Analysis of manufacturing systems by the Research Queueing Package

by We-Min Chow
Edward A. MacNair
Charles H. Sauer

Many aspects of manufacturing systems can be analyzed using simulation to model the system's behavior. The Research Queueing Package (RESQ) is a tool developed to construct and solve models of systems with jobs contending for service from many resources. The capabilities of RESQ are described in order to understand the model elements which are available for representing manufacturing systems. Then an analysis of several work-in-process (WIP) policies is presented using RESQ models solved by simulation. Four WIP management policies are analyzed and compared for a future assembly manufacturing line: (1) a push system, (2) a pull system, (3) a transfer line, and (4) a closed loop system.

1. Introduction

Manufacturing systems are very expensive to design. Once they are implemented, it is sometimes costly to improve their efficiency. When designing a system, it is often difficult to decide which of the many possible alternatives would give the best performance. After a system is running, it is a complicated task to improve its operation and plan for future changes. Because of contention for service, limited waiting areas, parallel operations, simultaneous activities, multiple interactions, and complex decision mechanisms, the operation of manufacturing systems is not easy to predict. The complexity of such systems requires the use of methods and procedures to understand their behavior.

Performance modeling [1–4] is a technique employed to study the behavior of manufacturing systems. Modeling is an art which requires much intuition to accurately represent the behavior of these systems, and the methods used to produce the performance measures of the models employ sophisticated mathematical techniques. Modeling tools help simplify an analyst's job of constructing and solving models, and the Research Queueing Package (RESQ) [5–10] is a tool developed to aid performance analysts.

Manufacturing systems are composed of many components which we call resources. Examples of resources are work stations, tools, buffers, robots, storage areas, presses, baths, transfer units, conveyor mechanisms, and people. Customers make use of the resources, visiting the resources and requesting service from them. While a customer is receiving service, other customers can arrive to request service from the same resource. This causes contention among the customers for the resources and results in queues or waiting lines. The amount of contention

and the length of the service requests affect the behavior of the system. This contention for resources in the system is the fundamental issue which must be understood in studying the operation of these systems.

In this paper features of RESQ are briefly discussed and illustrated by a model that solves an actual problem. Section 2 presents RESQ modeling elements and functions.

Section 3 discusses four operating policies for an assembly line: (1) a push system, (2) a pull system, (3) a transfer line, and (4) a closed loop. A description of the assembly line is presented along with details of the four policies. The section also includes an overview of the RESQ models and an explanation of the results. The Appendix contains the entire model with annotations.

2. RESQ modeling elements, solution methods, and performance measures

The model elements are the building blocks of a modeling tool. Once we understand what the model elements are and how to use them, it becomes a relatively simple task to use a tool like RESQ to build a model of a complex system if we understand how the system operates. We just need to know which elements to use and in which order to use them. By putting the building blocks together in different fashions, we can construct many different models.

• *Customers*

In building models we focus our attention on the customers circulating through the model and demanding service from the resources. The customers can represent many different kinds of entities. They can be people, manufacturing tasks, parts to be assembled, and many more items found in systems. Each place a customer visits in a model is called a node. There are several different kinds of nodes, which represent various kinds of actions performed when a customer arrives at that location. The collection of nodes visited by a customer and the order in which they are visited is referred to as the routing.

Different kinds of customers are distinguished by different attributes. Some examples of attributes of customers include the type of job, the priority level, the number of times the customer should visit a portion of the model, the time of arrival at or departure from a particular node, and many other identifying characteristics. The attributes are attached to the customers and can be interrogated while making routing decisions or when determining how much service a customer demands.

There are certain instances when we want the customers to make copies of themselves, with the original customer and the copies possibly proceeding over different paths. If we want 100 pieces of a subassembly to arrive at a service center all at the same time, we can have one customer split itself into 100 separate customers which then progress separately through the model. The copies which are produced can be independent customers which follow different paths, or they can be related to one another and join back together again at an appropriate place in the model.

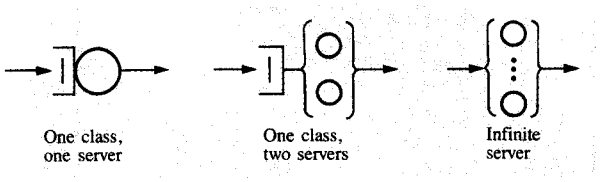

Figure 1

Three service centers.

Different nodes are associated with the generation of customer copies, depending upon whether the customers are related or not. The nodes are called split, fission, and fusion nodes. Customers passing through a split node generate unrelated customers. The fission and fusion nodes are used in pairs. Customer copies generated at a fission node are joined together at a corresponding fusion node. Only a single customer leaves the fusion node. This occurs after all of the related customers arrive.

• *Service centers*

Service centers are the major model elements used in RESQ. They are composed of one or more servers, one or more waiting lines, and a queueing discipline or scheduling algorithm for determining which customer to put into service next. The customers arrive at the service centers and request a certain amount of service. This service is usually determined by a service time distribution specified when defining the service center.

Figure 1 shows a service center with a single server, one with two servers, and one with an infinite number of servers. A circle is used to represent a server at an active resource. The waiting lines, which are shown as a rectangle with one side missing and a vertical line in the middle, are also called classes. Some service centers have more than one class. Several reasons for having multiple classes at a service center involve specification of different service time distributions, different priority levels, and alternate routing paths. The classes are the nodes at service centers which are used in the routing definition. Since there is no waiting at an infinite server, there are no waiting lines shown with the symbol of the IS center. There can still be multiple classes at an infinite server, so that customers can have various routing paths.

When a customer arrives at a service center other than an infinite server, the customer waits in line until a server is free. When a server is available, customers are scheduled according to the queueing discipline. Some commonly used queueing disciplines include first-come-first-served (FCFS), last-come-first-served (LCFS), processor sharing, round

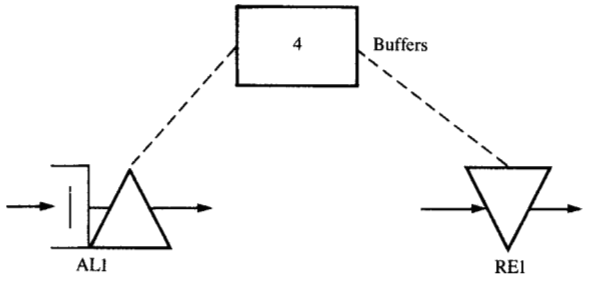

Figure 2

Allocate and release nodes.

robin, preemptive priority, and non-preemptive priority. The queueing discipline determines whether the service may be preempted by other jobs arriving at the service center or whether the server is shared among the customers. A customer's activity is usually focused on the resources of a service center and typically has no interaction with other modeling elements while at a service center.

When a customer is put into service, the amount of service requested can be specified in two different ways. The first approach is to specify a service time equal to the amount of time spent in service during a single visit to the service center. The second method is to specify an expression for the work demand and a rate of service. The rate of service is the amount of work a server can perform in one unit of time. The service time is then calculated as the work demand divided by the service rate.

The amount of service requested is normally specified by a probability distribution. Some of the commonly used probability distributions are constant, discrete, uniform, Erlang, exponential, hyperexponential, and normal.

• *Passive centers*

Given only customers and service centers as model elements, there are many situations which exist in systems which are difficult or impossible to represent accurately. Passive centers permit us to represent many of these complex features.

Allocate and release

Passive centers are mainly used to model a resource which has a limited number of elements that are allocated to customers, held by the customers while they receive service at service centers, and then released by the customers. The major difference between service centers and passive centers is that a service center has one or more servers actively engaged in providing service to customers. This is not the case at a passive resource, where there are no servers actively providing service. However, there are elements called tokens which are in some ways analogous to servers. There is usually a limited number of tokens at the passive center. These tokens can be used to represent a finite number of elements of a resource such as the number of positions in a buffer and other limited resources.

As an example of a passive center, we will consider how to represent buffer contention. **Figure 2** depicts a passive center with the number of tokens, which is shown in the box, being equal to the number of buffer positions. The rectangular box is the pool of tokens from which the buffer positions are requested. AL1 is an allocate node where customers request tokens. If the number of tokens remaining in the pool is less than the number of tokens a customer requests, the customer waits in the line associated with the allocate node until a sufficient number of tokens become available. It is important to remember that customers retain possession of the tokens until they are explicitly released. Tokens are returned to the pool when a customer holding tokens from the passive resource passes through release node RE1. The customer flow is shown with solid lines and arrows. The flow of tokens is illustrated with dashed lines.

The passive center facilitates the representation of many situations where customers simultaneously hold multiple resources. A customer acquiring tokens from a passive center can request service at service centers and can also request tokens from other passive centers. This type of model element is a very powerful extension to conventional queueing networks.

Figure 2 shows one allocate node and one release node. However, there is no restriction on the number of allocate or release nodes which belong to a passive center. There is also no one-to-one correspondence necessary between allocate and release nodes. There can be any number of allocate nodes and any number of release nodes.

Create and destroy

With only allocate and release nodes, there is no way to change the number of tokens at a passive center. There are times when we would like to increase or decrease the number of tokens. This can be accomplished as shown in **Figure 3** by using create and destroy nodes. A customer going through a create node adds a specified number of new tokens to the pool, and a customer holding tokens when it is routed through a destroy node discards the tokens it holds. This permits the number of tokens associated with a passive center to change dynamically. One use for these model elements is to hold customers in the queue at an allocate node until another customer creates tokens for them to advance. This is a type of synchronization which is very common in contention systems. When a passive center is used to count the number of parts in an assembly work station, for instance, arrivals of part trays can be modeled by a create node, while consumption of parts is represented by a destroy node. An assembly job can be processed only if there exists a sufficient quantity of parts. These model elements

can also be used for communication between independent processes.

• *Model variables and status*

In order to make certain types of decisions, we use two different types of variables. One type of variable, which was discussed briefly in the section on customers, is used to hold the set of attributes of each customer. The variables which contain the customer attributes are called job variables. The other type of variable is accessible by all customers. This type of variable is a global variable in the sense of a variable in a programming language. As customers proceed through the model, these global variables can be assigned values and used in ways similar to the job variables.

All job variables are automatically initialized to a value of zero, and global variables are initialized to a specified value when they are defined. They must be explicitly assigned other values as the model is progressing. In order to assign values to the job variables and the global variables, we need a special kind of node which we designate as a set node. As customers pass through a set node, one or more assignments are made to the job variables or the global variables. The symbol for a set node is a rectangular box, as shown in **Figure 4**. The assignment statements can contain expressions. In this case a global variable counting the number of failures is being incremented by one.

In addition to making decisions based on the values of variables, we can also interrogate the status of various conditions of the model. Some of these conditions include the queue lengths at classes, service centers, allocate nodes, and passive centers, the number of servers or tokens which are available for service or allocation, the number of customers related to a customer which has gone through a fission node, and the number of tokens a customer holds from a passive center. These conditions provide the status of the model necessary to control the flow of customers through the network.

• *Chains*

Chains are used to classify different types of customers into different collections of routing paths. A path consists of all the nodes visited by a customer. When a model contains different types of customers, it is convenient to define separate chains for each customer type. However, different types of customers may belong to the same chain. In this case, the customer attributes can be used to identify the customer type, and routing decisions can send customers to different resources.

By using different chains for different types of customers, the model need not explicitly check the customer type. This is implicit in the chain to which the customer is assigned. Customers and nodes are uniquely assigned to one chain. A customer in one chain can never visit a node which belongs to a different chain, and a node which belongs to one chain

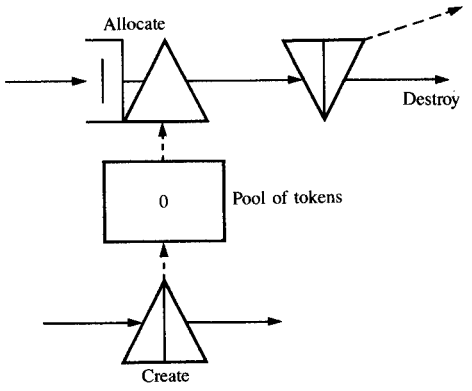

Figure 3

Create and destroy nodes.

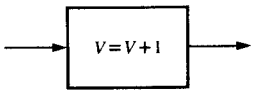

Figure 4

A set node.

cannot be used in a routing statement of another chain. This does not preclude customers in different chains from contending with one another. A center representing either an active or passive resource can have nodes belonging to different chains. The customers belonging to the different chains can still contend for the same servers or tokens.

There are two types of chains. An open chain usually contains one or more sources where customers enter the chain and a sink where customers depart. It is possible for an open chain to contain no source. In this case, customers must be initialized at one or more nodes of the chain. Open chains permit the number of customers in the chain to vary. There are customers arriving and departing at various times, and the number of customers present is continually changing. An open chain is frequently used to model a system where the population is not static.

A closed chain contains a fixed number of customers. We specify the chain population, and these customers always remain in the chain. A system that has a finite number of customers is conveniently represented by a closed chain. An example of this is using a closed chain to represent jigs on a circular conveyor. When we can identify a relatively small, fixed number of customers in a chain, a closed chain is the appropriate model element to use.

One other type of node which is sometimes useful in routing statements is a dummy node. Nothing happens to a job at a dummy node. This type of node provides a place for a job to visit with no actions occurring. An example of using a dummy node to initialize jobs at the beginning of a simulation is given in Section 3.

• *Submodels*

A submodel is a parameterized portion of a model. We can define a submodel which contains any subset of resources present in the model and make one or more copies of the submodel. Submodels can be used to clarify the structure of a model, to avoid duplication of effort within a model, to permit sharing of parts of models, to vary the number of model elements at solution time, and to solve the submodel separately and replace the submodel with a flow equivalent server.

The structure of the model can be clarified by constructing submodels for the major subsystems to be represented. The submodels can be used to represent high-level abstractions of the subsystems which can be easily connected to form the overall system. If a model contains subsystems which are similar, we can construct a submodel representing one copy of the subsystem with parameters which capture the differences. Then the submodel can be duplicated for each subsystem with different values supplied for each copy of the submodel. This is sometimes referred to as hierarchical modeling.

Very frequently models are required to have a variable number of resources. The number of resources can be specified as a model parameter, and a submodel can be built to represent one of the resources. RESQ permits an arbitrary number of copies of the submodel to be created based on the value of a model parameter.

Hierarchical decomposition is a widely used technique for simplifying the solution of certain types of models. The model is decomposed into one or more submodels which are solved separately from the remainder of the model. Results from a submodel solution are used to characterize a flow equivalent server, which is used in place of the submodel in an aggregate model. The flow equivalent server is usually a queue-dependent server with appropriately chosen service rates.

• *Solution methods*

RESQ permits the use of two solution methods. An analytic solution involves solving equations to produce the performance measures. Simulation is a statistical experiment which observes the behavior of the model and generates the performance measures from the observations. An analytic approach is usually a faster solution method and is preferable when it is applicable. The problem is that many simplifying assumptions must be made in order to be able to solve a model analytically. Simulation is much more general and can be applied to very complex situations. The price which must be paid for this generality is the longer time required to obtain accurate performance measures.

Because of the randomness inherent in the results of a simulation, a statistical analysis must be performed on the results to indicate their accuracy. RESQ provides three methods for producing confidence intervals and an automatic stopping procedure for detecting a specified level of accuracy.

• *Performance measures*

We are interested in several different performance measures calculated by the solution technique. The utilization of a resource is the fraction of time a server or token is busy. The queue length is the number of customers waiting or in service. The throughput is the customer completion rate, or the number of customers that complete their service in a given unit of time. The queueing time is the time a customer spends in the waiting line and in service at a center. The queueing time at a passive resource is the time a customer spends waiting for and holding tokens. The time it takes for a customer to travel between any two points in a model is called the response time. We are interested in mean values for these results and sometimes in the distributions of the queue length, the queueing time, and the response time.

A useful feature of RESQ is the output analysis performed on the simulation results. There are three methods of producing confidence intervals and a sequential sampling procedure for detecting when the results have reached a specified level of accuracy. It is also possible to delete some of the transient data at the beginning of simulation runs.

The next section discusses a RESQ simulation study for the work-in-process (WIP) management policies.

3. Analysis of WIP management policies

One of the major problems in a manufacturing system is the control of work in process. WIP is a semiproduct in a production line, including the items in the tools as well as those awaiting processing or transportation. A good WIP management policy can eliminate unnecessary on-line inventory, save storage space, reduce material handling cost, and improve the system throughput.

Consider a manufacturing system that is composed of a number of different work stations. If a work station is manually operated, the cycle time for producing a piece of product is typically a random variable, due to the inconsistency of human behavior. On the other hand, an automated work station is subject to failure and a production cycle time may be interrupted by breakdowns. Because of cycle time variation, a work station may temporarily become a bottleneck and keep the stations immediately downstream idle for a significant amount of time. Consequently, cycle time variation means underutilized work stations and possibly an insufficient

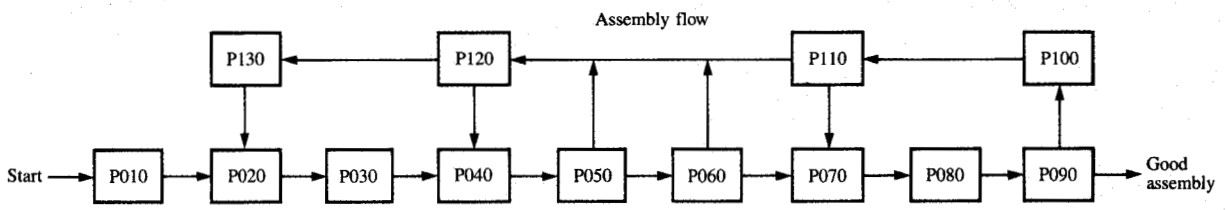

Figure 5

Assembly flow chart.

system capacity. There are at least three alternatives for solving such a problem:

1. Increase the system capacity by installing more work stations.
2. Control the cycle time variation by introducing more automated work stations and improving the tool reliability.
3. Allow more WIP so that no work stations can trap enough WIP to keep others idle.

Choosing among these alternatives is a complicated problem. One must consider the ultimate manufacturing cost. In this paper we consider the last approach only. A future assembly line is used as the study case throughout the discussion in this section. Different WIP management policies are investigated by a RESQ simulation model, and descriptions of the models and their results are presented.

• *Description of system*

The assembly line under consideration consists of 13 different operations, including assembly, inspection and test, and rework stations. The manufacturing process flow chart is shown in **Figure 5**, where operation numbers P050, P060, and P090 are inspection and test, P100 to P130 are rework, and the rest are assembly operations. The yield factor observed at each test station is a random variable ranging from $0.90y$ to $1.05y$, where y is the average yield. The defective assembly units are sent to rework stations. The rework flow does not necessarily have a single path. Different routes may be taken, according to a certain probability distribution. Let p_{ij} be the routing probability from operation i to operation j and e_i be the start factor at operation i (the expected number of visits to operation i by the same assembly job). For a given $\{p_{ij}\}$, $\{e_i | i = 1, 2, \cdots, 13\}$ can be obtained by solving

$$\sum_{i=1}^{13} p_{i1}e_i + 1 = e_1,$$

$$\sum_{i=1}^{13} p_{ij}e_i = e_j, \qquad j = 2, \cdots, 13.$$

The mean cycle time of each operation has been estimated by engineers. According to past experience, it is believed that a manual operation cycle time can never be less than 80% of its mean value. Since we are dealing with a new assembly line, neither the distribution nor any higher moments are available. In this study we investigate WIP policies with different coefficients of variation by using the distributions provided by the RESQ package [5–10].

Under the current management policy, the assembly line is operated for eight hours a shift, three shifts a day. Each operator may have two 15-minute break periods and one 40-minute lunch time. During a break the line is never shut down; only one half of the operators leave the line. After their return, the other half take a break for the same amount of time. The automated work stations are operated continuously. Thus the manual stations and the automated stations have different effective working hours.

Work stations may be subject to failure. The well-known machine repairman model [11] can be used to evaluate the station reliability as a function of the interval between successive failures, the repair time, the number of repairmen, and the number of stations.

For a given daily production demand D, the minimal number of work stations of operation i, n_i, is the least integer greater than or equal to

$$\frac{e_i D}{T_i r_i / s_i},$$

where

T_i = effective working hours for operation i,

r_i = the reliability of a type-i station, and

s_i = the mean cycle time of operation i.

Note that the line capacity is by no means equivalent to the capacity derived from the tooling only. Very often the material handling system can be the bottleneck. Design of the material handling system should be closely related to the WIP management policy. One simple example is the use of a conveyor system for WIP storage and transport from one

work station to another. A relatively short conveyor line limits the WIP size and therefore reduces the line throughput. We do not consider any specific material handling system. Instead we assume that there are limited buffers associated with operations. The material handling delay is small and negligible, at least compared to the operation cycle times.

• *WIP policies analyzed*

We are going to compare four different WIP management policies:

1. a PUSH system,
2. a PULL system,
3. a TRANSFER LINE, and
4. a CLOSED LOOP system.

In Method 1 one acquires all assembly parts according to a production schedule. Typically the quantity of each individual part is determined by the material requirement planning (MRP) technique [12]. Parts are "pushed" to the assembly line until the staging area becomes empty.

Method 2 is derived from the well-known KANBAN system [13], in which each operation has two buffers, one for input and the other for output. In this study, we assume that each operation has a single buffer for both input and output. This buffer, for instance, can be a multilevel storage rack placed in front of work stations. WIP can be stored into and retrieved from the rack by an automatic storage and retrieval device. The same rack can be used for both input and output queues. When the buffer is full, no input is allowed and the immediate upstream operation may be blocked. For the same total buffer size, the buffer sharing policy has less blocking probability than the conventional KANBAN system.

The ideal case for a transfer line (Method 3) requires a fixed uniform cycle time for each operation so that subassemblies are moved from operation to operation at the same pace. If all the cycle times are constant but not uniform, the line throughput is uniquely determined by the operation that has the minimal ratio of the number of work stations per operation to the cycle time [14]. We are sure that the WIP level is minimum. Thus in a transfer line we assume that no buffers are used.

The closed loop system (Method 4) attempts to control the WIP level through parts feeding. A simple procedure has been developed to synchronize the feeding speeds of different parts. Logically we may consider that each feed is composed of a set of parts for exactly one assembly. This procedure links the logical feed to the physical material moves. We illustrate this procedure by a simple example as follows.

Assume that all parts are delivered by trays and part type j is used at operation i. The packing density of part type j, d_{ij}, equals k assemblies per tray; i.e., the number of parts in the tray can make k assemblies. The part-j feeding sequence is characterized by a list of elements. For instance, if $n_i = 3$ and $k = 2$, the feeding list is

$$L_j = (1, 1, 1, 0, 0, 0).$$

Each element in the list corresponds to an assembly. The value of an element indicates that that number of trays should be fed to the line. According to L, three trays of part j are sent to three type-i work stations, respectively, for the first three assemblies. Since each tray contains two assembly parts, no additional feed is needed for the next three assemblies (this is indicated by three zeros). L_j is used as a wraparound list such that the feeding schedule for part j starts all over again for every six assemblies. If a bad part is detected, the list content may be modified by shifting all elements to the left. Thus the pattern becomes $L_j = (1, 1, 0, 0, 0, 1)$.

The number of assemblies does not have to be an integer. Assume that $k = 0.4$ and $n = 2$. L_j becomes (3, 3, 2, 2). However, if k is greater than or equal to 1 as in most real cases, L_j is always a 0-1 list. Finally, after the feeding lists have been obtained for all parts, the part feeding synchronization is achieved by linking the lists to the assembly serial numbers.

Because of part synchronization, the closed loop system can always maintain a fixed number of assemblies or assembly parts in the line. A completed set of parts is released to the line when and only when a good assembly has just left the production line. The only controllable variable is the total number of subassemblies in the line. For a given throughput, the minimal WIP level can be found by a simple searching procedure.

• *Description of models*

There is a trade-off between the model run time and the model programming effort. One can construct a "customized" model for each individual WIP management policy or develop a single "generic" model which can be applied to all different policies. To simplify our discussion, the remainder of this section presents only the latter approach.

In our RESQ model, there are two types of FCFS queues: (1) an active queue that serves as an assembly station, a tester, or a rework station, and (2) a passive queue that represents a buffer. Since there are break times for operators, a manual operation is modeled by two active queues associated with the same passive queue. One of the active queues does not operate during the break time and becomes an inactive queue. In our model, we use the job routing mechanism (discussed later) to convert an active queue into an inactive one and vice versa. If both queues are active, the arrivals always join the one with smaller queue length. This is called "smallest queue first" discipline. The number of servers in the two active queues can be either identical (if the

number of work stations is even) or different by one (if the number is odd). The automated operations and the manual operations of a single station are never shut down, and therefore are modeled by single active queues.

For simplicity, we combine the work cycle time with the possible station down time as the service time of the queue. The interval between failures is often subject to an exponential distribution. In our model, each work cycle experiences a Bernoulli trial to determine the occurrence of failure. Thus the number of cycles between successive failures is a geometric random variable. If the work cycle time is small compared to the interval between failures, this approach will be close to the exponential assumption. Down time is composed of the waiting time for a repairman and the repair time, which can either be obtained from past machine records or estimated by the repairman model. See Oates [15] for an alternative method of modeling failures.

For each operation, a passive queue is used to regulate the WIP size. This is similar to a queueing system with a finite job waiting room. The number of tokens in the passive queue is equal to the sum of the buffer size and the number of work stations. For the push system and the closed loop system, the number of tokens can be arbitrarily large so that the line behavior is not affected by the buffer space. (These two systems may be simulated by a different model in which no passive queues exist, thereby reducing the simulation run time.) Associated with each operation there is exactly one passive queue. A token can be interpreted as a working permit. Each piece of work must own a token before its entrance to an operation (an active queue). A token must be obtained from the next operation before the current token is released. Token exchange implies a physical move from an operation to the operation immediately downstream.

In our model, identical work stations (performing the same operation) share the same buffer. This assumption may not always be valid. The assumption is justified for a process layout in which identical stations are placed together. For a product layout where the identical stations are distributed to separate areas, the assumption approximates the "smallest queue first" discipline. Any precise models must be layout dependent and cannot be generic.

The job flows follow the process flow chart (see Fig. 5) and form a closed chain. Obviously, this is exactly the case for the closed loop system. For the push system, we can choose a large job population such that the work stations of the first operation (P010) are fully utilized. For convenience, a dummy node may be used as a job reservoir which dispatches new assembly jobs and receives completed ones. The pull system and transfer line can be treated in the same fashion, except that the numbers of tokens are different. The population size is equal to the total number of tokens.

The rework flows in the assembly process constitute feedback loops. When the finite buffers are installed, a deadlock situation may arise. This problem can be resolved by either priority queueing disciplines or large buffer sizes for rework operations. In reality, a priority discipline means a more complicated shop floor control system. For a production yield close to 90% or higher, the rework flow volumes are relatively small. A small amount of additional buffer space for each rework operation serves the purpose.

Because of rework loops, outbound traffic at a node may take different routes. This feature is handled by using the job variable. For instance, a job leaving P050 may take either of two values: the value is 1 (or 2) if the job goes to P060 (or P120). Similarly, an arrival has a value equal to 2 (or 3 or 4) if it comes from P050 (or P060 or P110). The value of the job variable is assigned at set nodes.

The production yields at different assembly stages are random variables and are treated as global variables in the RESQ model. Random number generators are invoked at set nodes to assign the values of these global variables. These values are interpreted as the routing probability.

It is worthwhile to mention that random variables such as yield and assembly cycle time are considered to be independently, identically distributed. This may not be the case in real world problems. It is possible to generate a sequence of autocorrelated variables in the model. One can set up an autoregressive or a moving average model [16] at a set node to generate the yield value. To deal with autocorrelated assembly time, we can adopt the multiclass concept: An active queue may serve a number of classes of jobs. Upon its arrival, a job joins one of the classes with a certain probability. This probability can be evaluated by any model for an autocorrelated sequence.

- *Discussion of results*

An assembly line with a capacity of 500 assemblies a day has been studied by RESQ simulation. The model input includes

- number of work stations per operation,
- buffer size per operation,
- mean and coefficient of variation of each operation cycle time,
- mean time between failures,
- mean time to repair,
- duration of each break period,
- duration of lunch time,
- average yield factors,
- initial queue size at each operation,
- total population size.

In the pull system, we assume that each assembly or test operation is equipped with a four-unit buffer while each rework station has a 25-unit buffer. When the closed loop system is simulated, we have to know the total number of assemblies in the line. Three different numbers are used: 100, 150, and 200.

Confidence intervals are obtained by the independent replication method. The convergence of the simulation

Table 1 Summary of simulation results.

	Low variation	*High variation*
Average work station utilization		
Push system	0.92	0.90
Pull system	0.79	0.69
Transfer line	0.72	0.58
Closed loop (100 ASM)		0.77
(150 ASM)		0.81
(200 ASM)		0.81
Average WIP level		
Push system	1043	1044
Pull system	51	44
Transfer line	5	4
Closed loop (100 ASM)		52
(150 ASM)		99
(200 ASM)		148
Throughput (pieces per day)		
Push system	496	496
Pull system	492	426
Transfer line	453	358
Closed loop (100 ASM)		469
(150 ASM)		492
(200 ASM)		495
Average residence time (h)		
Push system	48.1	48.1
Pull system	4.4	4.5
Transfer line	2.8	2.9
Closed loop (100 ASM)		4.0
(150 ASM)		5.8
(200 ASM)		8.7

ASM—assemblies.

process is examined by comparing the width of a 90% confidence interval to its corresponding estimate. Normally we accept the simulation result if the confidence interval to mean ratio (CITMR) is no more than 15%.

It is important to point out that the results from a simulation model are highly dependent on the assembly cycle time behavior. Indeed, the existence of WIP is in part due to the cycle time fluctuations. Independent studies reported in professional journals have concluded that the manual assembly work time distribution is positively skewed and has a coefficient of variation (the ratio of standard deviation to mean) between 0.1 and 0.5 [17, 18]. Since the average cycle times estimated by engineers contain other factors such as human fatigue, material handling delay, etc., the coefficient of variation (CV) should be larger than that reported in the existing papers. In our study, two cases are investigated:

a. Low variation: $CV = 0.5$ for the push and the pull systems,
   $CV = 0.1$ for the transfer line case.
b. High variation: $CV = 2$ for all methods.

These numbers are applied to manual assembly stations only. All testers are automated stations and have $CV = 0.1$.

Line performance measures are estimated on the basis of three replications of 80 hours of simulated time (an initial eight hours of statistics in each replication is discarded to eliminate the transient stage), including work station utilization, WIP level, throughput, and WIP production lead time (amount of time that an assembly remains in the line). A summary of the simulation results is attached. The average utilization, WIP level, throughput, and average production lead time are all positively correlated. The throughput estimates should be close to the true value under the given assumptions, for the CITMR of the throughput at P090 is consistently below 3%. The numerical results are summarized in **Table 1**.

Under the push system, a large amount of WIP can quickly absorb any cycle time fluctuations. Consequently we do not detect any performance difference between the low-variation and the high-variation cases. Both cases produce 496 pieces a day. The average WIP sizes are about 1050 and the average production lead time is two days (48.06 hours).

The pull system is more sensitive to the cycle time CV, owing to the limited amount of WIP. The throughputs differ from each other by $492 - 426 = 66$. Under the high-variation assumption, the line production is significantly below the 500 daily demand.

If the transfer line concept is adopted, the throughput falls far below our requirement in the high-variation case. When $CV = 0.1$ (close to constant), the throughput is still too low due to the yield problem.

For the closed loop system, only the high-variation cycle time has been investigated. It has been found that if the total number of subassemblies is 100, the throughput is lower than the requirement. For 150 subassemblies, the line throughput is 492. If the value is increased to 200, the line can produce 495 pieces a day. Since the CITMR is 2%, the difference between the last two cases is statistically insignificant.

Now let us compare the closed loop system with the pull system under the same cycle time variation. It can be seen that for a slightly higher WIP level (52 vs 44), the closed loop system has a smaller average production lead time (4.0 hours vs 4.5 hours) and 10% higher throughput (426 vs 469). If the WIP level is increased to 99 (the total number of subassemblies is 150), the closed loop system can produce $492 - 426 = 66$ more products each day and the average production lead time is $5.8 - 4.5 = 1.3$ hours longer as compared with the pull system.

Conclusion

The RESQ modeling elements, solution methods, and performance measures have been introduced. The modeling elements allow us to represent the behavior of complicated manufacturing systems. The solution methods produce accurate performance measures which can be used to predict the operation of a system.

Four different WIP management policies have been studied under the given assumptions. Although only the estimated average values are reported in the paper, the simulation programs are capable of collecting the variances and even the statistical distributions.

The push system is frequently used to achieve a high tool utilization, but is notorious for its large WIP size. One must carefully plan for the initial parts inventory and the parts delivery lead time. The example in Table 1 suggests that the studied assembly line under the push system has too much WIP. The major problem here is how to reduce the lead time for parts delivery.

The pull system with limited buffer sizes is very sensitive to the cycle time variations. An intensive study in cycle time behavior is a necessary step in the analysis of a continuous flow manufacturing system. The major design problem in a pull system is the buffer allocation. Unfortunately, no efficient analytical algorithm is available [19].

It does not seem likely that an efficient transfer line could be used for the studied assembly. The complexity of the product leads to nonuniform cycle times, while the manual operations create a large cycle time variation. For the low-variation case, we have assumed $CV = 0.1$. But the system can only produce 453 pieces a day and the average work station utilization is as low as 0.72. This implies that even if the cycle times are all constant, we still have to face the yield problem.

The closed loop system is an interesting case for its simplicity and efficiency. This system has only one variable, that is, the total number of subassemblies. The optimal solution can be obtained by a simple search procedure. This optimal solution is a function of cycle time variations. Under the optimality condition, if all subassemblies are "evenly" distributed over the assembly line, the production line is running as a continuous flow. On the other hand, if congestion occurs at a certain operation, any additional parts supply simply creates an even worse traffic problem and little improvement on the line throughput can be expected.

Future research

Since almost all the manufacturing problems are linked together, this study is by no means complete. Three immediate problems are identified: parts feeding, machine reliability, and material handling. It has been shown by simulating a closed loop system that parts metering can be an effective approach to controlling the WIP level. We should extend this study to explore the best parts feeding method.

Although machine reliability has been explicitly modeled in the current study, the model run length of each case is never long enough to reflect any "short-term" impact caused by the reliability problem. (If the model predicts a throughput of 495, it merely says that the long run average is 495.) For instance, a line consists of three machines for P050, each with a mean time between failures of about 500 hours. In order to observe ten failures, we have to simulate $500 \times 10/3$ or 1667 hours. When the RESQ model is evaluated, it takes about five hours of CPU time on an IBM 4341 machine. Since RESQ is a PL/I-based solution package, it is unlikely that the model run time will decrease significantly if other simulation languages are used. Obviously this prevents us from conducting even a moderate number of experiments. To overcome this difficulty, a hybrid hierarchical modeling approach may be helpful (i.e., using an analytical model to drive a simulation submodel).

At the early stage of the line study, one often assumes an adequate material handling system. This, however, may not always be the case. When a conveyor system is used, traffic blockage may occur due to congestion or deficient accumulation area on the conveyor line. On the other hand, if a handler [20] is adopted, a poor layout may result in an investment for extra handlers. An integrated study of line configuration, material handling system, reliability, parts feeding, and WIP management should be carried out.

Appendix

The following is a description of the RESQ model containing the passive queues representing the buffers. Numeric parameters are symbolic names which are assigned values when the model is evaluated. Numeric parameters can be scalars, vectors, or matrices. The numeric parameters defined here are used for the number of work stations, the mean and coefficient of variation of service time, buffer sizes, probabilities in discrete distributions, variables in routing decisions, initial populations, the chain population, the number of replications, limits for each replication, the total solution time, the initial portion discarded from each replication, and the initial seed for the random number generator.

```
MODEL:WIP
      /* study of wip management policies for an assembly line */
      /* stations 050 060 and 090 are inspection or test operations */
      /* the rest are assembly or rework stations */
      /* half of the assembly stations are closed during break */
      /* buffers are provided between operations */
      /* yield factors are random numbers */
      /* work cycle time is characterized by mean and variance */
      /* NWS = no of work stations */
      /* MCT = mean cycle time */
      /* CVT = coeff. of variation of cycle time */
      /* BSIZE = buffer size */
      /* FRT = failure rate per cycle */
      /* FTM = mean failure time */
      /* FCV = coeff. of variation of failure time */
      /* BK_T  LH_T = break time and lunch time */
      /* B5 B6 B9 = yield factors */
      /* B11 B12 = branching probabilities */
      /* INQ INQ1 OUTQ = initial queue sizes */
      /* POPSIZE = population size */
  METHOD:SIMULATION
  NUMERIC PARAMETERS:NWS(13) MCT(13) CVT(13) BSIZE(13) FRT(13)
  NUMERIC PARAMETERS:FTM(13) FCV(13) BK_T LH_T B5 B6 B9 B11 B12
  NUMERIC PARAMETERS:INQ(13) INQ1(13) OUTQ POPSIZE
  NUMERIC PARAMETERS:NREP TSIM NEVT D CNT TCPU PCT_D SD
      /* NWS = N+N1  THE LATTER IS CLOSED FOR BREAK  */
```

Numeric identifiers are symbolic names which are assigned values in the model construction phase. They are constants which can be scalars, vectors, or matrices. Their values can be numbers or arithmetic expressions.

```
NUMERIC IDENTIFIERS: N(13) N1(13)
   N:  CEIL(NWS(1)/2) CEIL(NWS(2)/2) CEIL(NWS(3)/2) CEIL(NWS(4)/2) ++
       CEIL(NWS(5)/2) CEIL(NWS(6)/2) CEIL(NWS(7)/2) CEIL(NWS(8)/2) ++
       CEIL(NWS(9)/2) CEIL(NWS(10)/2) CEIL(NWS(11)/2)              ++
       CEIL(NWS(12)/2) CEIL(NWS(13)/2)
   N1: NWS(1)-N(1) NWS(2)-N(2) NWS(3)-N(3) NWS(4)-N(4)          ++
       NWS(5)-N(5) NWS(6)-N(6) NWS(7)-N(7) NWS(8)-N(8)          ++
       NWS(9)-N(9) NWS(10)-N(10) NWS(11)-N(11)                   ++
       NWS(12)-N(12) NWS(13)-N(13)
```

Global variables are symbolic names which are assigned initial values. Global variables can be assigned new values as the simulation is running, and their current values can be checked to make routing decisions. BK and LH are used in routing decisions related to break and lunch times. BR5, BR6, and BR9 are branching probabilities. CLOCK is the simulation clock.

```
GLOBAL VARIABLES: BK LH BR5 BR6 BR9 CLOCK
    BK: 1
    LH: 1
   BR5: B5
   BR6: B6
   BR9: B9
   CLOCK: 0
```

Queue types are parameterized versions of queues. They can contain numeric parameters to permit queues with values to be invoked from a common definition. They contain node parameters to assign actual node names. The following two queue types are for the work stations and the buffers. The numeric parameters in the queue type for the work stations represent the number of servers and values used in the work demand distribution expression. The node parameter is used as the class at the active queue. The numeric parameter in the buffer passive-queue-type definition is used to specify the number of tokens. The node parameters are used as allocate and release node names.

```
   /* WORK STATION QUEUES */
QUEUE TYPE: WSQ
   NUMERIC PARAMETER: NO_WS M_CT CV_T F_RT F_TM F_CV
   NODE PARAMETER: WS_TYPE
   TYPE: ACTIVE
   SERVERS: NO_WS
   DSPL: FCFS
   CLASS LIST: WS_TYPE
     WORK DEMANDS: DISCRETE(0,1-F_RT;1,F_RT)*STANDARD(F_TM,F_CV)  ++
           + 0.8*M_CT*STANDARD(0.2*M_CT,5*CV_T)
END OF QUEUE TYPE WSQ
    /* BUFFER QUEUES */
QUEUE TYPE: BFQ
   NUMERIC PARAMETER: B_SIZE
   NODE PARAMETER: GET_BF REL_BF
   TYPE: PASSIVE
   TOKENS: B_SIZE
   DSPL:FCFS
   ALLOCATE NODE LIST: GET_BF
      NUMBER OF TOKENS TO ALLOCATE: 1
   RELEASE NODE LIST: REL_BF
END OF QUEUE TYPE BFQ
```

All of the actual queue definitions in this model are based on the previous two queue types. The parameter values are assigned to the numeric and node parameters defined in the queue type definitions in the order in which they were defined above. The following queue definitions are based on the WSQ queue type. They represent the work station queues.

```
    /* INVOKE WORK STATION QUEUES */
QUEUE: P010Q
  TYPE: WSQ: N(1);MCT(1);CVT(1);FRT(1);FTM(1);FCV(1);P010
QUEUE: P011Q
  TYPE: WSQ: N1(1);MCT(1);CVT(1);FRT(1);FTM(1);FCV(1);P011
          .
          .
          .
QUEUE: P040Q
  TYPE: WSQ: N(4);MCT(4);CVT(4);FRT(4);FTM(4);FCV(4);P040
QUEUE: P041Q
  TYPE: WSQ: N1(4);MCT(4);CVT(4);FRT(4);FTM(4);FCV(4);P041
QUEUE: P050Q
  TYPE: WSQ: NWS(5);MCT(5);CVT(5);FRT(5);FTM(5);FCV(5);P050
QUEUE: P060Q
  TYPE: WSQ: NWS(6);MCT(6);CVT(6);FRT(6);FTM(6);FCV(6);P060
QUEUE: P070Q
  TYPE: WSQ: N(7);MCT(7);CVT(7);FRT(7);FTM(7);FCV(7);P070
QUEUE: P071Q
  TYPE: WSQ: N1(7);MCT(7);CVT(7);FRT(7);FTM(7);FCV(7);P071
QUEUE: P080Q
  TYPE: WSQ: N(8);MCT(8);CVT(8);FRT(8);FTM(8);FCV(8);P080
QUEUE: P081Q
  TYPE: WSQ: N1(8);MCT(8);CVT(8);FRT(8);FTM(8);FCV(8);P081
QUEUE: P090Q
  TYPE: WSQ: NWS(9);MCT(9);CVT(9);FRT(9);FTM(9);FCV(9);P090
          .
          .
          .
QUEUE: P130Q
  TYPE: WSQ: NWS(13);MCT(13);CVT(13);FRT(13);FTM(13);FCV(13);P130
```

The following queue definitions are based on the BFQ queue type definition. They represent the buffer queues.

```
    /* INVOKE BUFFER QUEUES */
QUEUE: P010B
  TYPE: BFQ: BSIZE(1);G010;R010
          .
          .
          .
QUEUE: P130B
  TYPE: BFQ: BSIZE(13);G130;R130
```

Set nodes permit the execution of one or more assignment statements. The following set node definitions assign values to job variables and global variables. Arithmetic assignments can be used in the assignments, as well as samples from probability distributions.

```
SET NODES: SETJV
  ASSIGNMENT LIST: JV(0)=1                                    ++
                   BK=((0.5+CLOCK)/2)-FLOOR((0.5+CLOCK)/2)  ++
                   LH=((4.25+CLOCK)/8)-FLOOR((4.25+CLOCK)/8)
SET NODES: RESET1 RESET2 RESET3
  ASSIGNMENT LIST: JV(0)=1 JV(0)=1 JV(0)=1
SET NODES: SET050
  ASSIGNMENT LIST: JV(0)=2
SET NODES: SET060
  ASSIGNMENT LIST: JV(0)=3
SET NODES: SET090
  ASSIGNMENT LIST: JV(0)=4
SET NODES: SET110
  ASSIGNMENT LIST: JV(0)=5
SET NODES: SET120
  ASSIGNMENT LIST: JV(0)=6
SET NODES: SET121
  ASSIGNMENT LIST: JV(0)=7
SET NODES: P050BR
  ASSIGNMENT LIST: BR5=UNIFORM(B5*0.9,B5,.33,B5,B5*1.05,.67)
SET NODES: P060BR
  ASSIGNMENT LIST: BR6=UNIFORM(B6*0.9,B6,.33,B6,B6*1.05,.67)
SET NODES: P090BR
  ASSIGNMENT LIST: BR9=UNIFORM(B9*0.9,B9,.33,B9,B9*1.05,.67)
DUMMY NODES: JOBRSVR
```

The chain definition in this model is a closed chain with a fixed population size specified by a numeric parameter. The routing statements show how the jobs move through the nodes of the model. Logical tests given in IF predicates are used for making routing decisions.

```
   /* FLOW PATH */
CHAIN: JOBFLOW
  TYPE: CLOSED
  POPULATION: POPSIZE
  :JOBRSVR->SETJV->G010
  :G010->P010 P011;IF(QL(P011)*N(1)>=QL(P010)*N1(1) OR LH<LH_T   ++
                                                  OR BK<BK_T) IF(T)
  :P010 P011->G020->R130 R010;IF(JV(0)=7) IF(T)
  :R130->P020 P021;IF(QL(P021)*N(2)>=QL(P020)*N1(2) OR LH<LH_T   ++
                                                  OR BK<BK_T) IF(T)
  :R010->P020 P021;IF(QL(P021)*N(2)>=QL(P020)*N1(2) OR LH<LH_T   ++
                                                  OR BK<BK_T) IF(T)
  :P020 P021->RESET1->G030->R020
  :R020->P030 P031;IF(QL(P031)*N(3)>=QL(P030)*N1(3) OR LH<LH_T   ++
                                                  OR BK<BK_T) IF(T)
  :P030 P031->G040->R120 R030;IF(JV(0)=6) IF(T)
  :R030->P040 P041;IF(QL(P041)*N(4)>=QL(P040)*N1(4) OR LH<LH_T   ++
                                                  OR BK<BK_T) IF(T)
  :P040 P041->RESET2->G050->R040->P050->P050BR
  :P050BR->G060 SET050;BR5 (1-BR5)
  :SET050->G120        /* RWK FROM P050 */
  :G060->R050->P120 P060;IF(JV(0)=2) IF(T)
  :P060->P060BR->G070 SET060;BR6 (1-BR6)
  :SET060->G120        /* RWK FROM P060 */
  :G070->R110 R060;IF(JV(0)=5) IF(T)
  :R060->P120;IF(JV(0)=3)
  :R060->P070 P071;IF(QL(P071)*N(7)>=QL(P070)*N1(7) OR LH<LH_T   ++
                                                  OR BK<BK_T) IF(T)
  :P070 P071->RESET3->G080->R070
  :R070->P080 P081;IF(QL(P081)*N(8)>=QL(P080)*N1(8) OR LH<LH_T   ++
                                                  OR BK<BK_T) IF(T)
  :P080 P081->G090->R080->P090
  :P090->P090BR->R090 SET090;BR9 (1-BR9)
  :SET090->G100->R090
  :R090->P100 JOBRSVR;IF(JV(0)=4) IF (T)    /* JOB COMPLETED */
  :P100->G110->R100->P110->SET110 G120;B11 (1-B11)
  :SET110->G070
  :G120->R110 R060 R050;IF(JV(0)=4) IF(JV(0)=3) IF(T)
  :R110->P120 P070;IF(JV(0)=4) IF(T)
  :P120->SET121 SET120;B12 (1-B12)
  :SET120->G040
  :SET121->G130->R120->P130 P040;IF(JV(0)=7) IF(T)
  :P130->G020
```

The distribution of the tokens in use is an optional performance measure which must be specified for those passive queues of interest. In this model all possible values related to token use are being requested for all buffer queues.

```
QUEUES FOR TOKEN USE DIST: P010B P020B P030B P040B P050B P060B ++
                           P070B P080B P090B P100B P110B P120B P130B
  MAX VALUE: BSIZE(1)
        .
        .
        .
  MAX VALUE: BSIZE(13)
```

The method of independent replications is used for generating confidence intervals. Jobs are initialized at the specified nodes at the beginning of each replication. Ninety percent confidence intervals are produced. Numeric parameters are used to define the number of replications, the percent discarded at the beginning of each replication, various replication limits, maximum run time, and the random number seed.

```
   CONFIDENCE INTERVAL METHOD:replications
   INITIAL STATE DEFINITION -
      CHAIN: JOBFLOW
      NODE LIST: P010 P020 P030 P040 P050 P060 P070 P080 P090 ++
                 P100 P110 P120 P130                        ++
                 P011 P021 P031 P041 P071 P081 JOBRSVR
      INIT POP: INQ(1) INQ(2) INQ(3) INQ(4) INQ(5) INQ(6) INQ(7)++
                INQ(8) INQ(9) INQ(10) INQ(11) INQ(12) INQ(13)   ++
                INQ1(1) INQ1(2) INQ1(3) INQ1(4) INQ1(7) INQ1(8) OUTQ
   CONFIDENCE LEVEL:90
   NUMBER OF REPLICATIONS:NREP
   INITIAL PORTION DISCARDED: PCT_D
   REPLIC LIMITS -
      SIMULATED TIME:TSIM
      EVENTS:NEVT
      QUEUES FOR DEPARTURE COUNTS: P090Q
         DEPARTURES: D_CNT
   LIMIT - CP SECOND:TCPU
   SEED:SD
   TRACE:NO
END
```


Acknowledgments

Some of the information contained in Sections 1 and 2 is based on [3]. We wish to thank all the people who have been involved in the development of RESQ over the years and the users who have helped guide its progress. We are also grateful to the anonymous reviewers for their helpful suggestions.

References

1. H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology*, Addison-Wesley Publishing Co., Inc., Reading, MA, 1978.
2. *Computer Performance Modeling Handbook*, S. S. Lavenberg, Ed., Academic Press, Inc., New York, 1983.
3. E. A. MacNair and C. H. Sauer, *Elements of Practical Performance Modeling*, Prentice-Hall, Inc., Englewood Cliffs, NJ, in press.
4. C. H. Sauer and K. M. Chandy, *Computer System Performance Modeling*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.
5. C. H. Sauer and E. A. MacNair, "The Research Queueing Package Version 2: Availability Notice," *Research Report RA-144*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, August 1982.
6. C. H. Sauer and E. A. MacNair, *Simulation of Computer Communication Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.
7. C. H. Sauer, E. A. MacNair, and J. F. Kurose, "The Research Queueing Package Version 2: Introduction and Examples," *Research Report RA-138*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, April 1982.
8. C. H. Sauer, E. A. MacNair, and J. F. Kurose, "The Research Queueing Package Version 2: CMS Users Guide," *Research Report RA-139*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, April 1982.
9. C. H. Sauer, E. A. MacNair, and J. F. Kurose, "The Research Queueing Package Version 2: TSO Users Guide," *Research Report RA-140*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, April 1982.
10. Charles H. Sauer, Edward A. MacNair, and Silvio Salza, "A Language for Extended Queuing Network Models," *IBM J. Res. Develop.* **24**, 747–755 (November 1980).
11. D. R. Cox and W. L. Smith, *Queues*, Methuen & Co. Ltd., London, 1961.
12. J. Orlicky, *Material Requirement Planning*, McGraw-Hill Book Co., Inc., New York, 1975.
13. R. W. Hall, *Production Planning and Control in Japan*, Ch. 4, American Production and Inventory Control Society, Falls Church, VA, 1981.
14. H. D. Friedman, "Reduction Methods for Tandem Queueing Systems," *Oper. Res.* **13**, 121–131 (1965).
15. W. J. Oates, "Manufacturing Modeling Using RESQ," *Proceedings of the Winter Simulation Conference*, Dallas, TX, November 1984, pp. 357–359.
16. G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*, Holden-Day Publishing Co., Inc., San Francisco, 1970.
17. G. M. Buxey and D. Sadjadi, "Simulation Studies of Conveyor-Paced Assembly Lines with Buffer Capacity," *Int. J. Prod. Res.* **14**, 607–624 (1976).
18. N. A. Dudley, "Work-Time Distributions," *Int. J. Prod. Res.* **2**, 137–144 (1963).
19. Y. C. Ho, M. A. Eyler, and T. T. Chien, "A Gradient Technique for General Buffer Storage Design in a Production Line," *Int. J. Prod. Res.* **17**, 557–580 (1979).
20. W. Chow, "An Analysis of Automated Storage and Retrieval Systems in Manufacturing Assembly Lines," *Technical Report TR 02.1082*, IBM General Products Division, San Jose, CA, November 1983.

Received October 18, 1984; revised January 23, 1985

We-Min Chow *IBM General Products Division, 5600 Cottle Road, San Jose, California 95193.* Dr. Chow is a member of the advanced manufacturing engineering group in San Jose. He received a B.S. degree in industrial management from Cheng Kung University, Taiwan, China, and a Ph.D. in operations research from the University of California, Berkeley. From 1973 to 1982 he was a research staff member at the IBM Thomas J. Watson Research Center, where he conducted his research activities in computer systems analysis and algorithms. Dr. Chow's current interests include design optimization of manufacturing lines, analysis of assembly processes and material handling subsystems, and line operation management.

Edward A. MacNair *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. MacNair joined IBM in 1965. He has been a member of the research staff in the Computer Science Department at the IBM Thomas J. Watson Research Center since 1973. He is currently in the modeling and analysis software systems group, developing modeling programs to solve extended queueing networks. In addition, he is an adjunct staff member at the IBM Systems Research Institute, teaching a course related to performance modeling. He received a B.A. degree in mathematics in 1965 from Hofstra University and an M.S. in operations research in 1972 from New York University. Mr. MacNair is a member of the Association for Computing Machinery and the Operations Research Society of America. He received a Research Division Outstanding Contribution Award for his involvement with the Research Queueing Package, a tool for the solution of generalized queueing networks.

Charles H. Sauer *IBM Engineering Systems Products, 11400 Burnet Road, Austin, Texas 78758.* Dr. Sauer received his B.A. in mathematics and his Ph.D. in computer sciences from the University of Texas at Austin in 1970 and 1975, respectively. He joined IBM at the Thomas J. Watson Research Center in 1975. From 1977 to 1979 he was an assistant professor of computer sciences at the University of Texas at Austin. In 1979 he returned to the Watson Research Center and in 1982 joined the IBM Communication Products Division Laboratory in Austin. He is currently manager of system design in the area of advanced information products. Dr. Sauer has published two textbooks, *Computer System Performance Modeling,* co-authored by K. M. Chandy, and *Simulation of Computer Communication Systems,* co-authored by E. A. MacNair. Dr. Sauer received an IBM Outstanding Innovation Award for creation and basic design of the Research Queueing Package (RESQ). He is a member of the Association for Computing Machinery.