique measurement metrics and terminology.

(v) Mathematical algorithms were not considered a priority since demand was highly unstable due to being pulled just-in-time from a high-variance assembly area. At one point, the standard deviation of the daily demand was equal to the mean demand. While it may have been possible to negotiate custom development with a commercial PPS vendor, this option was considered lengthy, high risk and high cost. The plant wanted a system to be functional within 6 months, by the end of 1996, and did not want to commit to a large investment up front. A similar plant within the corporation with a variety of similar customization requirements had been attempting for over a year to implement a leading system with little success. Thus, the
decision was made to create a simple sequencing tool that would replace the physical Gantt chart-style board used by the schedulers. The ‘computerization’ of the physical board would have a number of benefits such as saving time, permitting daily sequencing and providing more accurate sequences. This basic functionality was considered sufficient by all involved and was typical of the typical functions found in the leading tools. In June 1996, a prototype system was built in MS Excel using a simple database of jobs to establish the necessary scheduling logic. The purpose was to verify the data fields required, the data interpretation and the logic required to determine realistic and feasible start–finish times. After 1 month, these objectives had been met and the second phase of the project was authorized. The second phase was to build a Gantt chart system using Excel and Visual Basic for Applications (VBA) to replace the physical board. Excel and VBA were chosen for several reasons including the ability to perform rapid prototyping, familiarity of the schedulers with Excel, and the infrastructure of the worksheet format. Accordingly, the initial pilot software was created by the end of August 1996, training was scheduled for September and parallel running was scheduled for October–November. The first functional version (Pilot version) took 2 months to construct with a total of four man–months by a two person team (one developer, one user).

Thirteen software characteristics are summarized in Table 1 and have been enumerated over four milestone versions. Comparing the Pilot to Live version captures the first evolution driven by the scheduler’s task. The Live to Post-SAP captures the evolution driven by information technology, and the Post-SAP to Mature captures changes associated with the ongoing management and organizational changes. In essence, the categories illustrate what remained relatively stable between versions and where development effort took place. For example, the number of menus and functions on the user interface double from the Pilot to Live version and then remained stable. The sequencing logic was stable between the first two versions and then increased with the integration with SAP. This illustrates that the sequencing logic was not impacted by the scheduling task evolution (Pilot to Live), but was impacted by the information technology evolution (Legacy to SAP).

By the end of September 1996, it was believed that the first Pilot version was complete and robust enough for parallel use and comparisons. The functionality represented by the 6542 lines of code was considered sufficient for interactive Gantt chart presentation and manipulation, for importing of daily MRP requirements, for sequencing jobs on the resources, and for preparing reports describing the sequence. However, due to circumstances in the plant, the October parallel test turned into a ‘go-live’ situation resulting in accelerated training and live usage in a stressed situation. This effort was not successful, use of the Pilot software stopped, and a second major development undertaken to upgrade the software for live use in January 1997. After 2 months of development (again by the same two-person team), the software had the characteristics shown in the ‘Live (1997)’ column in Table 1.

In this second version (1997), the code tripled in size, the number of menu functions and dialogs doubled, the number of reports increased from 1 to 14 and the number of fields in the schedule database increased by 50%. The reasons for these changes are discussed in the following sections. With some minor changes, this version remained operational until July 1998 at which time the factory converted to the SAP ERP system and a different task structure within the Production Control department was used. A major software update was performed in May–June 1998 to prepare the system for the SAP integration and the new tasks. As can be seen in the third column, the move to the SAP ERP system and task re-design resulted in roughly double the code, 50% more dialogs, 50% more reports, 50% more fields in the part and scheduling databases and more types of data being imported into the system. The fourth column documents the characteristics for the mature, 2006 version. While the count of certain characteristics remained roughly the same, some interesting changes took place which will be discussed in Section 7. Section 4 will discuss the development process chosen and certain, key aspects of the underlying architecture. The subsequent section will discuss in detail the evolution of the production planning system (PPS) from a simple Gantt chart

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>6542</td>
<td>18,728</td>
<td>33,574</td>
<td>31,793</td>
</tr>
<tr>
<td>Size of sequencing module</td>
<td>1100</td>
<td>1,100</td>
<td>1,493</td>
<td>1,924</td>
</tr>
<tr>
<td>Menus</td>
<td>6</td>
<td>10</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Functions on menus</td>
<td>63</td>
<td>124</td>
<td>110 scheduling, 115 dispatching (93 common)</td>
<td>91</td>
</tr>
<tr>
<td>Dialogs</td>
<td>16</td>
<td>30</td>
<td>39</td>
<td>28</td>
</tr>
<tr>
<td>Other messages</td>
<td>97</td>
<td>420</td>
<td>460</td>
<td>422</td>
</tr>
<tr>
<td>Reports</td>
<td>1</td>
<td>14</td>
<td>22</td>
<td>15</td>
</tr>
<tr>
<td>Files imported</td>
<td>1</td>
<td>6</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Fields in part file</td>
<td>30</td>
<td>30</td>
<td>44</td>
<td>55</td>
</tr>
<tr>
<td>Fields in tool file</td>
<td>7</td>
<td>36</td>
<td>36</td>
<td>37</td>
</tr>
<tr>
<td>Fields in schedule database</td>
<td>54</td>
<td>79</td>
<td>105</td>
<td>101</td>
</tr>
<tr>
<td>Change functions under edit</td>
<td>3</td>
<td>7</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>Finding and searching</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 1
sequencer (i.e., physical board) to a scheduler’s information system.

4. Development processes and underlying architecture

The choice of a development strategy and architecture will undoubtedly affect how a system will respond to changes and evolve over time. If the choice is not appropriate, then it is possible to claim that changes to the software were not driven by the environment or factory situation per se, but by a faulty process—the requirements analysis was not done properly and requirements anticipated, and the architecture was not appropriate and supposedly minor options and changes escalated into larger efforts.

4.1. Requirements analysis

The starting point for features and functions to provide the scheduler/dispatcher for the initial study at the factory was the set of features and functions commonly found in commercial planning and scheduling tools. Since the tool was to respond to ‘pull’ from a MRP system, the scheduling tool had to support the import of data from the legacy system representing demand, and inventory levels. The system had to support the basic terminology used by the shop and support the operational routing and descriptions of the materials and tasks. The primary user interface was to be a Gantt chart editor and by 1996, the features needed in such a tool were reasonably established—the edit, cut, paste, copy, click for more data, etc. The shop had been using a wall-sized Gantt chart for the sequencing of work and this provided the visual clues as to how the users ‘saw’ their work and represented it. As noted above, approximately 6 months were spent gathering specific requirements for the sequencing tool and understanding the operations of the press shop. The DSS project also started after a year long simulation project completed elsewhere in the plant, so the basic plant shop. The DSS project also started after a year long simulation process flows, product characteristics, and business processes were also known.

The actual process used to study the scheduling situation for the 6 months and derive insights into the actual requirements was ethnographic [20]. The focus was on the day-to-day level of sequencing the lines and dispatching work. The planning tasks were not included and thus, there were no issues or requirements for aggregate or hierarchical modeling as in Zoryk-Schalla [17]. Similar to Zoryk-Schalla’s work, the initial analysis focused on the actual users of the software and not on the indirect users of the software’s output, nor on the micro-operations performed by the user in crafting the output.

In terms of the basic data modeling and operational modeling, the PPS model and software has been largely unchanged since the first prototype system of August 1996. The initial system had the core set of functionality found in commercially available systems and the basic representation of material, operations, Gantt chart, Gantt chart operations, and schedule reports have been stable. This stability in the basic problem support over 10 years suggests that this part of the requirements was adequate and that any subsequent changes were not associated with the basic manufacturing model. As will be discussed in Section 5, the types of changing requirements and evolutionary demands were not related to the fundamental scheduling problem, nor the types of issues suggested by Zoryk-Schalla.

4.2. Development strategy

Traditional software engineering development practices have been “plan driven” and characterized by predictable, repeatable processes [21]. Users tell the developer once and for all exactly what they want. Programmers then design the system to deliver those features using conventional coding and testing processes. In theory, this should work fine. However, in practice, users did not always tell developers exactly what they wanted; they changed their minds. In addition, programmers misjudged their progress. As a result, cost overruns and schedule delays often followed. This problem has been exasperated as more companies seek to develop competitive advantage through timely deployment of Internet-based solutions. Developers have been under increasing pressure to produce software more quickly [22,23]. Consequently, development processes needed to become more flexible to respond to dynamic changes due the inherent oversights and complications that resulted.

“Agile processes” are intended to support quick and early development of working code. The focus is on simplicity and speed. While no clear agreement has been reached on where exactly the boundary lies between agile and plan-driven processes [24], there are some basic distinguishing characteristics [25,26]:

- Agile processes focus on individuals and interactions instead of processes and tools.
- Agile processes focus on working software over comprehensive documentation.
- Agile processes focus on customer collaboration over contract negotiation.
- Agile processed focus on responding to change over following a plan.

The agile movement emphasizes the relationship and communality of software developers (i.e., the human role), as opposed to strict institutional processes and development tools. The vital objective of the software team is to continuously turn out tested and working software. New releases are produced at frequent intervals and developers strive to keep the code simple and straightforward. The relationship between developers and clients is given preference over strict contracts, although the importance of well-drafted contracts does grow at the same pace of the size of the software project. The negotiation process itself is seen as a means of achieving and maintaining a viable relationship. From a business standpoint, agile development focuses on delivering business value immediately as the project commences, thus reducing the risk of non-fulfillment of the contract. Finally, the development group, comprised of programmers and customer representatives,
are prepared and authorized to make changes to address emerging needs during the development process [24].

Agile versus plan-driven approaches are often viewed as polar opposite ends of the software development methodology spectrum. Agile methods are viewed as totally adaptive, while plan-driven methods are viewed as totally controlled. However, some researchers have claimed that synthesizing the two approaches can provide developers with a comprehensive spectrum of tools and options that can be preferable in some circumstances [27]. For instance, ‘over-responding’ to change has been cited as the source of many software disasters, such as the $3 billion cost overrun of the U.S. Federal Aviation Administration’s Advanced Automation System for national air traffic control.

Agile development has been widely documented as working as working well for small (<10 developers) co-located teams that are facing unpredictable or rapidly changing requirements [21]. However, its applicability in the following scenarios has been questioned and criticized for use in situations such as:

1. Large-scale development efforts (>20 developers).
2. Distributed development efforts (i.e., non-co-located teams).

Reasons for these criticisms relate to the reduced amount of documentation, the compressed software design timeframe, the trend toward globally distributed development environments, the unproven quality control mechanisms and the cultural change that is sometimes required to provide the development team with true empowerment to identify and make major changes to the software and contract. One paper has stated, “It is also clear that companies that develop long-lasting, large complex systems may not be able to use agile processes in their current form” [28]. However, much of this criticism has been refuted by agile practitioners as simply being misunderstandings about the agile development process [29].

Thus far, we have discussed the agile development approach in general. However, the agile approach contains a number of different methods such as extreme programming, Scrum, the Crystal family of methodologies and adaptive software development, among others. Extreme programming (XP) is arguably the most popular agile methodology and has evolved from the problems caused by the long development cycles of traditional development methods [21]. It started as 'simply an opportunity to get the job done' [30] using practices that had been found effective in software development during the preceding decades. After a number of successful trials in practice, XP was theorized upon the key principles and practices used [31]. Even though the individual pieces of XP are not new, XP has integrated them in such a way as to form a new methodology. The term ‘extreme’ comes from taking these commonsense principles and practices to extreme levels [31].

The XP lifecycle consists of six phases: exploration, planning, iterations to release, productionizing, maintenance and death [31]. In the exploration phase, customers write the ‘story cards’ that they wish to be included in the first software release. Each story card describes a feature to be added. Concurrently, development team members familiarize themselves with the technology and tools to be used. This phase can take from a few weeks to a few months. The planning phase sets the priority order for the stories (i.e., features) and an agreement of the contents of the first release. The time span until the first release normally does not exceed 2 months. This planning phase itself normally takes just a few days. The iterations to release phase breaks down the schedule from the Planning phase into a number of iterations, each of which will take 1–4 weeks to implement. The first iteration creates the architecture for the entire system. Functional tests and stories to be included are specified by the customer. At the end of the last iteration of this phase, the system is ready for production. The productionizing phase requires extra testing and checking of the system before it is released to the customer. During this phase, new changes can still be identified and incorporated. Postponed ideas are documented for implementation at a later phase. The maintenance phase is reached after the first release is provided to the customer. The development rate may decelerate as the customer uses the system. Customer support tasks are provided. New iterations are produced as the customer identifies new stories to be added. New people may be incorporated into the team. Finally, the death phase arrives when the customer no longer has any stories to add and the system meets his/her needs in all other respects (i.e., performance, reliability). At this time, necessary documentation is finally written since no more architecture, design or code changes are required. Death may also occur if the system is not delivering the desired outcomes, or if it becomes too expensive for further development.

For the PPS development, a lifecycle following the XP model was generally followed. The exploration and planning phase took place in the first half of 1996 and by using the ethnographic methods, the equivalent of stories or scenarios were captured. The basic developments were then phased in short 1–4–week cycles with the developer imbedded with the final user. The first phases were performed in 1996 with the software going into production January 1997; entering the maintenance phase. Suitable system documentation was also created early 1997, explaining the architecture, data model, and so forth. The maintenance phase is still active and the death phase has not been entered. It is unlikely that a software tool that has to respond to the environment will ever be static in the sense of no further development. For software that needs to be periodically adjusted, the death phase will result from a management decision to terminate the use of the software or such a major change in the environment occurs that the software cannot be modified and a new solution is required.

The XP approach and concept requires certain domain knowledge and expertise and cannot be used as an excuse for substandard thinking and analysis. In this case, the factory context, knowledge of features and functions commonly found in commercial packages, and an experienced developer (two decades of tool smith, software development experience) were important for the process to work. The XP approach requires that you have such experience in the development team and that
the skill level is above average. In a 12-month period, with approximately 12 person–months of effort, the project went from—‘have a look at what the schedulers are doing’ to a scheduler’s information system, live in production. This period included the first evolution described in Section 5—the evolution from a basic Gantt chart scheduling tool to the scheduler’s information system.

4.3. System architecture

The PPS system has the normally expected data encapsulation and data independence expected of modern systems using object-oriented approaches. The major objects on the user input side include part and tool information. The part information encompasses data needed for sequencing, some specific material requirements planning for steel and blanks, and supporting data used on reports. The tool information provides details about the dies, die series, and press sequences that need to be coupled together. The other major data objects are downloaded from the main system daily—the demand and inventory levels. Within the software, functions and logic for each data object are isolated and are accessed by appropriate methods. The database design and support is considered to be standard. The object classes exist as worksheets in Excel and this makes data inspection and manipulation during development very easy—everything is visible in cells and with the interactive debugging of VBA, objects can be directly interacted with. It is also very easy to add columns and fields as needed to object definitions while prototyping.

The internal logic also has the major groupings that would be expected in any sequencing tool—importing and exporting of data, report generation, user interface, and sequence generation. Fig. 1 illustrates the basic architecture.

What is perhaps different in the PPS architecture is how the system actually works at the detailed level—within the objects and logic. There are two aspects of the system to note from an architectural perspective.

First, there is a task or engine philosophy using lists of entities. There are standard components that can be used and reused to create lists of entities—lists of jobs, lists of resources, lists of jobs associated with a resource—this is similar to the capability supported by relational database systems such as SQL or languages such as LISP. The lists can be generated as sub-lists for logical groupings in reports and data analysis (e.g., isolated MRP logic for the steel and blank analysis). The concept of lists of objects and the processing of lists is dominant in a repetitive task such as scheduling and a list-oriented paradigm for execution appears to be a reasonable choice.

Second, the sequencing engine has several important characteristics. The most important is that the engine mimics the actual shop and how the operations are analyzed and sequenced. The same types of issues are thought through in the same sequence and with the same emphasis. This overall matching of the sequencing logic and sequence construction is at the physical reality level. There are internal differences and optimal concepts since software can do things a human cannot, but the structure of the sequencing has fidelity with the real shop. On a technical level, as with all of the functions, the engine is driven by various list managers, always resulting in the lowest level being invoked—sequence an operation. Since the jobs are being pulled from an assembly area in a just-in-time fashion, the jobs are layered into the sequence using expected pull dates and resource conflicts identified. Instead of a resource view using a traditional dispatch rule approach, the jobs are the primary decision objects and they seek appropriate holes in the schedule and if necessary push other work aside.

The actual sequencing logic is used twice per job since setups can be sequenced separately (they use different crews) and the machine can sit idle between setup and the actual running cycle. In essence, each operation was modeled as two operations with each having all of the possible constraints and options. For each instance, the job and operations are first analyzed and constraints identified. The ‘current resource’, ‘current job’, ‘current earliest starting time’ and so forth are established as are many other ‘current’ variables and statuses. This creates a view of the world for all internal sequencing routines to use and manipulate without necessarily modifying the original or final values. The setup and actual operation are then sequentially fed into iterative logic that actually decides the sequence, start and end times. The iterative logic works in
what might be a consultative fashion with the job estimating how much time it will need and then the selected resource estimating how close to the earliest start time that the job can start and then when the job will be finished—taking into account shifts, weekends, etc.

The logic is decomposed along these lines—similar to how a human would think through the process of creating a layered schedule in a job shop being pulled by assembly lines. For each job, there is a preferred set of resources and while it is possible to place jobs in alternative locations, it is not the default selection; thus there are no complications arising from the selection of equally capable resources. The operations in a job are also executed in a virtual flow line fashion and the resources are literally bolted together into a ‘line’; the work associated with a batch is performed without other work inserting itself. The ‘line’ has to be reset in its entirety before the next job can be processed with its operations being executed in a flow fashion without intermediate work-in-process.

The two key design decisions made—the list and task engine design, and the sequencing engine design greatly supported the long-term use of the software. The list and task engine design allowed for many features and functions to be quickly built from the tools built into the system, combining various pieces together in new combinations when desired. It has also been easy to write new tasks and list generators to match the user requirements. The sequencing engine design has also proven itself solid over the life of the software. The basic characteristics of the physical reality were incorporated into the engine. It is not a general purpose engine suitable for other plants. Since the engine was custom designed and built, it was not necessary to live with options and settings not required, nor suitable. The engine captured the essence of the job shop, no more, no less and this meant that changes easily made in the real, physical shop were easily made in the software and the other jobs adjusted accordingly. The pilot system could do all of this and even more. For example, it would require a new magnet to be placed on the board when/where the jobs were expected to run. The board was updated weekly. Occasionally, when a mid-week update was necessary, the scheduler would move the magnets by pushing some out, pulling some in, changing the distance between some, moving them from one resource to another, etc. If a new requirement arose, a new magnet would be placed on the board and the other jobs adjusted accordingly. The pilot system could do all of this and even more. For example, it would automatically determine the length of jobs and compensate for weekends and different shift patterns. These calculations used to be done mentally. Each day the scheduler received a new ‘daily net requirements’ list which was inches thick and would scan this list looking for deviations from the prior day. Work beyond the next 2 days was largely ignored. The scheduler focused on the next 24 h. The pilot system imported this information and automatically layered the jobs on the schedule using finite scheduling logic based on due date. The benefits were obvious: obtain a daily, rolling view of the work based on finite loading, perform automatic calculations of manpower requirements and project die setting needs, eliminate the need to manually process the daily requirements list, automatically push work ahead as needed and automatically compute elapsed time estimates for each job.

So why was the pilot system not good enough? After all, it did what it was designed to do and even more. It had all of the basic features and functions of commercially available planning and scheduling tools—the ones needed to represent

5. Evolution from Gantt chart sequencer to a scheduler’s information system

There was one key observation about the evolution of the PPS spanning the time from the first pilot test to when the software was actually used in a live ongoing fashion—the pilot system was sufficient for creating a sequence, but it was not sufficient for actually executing the sequence, assisting the scheduler, or supporting the scheduler throughout the day when the scheduler was actually scheduling. The types of changes needed to go live in January 1997 were:

(i) additional information used to make manual decisions was added to the database;
(ii) information associated with the sequencing decisions and used by others was added;
(iii) reports for different audiences were generated—management and floor personnel;
(iv) reports for different time horizons were now desired—providing forward visibility for next several days;
(v) the planning and sequencing horizons and the associated visibility was increased;
(vi) outside activities such as tooling information was integrated to reduce data duplication and copying;
(vii) the sequencing entities were decomposed to a greater extent—e.g., die setting versus running parts;
(viii) the ability to model difficult aspects was extended—such as ‘run together’ when two independent demand parts must be run together at that same time on the same machine (e.g., die with two different parts in it);
(ix) more options for relaxing constraints and altering data were added;
(x) functions to better mimic the physical board (e.g., move magnets, leave gaps);

Could any of these enhancements have been anticipated? Should they have been? Was the Pilot a substandard system? Consider the original intent of the system. The system was supposed to ‘replace’ the manual (physical) scheduling board and allow the scheduler to update it daily. That was all the system was supposed to do. The physical board had a row for each resource, a column for each shift/day over a 2-week period, magnets containing part names to indicate what was currently running, and magnets scattered around to indicate when/where the jobs were expected to run. The board was updated weekly. Occasionally, when a mid-week update was necessary, the scheduler would move the magnets by pushing some out, pulling some in, changing the distance between some, moving them from one resource to another, etc. If a new requirement arose, a new magnet would be placed on the board and the other jobs adjusted accordingly. The pilot system could do all of this and even more. For example, it would automatically determine the length of jobs and compensate for weekends and different shift patterns. These calculations used to be done mentally. Each day the scheduler received a new ‘daily net requirements’ list which was inches thick and would scan this list looking for deviations from the prior day. Work beyond the next 2 days was largely ignored. The scheduler focused on the next 24 h. The pilot system imported this information and automatically layered the jobs on the schedule using finite scheduling logic based on due date. The benefits were obvious: obtain a daily, rolling view of the work based on finite loading, perform automatic calculations of manpower requirements and project die setting needs, eliminate the need to manually process the daily requirements list, automatically push work ahead as needed and automatically compute elapsed time estimates for each job.

So why was the pilot system not good enough? After all, it did what it was designed to do and even more. It had all of the basic features and functions of commercially available planning and scheduling tools—the ones needed to represent
the shop and schedule it. The answer is that the new system changed the status quo in terms of the scheduling task. It changed what was expected of the scheduler in terms of the amount of sequencing and planning to be done. It changed when the work was expected to be done. It changed the degree of accuracy expected in the task. For example:

1. The scheduler was expected to have a new ‘plan’ done by 7:00 a.m. What did it mean to create a new plan? The old sequence from the day before (except for what was currently running) was scrapped and the new MRP requirements were reacted to. This plan regeneration was necessary since MRP logic was pulling the job shop in a just-in-time fashion from the assembly area and an internal study showed that 75% of the plan was altered each day as different parts would overtake others in due date requirements, often due to common parts being used in different assemblies. In the old situation, the scheduler would do a complete plan once a week and the organization would have a planning meeting. In the new situation, new plans were expected daily and mini-planning meetings would be held each day.

2. Previously, the scheduler was only responsible to plan for the current shift by 7:00 a.m. Between 7:00 and 8:00 a.m., the scheduler would examine the daily net, see what was happening in the next day and prepare notes for a meeting. Time was then taken between 8:00 a.m. and 10:00 a.m. to craft reports. The scheduler was expected to have everything sorted out in time for the 7:00 a.m. meeting and this included the reports. The timing of the scheduler’s deliverables was totally changed by the expectations associated with an electronic scheduling tool.

3. The expectation of the plan also changed from rough estimates of ‘day level’ to that of rough position within a shift. Management did not expect to-the-minute or to-the-hour accuracy, but they wanted to improve from 1 day to half-shift accuracy. It was reasonable to expect that resource requirements for die setting and certain parts would be known to a greater level of accuracy. Management also expected that manpower and physical resource loading would be accurately smoothed and balanced for the next week or two. This had never been expected of the scheduler before.

These three changes – quantity, timing, and accuracy – significantly impacted the scheduling task. In 1996, the manufacturing situation was supported by a computer system that was an MRP-I home-grown legacy system. For incentive tracking and other reasons, shop floor data was batched for each shift. When the scheduler arrived at work each morning at 5:45 a.m., he would determine what did or did not happen since he left on the previous afternoon and what the current shop floor status was. This was his starting point.

Although the new PPS system did the basic Gantt chart preparation, it could not do everything automatically and required the scheduler to ‘tweak’ it. The scheduler was required to update the system with information about which parts were running and where, how many parts remained to be built, where to allocate the work crews, etc. The system simply took a rough-cut plan and prepared a basic Gantt chart. The scheduler had to do the ‘creative’ problem solving typical in a job shop such as splitting batches, reducing run sizes and using alternate resources to finalize the sequence. To generate the daily dispatch report, he had to analyze the entire shop. Previously, he only was responsible for one major task at a time and he performed fragmented problem solving with partial knowledge to distribute the sequencing task over the time frame. He did not have the time to prepare his other 7:00 a.m. deliverables and to prepare a daily update. The expectations were not humanly possible.

Upon analysis, there was nothing significantly wrong with the basic PPS resource model, nor the sequencing capability with the shifts and calendars. The scheduler was able to setup his work and make the computer screen mimic the physical wall board. It also saved time and produced a more consistent and accurate plan. If the output was acceptable later in the day, the problems would have been minimized. However, since the shop was being pulled with a short JIT lead time, this was not always possible.

Again, could this have been anticipated? It was in a sense. It was anticipated that the real usage would not be known and the real issues would not arise until the senior scheduler tried to use it live. This was in the spirit of agile development. The junior scheduler sat with the programmer during the development and this had provided confidence in the tool’s usefulness. So it was no surprise when the senior scheduler initially proclaimed the very first version as being very usable and “awesome”. However, the gap between the Pilot system at roughly 7000 lines of code and the useful Live system 4 months later at over 18,000 lines of code needs to be challenged and probed. What new functionality added to the live system could have been anticipated in the pilot system? It is not reasonable to expect all functionality could have been anticipated, but could more have been anticipated? The following will discuss these questions in detail.

Recalling the 10 categories of changes or enhancements added to the “go live” version, the following four categories could have possibly been anticipated if the more traditional ‘plan’ style of development had been followed:

- decomposing the problem to a greater extent—die setting versus running parts;
- modeling difficult aspects—such as ‘run together’ when two independent demand parts must be run together at that same time on the same machine (e.g., die with two different parts in it);
- more options for relaxing constraints and altering data;
- functions to better mimic the physical board (e.g., move magnets, leave gaps).

In terms of requirements and problem specification, the first two categories relate to modeling issues and the last two categories relate to usage (operational) issues.

The modeling issues relate to variants and exceptions to the norm. When the project started, the die setting crews and job
crews were working the same shifts. When the Pilot software was deployed, the die setting crews were not working the same shifts. In rapid prototyping, it is not typical to implement flexibility and multiple options unless they are actually being used—little effort is expended upon conceptual ‘what-ifs’ and ‘maybes’. Hence, the crew modeling, even if it was known that they could potentially run on separate shift schedules, would not have been implemented. It was known that the setup crews could run independently and that had been incorporated, but it was assumed that the shop would run on the same shift patterns. It turned out that by the Fall of 1996, the setup crews were scheduled three shifts a day and the actual lines two shifts a day.

The so-called ‘run together’ parts were another modeling problem. In that case, the software has to pick up the earliest demand of the pieces and pull all associated work ahead accordingly. The build quantity is also adjusted to the highest quantity (each part could have different due dates and quantities) to ensure the highest demand part triggers the next cycle. This was a case of tacit knowledge whereby the scheduler subconsciously adjusted the schedule and orders. It is possible that this modeling concept could have been identified in advance by more extensive interviewing and data analysis. The trade-off would be the effort (time and resources) required to discover the requirement versus the effort required to put the requirement into the system after the fact. With decoupled object-oriented designs, the impact of missed requirements can be minimized subject to the type of requirement and how fundamental the concept is. For example, a major concept relating to part formation or flow can still result in massive changes to the best object-oriented implementation; smaller or less fundamental concepts can usually be accommodated without much penalty. The ‘run together’ oversight fell into the latter category. Before live deployment, the Pilot system was known to generate schedules that passed scrutiny. The developer and junior scheduler felt that while some things might be missed, the basic problem had been captured in terms of job parameters, work options, and schedule generation. Any missed requirement was assumed to be minor. Although these two modeling deficiencies could have been avoided, it is not clear that the cost and effort would have been justified. Each of these problems was addressed later with less than 2 days of programming.

The two operational deficiencies relate to the relaxing of constraints/altering data and the movement and manipulation of jobs on the Gantt chart. Operational modeling and simulating live usage is a topic area specifically limited by the rapid prototyping approach. Simulating ‘what-if’ situations and full case analysis are the ideal and preferred development approaches when budget and time are not issues. Special test cases are constructed and the user interaction studied to identify the difficulties. An alternative to this is a parallel test mode whereby a user of the pilot system attempts to replicate what the non-assisted user is doing. This was originally planned for the PPS tool. Recall that September 1996 was intended to be training and refinement of the pilot tool with a parallel test to be conducted in October. If this plan had been followed, some of the operational difficulties would likely have come to light before the tool was deployed in a live fashion. However, circumstances dictated that the parallel test was scrapped and October turned into a live test. Although this was certainly not ideal, the findings were again in line with the rapid prototyping and agile approach. In other words, one does not develop features or options beyond the foundation set until they are needed. In that sense, the ability to alter information was suspected and some ability was built into the pilot. Nothing was assumed to be fixed in the pilot and all of the typical scheduling data was table driven—setup times, due dates, earliest start times, processing times, run quantities, whether a setup was needed, which resources could be assigned, etc. Changes to these requirements discovered during the live test in the Fall of 2006 were readily incorporated into the system and took less than 2 days to implement. The cost incurred by delaying their development until it was clear what was needed to be changed was not significant.

The operational changes made to job movement on the chart were of a different nature. If a physical wall board (i.e., Gantt chart) is envisioned, it is possible to remove a magnet and nothing else changes. It is also possible move an entire group of magnets simultaneously to the left (i.e., global left shift) to fill a schedule gap. A magnet can also be moved to the right in order to insert a new job in between. It turned out that the scheduler wanted the user interface to be that simple. He wanted the computer screen and controls to require no more effort than the magnetic board. In some cases, this was not possible; in many other cases, this was feasible. Consider a child pushing magnets around on a refrigerator door—there are few rules and many sweeping actions are possible that achieve great change. Change was needed to make the screen ‘more like his board’. This was amplified by the fact that daily planning was now required. In the prior weekly mode, scheduling was not done under pressure in a time-constrained fashion. The daily situation would have been hard to simulate in advance because the scheduler would not have been operating under the same level of urgency in an artificial situation as he would have been in a real stressed situation. Although these requirements had been overlooked, the implementation effort was minimal with 1 or 2 days being required to add the functionality.

In summary, these four types of changes and enhancements can be described as follows:

(a) Decompose the problem more—die setting versus running the job:
- even if known about, it would not have been implemented anyway since the situation did not exist at the time of development.

(b) Model some difficult aspects such as ‘run together’:
- this requirement could have been identified using a more thorough analysis, but the impact was not significant due to software architecture.

(c) More options for relaxing and altering data and constraints:
- concepts could have been discussed as a result of a more in-depth analysis, but they would not have been implemented anyway until they were actually required.
(d) Functions to better mimic what you can do with magnets—slide in, leave hole:

- a full dynamic simulation would have been needed; this was not feasible and the changes were easily incorporated in the design once the function was noted.

In summary, 4 of the 10 changes could have been anticipated to some extent had a traditional approach been taken instead of the agile approach. Given the rapid development of the complete system (6 months from the decision to actually build a system), and the fact that the functions were readily added, the traditional method with its slow, careful approach does not appear warranted. As demonstrated, these are the types of changes and functionality readily addressed by rapid prototype techniques. Compromises are made and risks are taken, but the software design is assumed to be robust enough to address ‘minor’ changes in an effective and efficient manner.

The remaining six change categories are interrelated and represent the majority of the changes and enhancements. They are the types of changes that neither traditional nor agile approach would likely anticipate. They are as follows:

- additional information used to make manual decisions was added to the database;
- information associated with the scheduling decisions and used by others was added;
- reports for different audiences were generated—management and floor personnel;
- reports for different time horizons were now desired—providing forward visibility for next several days;
- the planning and scheduling horizons and the associated visibility was increased;
- outside activities such as tooling information was integrated to reduce data duplication and copying.

These six changes were related to the increase in scheduling expectations, the impact on the scheduling task and the increased expectations about the use of the schedule by other functions (e.g., proactive anticipation rather than reactive fire fighting). They were secondary issues when the scheduling study was initially done and a preliminary discussion and analysis did not indicate that there would be a problem. However, the more that people were held accountable to use the output of the scheduling system, more questions and demands were placed on the scheduler that required his time. The plan was expected to be more accurate, as were status information and processing time estimates. During the prior weekly activity, the scheduler balanced his time between reduced efforts on other tasks and more proactive efforts. This fact was not recognized beforehand and the scheduler was optimistic that he could still get it all done. However, once the activity was performed daily, he was unable to move other activities to other days to balance his workload.

To be useful and usable, the software had to address almost all other tasks the scheduler performed between 5:45 a.m. and 10:00 a.m., not just sequence building. The production planning system (PPS) had become a scheduler’s information system. It managed the majority of his informational tasks while automating as many as possible, eliminating duplication of effort where feasible, and integrating decision output with factory status input, all in one system. Basic sequencing functionality now only comprised one-third of the system itself.

The scheduler’s task was radically changed through the introduction of the system. Instead of doing piecemeal activities throughout the morning, the key activity was getting the Gantt schedule established which would then drive many of his other outputs. He still did partial decision processing at 6:15 a.m., 6:30 a.m., and 7:00 a.m., but the system facilitated the process and allowed him to do it in a partial fashion in a quick and easy way. The new dialogs and interface presented information automatically without extraneous data displayed so the scheduler could do his tasks faster, or as fast, as in the past. The benefit was that the task output was part of the system and could be integrated with the decision making done later in the day without replication or duplicity. By 7:30 a.m., the scheduler basically had almost all of the reports done that used to take until 10:30 a.m. to craft. The scheduler was not planning out 2 weeks (he allowed the basic scheduling logic to drive the future). He was now looking at 2–3 days in the future which was a significant improvement over past practice.

These scheduler’s information system changes and software requirements were outside of the original mandate for the problem. The scheduler had been optimistic and as it turned out, was not even consciously aware of everything he did during the weekly scheduling task to fit it into his own (now daily) task schedule. A detailed time study of the scheduler would have brought some of the issues to light. However, other reactions by management and supervisors would have been difficult to capture. If the PPS tool was still used in a weekly fashion, few of these changes would have been necessary. The whole issue related to the daily change in routine. The scheduling logic and basic structure was retained and only a few minor changes were made to the Gantt chart representation and user interface. Hence, the problem was not related to the implemented functionality, but rather to the missing functionality.

What were the warning flags that, if detected, could help a similar development? They needed the tool in October 1996 and did not get an operational tool until January 1997, thus resulting in cost and delay. The possible warning flags to recognize were:

- Changing the frequency of the task should have been a warning flag. A detailed study of what happens before, during and after the task should have been done. What other tasks are done in the same time window, what conflicts exist, what happens when 10 kg are put in a 5 kg bag? How is data usage affected? Are different people now involved?
- Changing the accuracy expectations of the task should have been a flag. Accuracy and performance metrics affect people and personal agendas. The source of data is affected, more communication is required to vet the data, and the users of the data will challenge the results with more vigor which will impact the time and effort needed to use the tool.
Changing the amount (scope) of decision making should have been a warning flag. The scheduler used to develop a 24-h dispatch list daily and a 2-week schedule once a week. Management now wanted a rolling 2-week plan. The scheduler was ultimately capable of doing 2–3 days with a rough-cut finite load view of the future. Although not what management wanted, it did identify future conflicts and provided a more detailed plan for the next 48 h. There were too many resources in an unstable situation for the scheduler to manually resolve conflicts for each day in a rolling 2-week period. Conflict resolution would have required higher-order problem solving beyond simple sequencing. He would have needed to delay jobs, split batches, ‘create’ new resources and negotiate almost every option with various players. Time studies were done after the fact to explain why the scheduler could only do so much. The 2-week expectation had been warned about, but it was a dream and driving force in the project.

These three flags, frequency, accuracy and scope, turned out to be significant task modifiers requiring substantial development effort. We are not aware of these issues explicitly discussed in the DSS literature, nor production planning control literature. Perhaps a socio-technological task analysis might have picked up the issues, but the types of changes did not reveal themselves until the system was being used—until they had it, they did not know how they were going to be able to use the output and what this might imply.

As shown in Table 1, the code tripled and many functional areas doubled in capability. In 2 months, the majority of changes had been made and the new system was ready for live use in January 1997. How did this rapid growth and development impact the original base code within the Pilot software?

The scalable architecture using object-oriented concepts and the list—task engine approach permitted rapid growth of the system; without causing the software to become fragile or troublesome. The other information system requirements were also easily integrated within the underlying system without any structural changes to the platform. The quality and robustness of the software was not compromised. In the first 1.5 years of use, the system was in use almost every day, 7 days a week, 8–10 h a day. During that time, the system had several dozen minor tweaks to reports and manipulation functions. None of these minor problems prevented the system from performing its primary tasks and supporting the scheduler. In all but one case, each minor problem was fixed in less than 1 h. There was one major system problem that prevented the system from being used for one half day. Otherwise, the system ran and did what it was supposed to do.

6. Evolution from legacy MRP system to SAP ERP system

Recalling Table 1, we note that, despite the fact that the physical products and production processes did not change, the move from the legacy MRP system to the SAP ERP system resulted in roughly double the code in general, a small increase in the scheduling engine logic, 30% more dialogs, 50% more reports, 50% more fields in the part and scheduling databases and more types of data imported into the system. Because the products and processes were the same, the basic sequencing technology remained largely similar, as did basic dispatching-related tasks and daily information flows. However, the ERP migration did affect the tasks and activities related to scheduling. The workload in the scheduler position increased to 1.5 people (instead of one), and the renewed expectation of a 2-week planning horizon occupied another 1/2 person. A significant portion of the redevelopment effort addressed this additional expectation. Other major activities related to replacing portions of the legacy system which were not supported by the ERP system and supporting data and information requirements required by the ERP.

The legacy system modeled the press shop as a supply sink (e.g., build a certain quantity by a certain date). There was no real model of the manufacturing situation in the legacy system. In SAP, the shop was modeled as a discrete parts producer, and the assembly area was modeled as a repetitive (flow) area. This resulted in actual production orders for dependent demand in both the blanking and stamping areas. Instead of simply reporting part production, it was now tracked against specific production orders in SAP. Utilizing the production order paradigm was the only way to model and map the press shop.

At the dispatch level, it was important that only firm orders were released to the shop floor. Unlike before, orders and tracking existed for the blanking operations. These were the only changes to the daily dispatching process. What other activities were implied by the ERP conversion? First, planned or temporary stamping orders had to be converted to firm status. Second, blanking orders also had to be firmed and synchronized to the firmed stamping orders. Third, since firm orders did not float or get adjusted in the daily MRP regeneration, order maintenance was needed to monitor changing requirements due to assembly conditions. These activities were estimated to require roughly 3–4 h a day. It was also estimated that smoothing and balancing the schedule for a 2-week horizon would require 2–3 h each day. The extra workload justified having two schedulers; the question was how to split the task and support the functions.

The scheduling activity was split into two components—dispatching and sequencing. The dispatching component dealt with firm orders and a 48-h horizon. This component supported the dispatching function and all associated shop floor paperwork. The dispatcher would perform the daily reactive procedures. Although much of the shop would run as expected, the shop was large enough such that something was happening somewhere all of the time that required higher level problem-solving and often the solutions echoed across multiple resources. The use of the ERP was minimized in the dispatching task and key information was presented in a pre-digested fashion. The dispatching tool imported the firm orders from the scheduling tool. The dispatching component ‘owned’ the 48-h zone, did not see beyond it and did not see any planned orders. To maintain a coherent view of production, dispatch schedules
and decisions would be periodically transmitted to the sequencer component. In terms of user interface functions, the 6:15 a.m., 6:30 a.m. and 7:00 a.m. functions remained practically unchanged to ensure that the daily production routine would not be negatively affected by the burden of the ERP system.

The sequencing component had a frozen 48-h zone, saw the dispatching situation in a read-only mode and had a view of all firm and planned orders for 8 weeks. The goal was to smoothly and firmly up 2 weeks of production so that resource planning could be stabilized and that upstream pulling of steel could be smoothed. Changes in the pull demand from assembly would be analyzed and the 2-week view adjusted accordingly. Dealing with production orders and blanking resources was totally new. Additional reports were needed to analyze orders for conflicts, problems and to provide other diagnostics. The sequencer would progressively firm up and move work into the 48-h zone for the dispatcher to consider.

The total number of functions faced by a dispatcher or sequencer dropped from the previous 124. As shown in Table 1, the sequencer saw 110 functions and the dispatcher saw 115 functions, with 93 functions in common. This split simplified the dispatcher’s (the senior scheduler) task, thereby allowing him to focus more on dispatch decisions than with the pre-ERP system in which he was also responsible for some planning. The dispatcher’s task was a richer domain and required deeper knowledge about how problems can be solved and what assignments were feasible and infeasible. The sequencer’s task was based upon primary or preferred resource assignments with one alternate resource—some knowledge was necessary, but not to the same depth or richness as the dispatcher.

Approximately, 15,000 additional lines of code were required to support the dispatching and scheduling logic within the SAP ERP environment. The dispatcher had most of the original reports while most of the sequencer’s reports were new. The specialized functions for dispatching were not new, but specialized functions for dealing with planned and firm orders were needed for the sequencer. The basic user interface (e.g., screens, job manipulation) were the same as before for both the dispatcher and sequencer with the minor addition of interlock logic to prevent the sequencer from changing the dispatcher jobs and restrictive logic to limit the jobs that the dispatcher could view.

In addition to the dispatcher and sequencer modes, the staff wanted the system to be capable of working ‘as one’ on the weekends when the sequencer or dispatcher was absent. This feature required the system to act in ‘Master mode’ as well. This mode was also used in a conference room setting for several months to facilitate real-time scheduling adjustments during morning management meetings during a particularly volatile period.

The pre-ERP system was converted in parallel during the 2 months prior to going live. Final formats with live data were obtained the day before the switch, and a system test was conducted overnight to ensure a functional system the next morning at 6:00 a.m. Since the sequencing engine and basic user interface were decoupled from schedule analysis, the conversion (while hectic) did not encounter any structural problems. The ‘MRP Job Importer’ was also decoupled from the engines so that SAP input streams could quickly be adapted for. The inclusion of blanking and stamping resources, where previously only stamping existed, required substantial work since blanking jobs were different from stamping jobs in many ways. The importing, exporting and merging functions also required substantial effort to design, implement and test.

To address the variable functionality while allowing the system to be maintainable, a flexible structure was used. The software senses the name of the file launching PPS and based on what the filename is configures the PPS tool for dispatching, sequencing, or both. The configuration alters the menus, functions on the menus, what the functions can do, and what reports are generated. For example, deleting work on the plan is different if the work is in the dispatching mode versus scheduling (planned versus firm orders). Roughly 80% of the software is used in all three modes and a dynamic front end appeared to be an effective way to support the ongoing evolution of the software. It took a review of each function to determine if there should be a difference in the function within the dispatching or sequencing modes. However, once done, a coherent system existed that facilitated easy support and maintenance.

Steel ordering had to be based on the finite loading and sequencing in the schedule, not the short-term EDI (electronic data interchange) in ERP based on lead times and infinite loading. Hence, the PPS added the functionality to net blanks and steel against the demand and to create a steel order profile for each mill. With multiple levels of the bill of materials now to model, the original Gantt chart sequencer had evolved into a mini-MRP system. Steel netting and ordering were major components of the new version.

In summary, the ERP migration provided several key insights into the production control situation. First, scheduling and planning effort at the site increased by 50% as a result of converting from an ‘orderless’ production control system to one involving orders and tracking. Second, the new system provided insight into what happens when the dispatching and sequencing tasks are too large for one individual and when two small focused factories are not viable to create, thereby requiring discussion regarding work splitting, how to work together in a complementary fashion and the how to design a system with split functionality. Third, it provided insight into the reports and diagnostic aids needed by a scheduler when dealing with large numbers of orders within an ERP environment. Fourth, by integrating steel ordering within the finite sequencing logic, the full complexities of modeling a press shop became evident. The modeling requirements were challenging with many possible combinations of options available.

7. Maturity

The third evolution was driven by management changes and minor changes in the way the factory works. It started after the SAP ERP implementation and has continued. There have
always been minor (and sometimes major) changes when a plant manager or production control manager has changed. The majority of changes during this phase of usage have been in the reporting area. Almost all of the reports have changed somewhat, a number of the reports are no longer used, and some new ones have been crafted. Overall, the main characteristics, as shown in Table 1, indicate that the system has remained largely the same. During one 3-year period, no changes were made to the code—functional or code fixes. The management team was stable, the production environment did not change, and the PPS tool was used 7 days a week, from 5:30 a.m. until 8:00 p.m. each day.

Recently, the scheduler’s activities have been monitored and analyzed. If the scheduler is doing things that can be better done in the system, small development activities are undertaken. For example, if the scheduler is hand crafting reports outside of the system or doing redundant data entry associated with reports, code revision may be possible to reduce the effort. In January 2006, such an activity saved the scheduler almost 2 h each day. The scheduler now has more time for actual decision making instead of clerical functions. It was also observed that much of the morning routine was very stable and could be automated using larger functions. What used to take five to seven functions to perform now is done using one function. This has helped other individuals, including the scheduler’s manager, to step in when necessary and use the system.

8. Discussion

This case study has presented three types of system evolution. The first evolution converted the fully functional Gantt chart sequencer to a scheduler’s information system. This was caused by changing the frequency, accuracy and scope of the scheduling task. These changes should have raised warning flags and altered system deployment expectations. In other words, a longer development time should be necessary with a true parallel and test mode. The substitute scheduler should have been assigned to try daily parallel generation. Perhaps January 1997 would have been the date to go live, but there would have been less frustration and hectic activity during Fall 1996. This challenge may be one of the reasons why many scheduling systems fail in practice and do not find sustained usage after implementation. If the scheduler’s manual information system is disrupted and the new system does not offset this disturbance, it is unlikely that the scheduler will want to use it or assist with its success.

The second evolution was triggered by the migration from the legacy MRP system to the SAP ERP system. In addition to extra data and logic requirements, this migration altered the scheduler’s task flow and task load. The introduction of order-based processing and increased tracking required by ERP increased the scheduler’s workload by 50%, thus resulting in the task being too large for a single individual and the previous dispatching/sequencing situation transformed into separate dispatching and sequencing activities. Task sharing and information flow represented a significant portion of the evolution. Removing steel ordering from ERP and modeling the blanking and stamping also raised the functionality from a simple job sequencer to a mini-MRP system with finite capability.

As the system matures, a continuous form of evolution is to be expected. Reports and day-to-day routines can be expected to be scrutinized at regular intervals and mechanisms should exist in the software for this purpose. In PPS, the interfaces and engines were decoupled from the beginning and all new functionality was added in a consistent, decoupled fashion. For a decade the system has been in daily usage for 8 h a day and has proven itself to be very stable and robust. Changes were made in a controlled fashion to minimize disruptions to the scheduling task. One of the keys to the technological evolution was the choice of the agile development approach and sound software engineering concepts that permitted the system to grow rapidly without losing integrity. One person performed the design and coding of the system during its complete lifecycle which contributed to the consistency and robustness of the system. The developer was a professional systems architect with over two decades of systems programming experience. Previous experience had provided evidence of problems when using student or junior programmers for developing scheduling systems and, thus, was avoided in this project. Scheduling systems are vital and must be robust for schedulers to actually use them. A system that is not robust will not be trusted or utilized.

Since the DSS was used in one department within a large plant, it is not possible to link DSS usage with overall plant performance. Within the department, commenting on the benefits attributed to using the DSS is also difficult since the department was large and performed multiple continuous improvement activities simultaneously. However, the use of the DSS has been the subject of regular reviews during the past decade and while specific numbers cannot be made public, the following observations have been made by the plant about the software’s use. First, without the custom DSS for dispatching, sequencing and steel ordering, approximately two to three more people would be required in the scheduling role when SAP was migrated to. Second, the number of production errors attributed to miscommunication between various production departments dropped dramatically with the integration of data in the DSS and the resulting data dissemination. Third, the number of production errors attributed to the scheduler dropped substantially with the full integration of the scheduler’s information in the DSS—reducing errors associated with data entry, overlooking special statuses, and so forth. Fourth, the number of unnecessary setups was reduced as a result of better sequencing and reacting. Fifth, the department was running smoother with better resource allocation. Sixth, another person would be needed in the scheduling role if a standard, commercial APS tool was used unless a large amount of customization was done to support the scheduling task. The periodic review of the DSS has been done with the objective of justifying its ongoing use versus the acquisition and implementation of one of the leading APS tools. The custom DSS has been considered very effective by
the plant and the plant has continued its use thus far. The list of benefits cited by the plant has focused on supporting the scheduling task and not on the better sequencing per se. This supports the primary insight of the case study—the importance of understanding the difference between sequencing and scheduling and supporting the latter.

9. Conclusion

Decision support tools for planning and scheduling are complicated entities. They need to ‘fit’ the scheduler’s tasks and support information processing beyond simple sequence generation. Understanding the impact of frequency, accuracy and scope changes on the task and the resulting disruptions in the current scheduling process appears to be important in the development and deployment of the decision support systems. The introduction of added functionality can also force a task to outgrow a single individual. However, since the tasks of planning and control cannot be totally independent among individuals, a task sharing structure might also be required and this should be considered in any system design.

It is not obvious from the literature if the current generation of scheduling and planning tools (finite capacity schedulers and advanced planning and scheduling systems) address these issues. From reviews and experiences with a number of the commercial systems, it still appears that the focus is on the sequencing and creating the initial plan and not on the actual tasks of planning, scheduling, and dispatching [2]. These are clearly areas for future research. We have been able to build and probe in one situation, but the lessons are not necessarily generic or prescriptive.

A flow-shop tool was created for the assembly area of the factory and the user-interface and functionality suite are almost 100% different from the job shop tool. An agile development approach was also used and similar results obtained—excellent user adoption, robust tool, and rapid delivery of a production quality system. A significant difference is that the flow-shop area functions in a focused-factory fashion with one individual planning, scheduling, and dispatching. While not discussed in this paper, the contrasts between flow and job shops, and hierarchical and focused factory schemas are also suggested as areas for fruitful research. Solid taxonomies and guiding principles are required.

Finally, the impact of an ERP implementation on production planning and control was also observed in this case study. It was observed that although ERP systems can yield substantial benefits in many parts of the organization, it is possible in certain situations that the implementation can lead to additional personnel requirements within production control.

Acknowledgements

This research has been supported in part by NSERC grant OGP 0121274 on ‘Adaptive Production Control’. The authors also acknowledge the thoughtful and valuable suggestions by the reviewers.

References


Kenneth N. McKay is a professor in the Department of Management Sciences, University of Waterloo. Research interests include socio-technical systems involving the human decision maker in production control systems, hierarchical production control paradigms, adaptive heuristics for risk management, advanced planning systems, context aware interfaces for decision support systems, temporal context modeling, and the evolution and history of production management. In 2004, "Practical Production Control: A Survival Guide For Planners and Schedulers" was published (Japanese edition in 2005), Vincent Wiers co-author.

Gary W. Black is an assistant professor of production/operations management at the University of Southern Indiana in Evansville. His research interests include adaptive heuristics for production control applications. The current thrust of this research agenda is studying empirically observed risk mitigation techniques used by schedulers.