Process Improvement Principles

A Concise Guide for Managers

C. Dennis Pegden

Published by Simio LLC

www.simio.com

Process Improvement
A Concise Guide for Managers

First Edition
Last updated: June 9, 2015

Published by Simio LLC, 504 Beaver St, Sewickley, PA 15143

A free evaluation version of the Simio software used in this book can be downloaded at no charge from www.simio.com.

The models used to illustrate the performance improvement principles can be run in evaluation software and can be found under the Examples folder on the Support ribbon.

Table of Contents

Introduction

This book presents 25 process improvement principles that can be used by managers to improve the design and operation of their production systems. Both manufacturing and service systems can take advantage of time and cost savings offered by these principles.

A production system uses one or more resources to perform a specific task or set of tasks on an item as it moves through the work system. In a manufacturing setting, the resources might include machines, tools, operators, and material handling devices. In a service system--such as a health clinic--the resources might include a waiting area, doctors and nurses, imaging equipment, and exam rooms. In any case, the flow of work through the system depends on the availability of resources. .

The word *entity* denotes the work item that moves through the resources of the system. An entity might be a part that we are manufacturing, a patient that we are treating in an emergency department, a passenger that we are processing at an airport, or a business transaction that we are executing.

Our objective is to design and run production systems that most efficiently process entities using the resources that we have allocated to the system. We want our entities to be processed with least cost and in a timely manner.

Simulation gives us the ability to model a wide range of production systems. Because a simulation model can "act out" the movement of entities through the system, it can capture the impact of limited resources on the system performance. It has the great advantages of speed and flexibility in identifying variability within processes. Watching animations of our systems provides insights into the system behavior. We can quickly evaluate many different system alternatives, and compare key performance indicators (KPIs) to select the best strategies.

Although we will use many different simulation models in our analysis, there is no need for the reader to be skilled in building their own simulation models. We will use Simio to illustrate the principles we discuss. Although Simio supports realistic 3D animated models, our focus here is on quantitative results and not visualization; hence, we will restrict our use of Simio to basic 2D visualization. For those readers interested in running the models that are used in this book, they may be executed using the free evaluation version of Simio and can be found via the Support ribbon in the Examples folder.

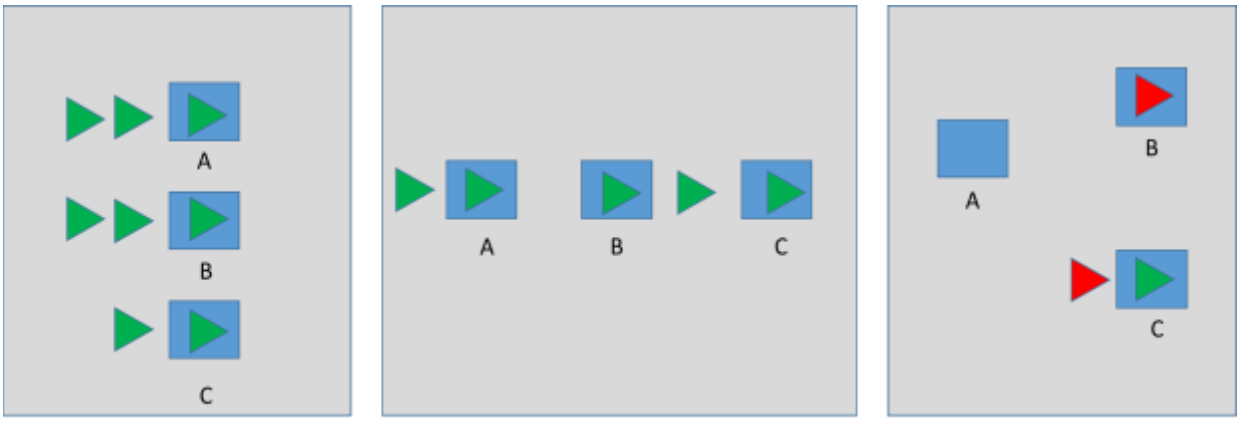
Simio simulation can be used to assess many processing systems that we encountered in daily business life. We will consider three primary types of production systems, as illustrated below.

Parallel Servers. The most basic form is a stand-alone service area, where one or more servers process entities in parallel. In the example on the left, three servers (blue rectangles) operate in parallel to process arriving entities (green triangles). An example of this is an airline check-in counter where each passenger is processed by a check-in clerk.

Flow Line. We will also consider a linear sequence of servers where each entity moves from server to server along the fixed sequence. In a flow line, entities move in a fixed sequence from server A, to server B, and then to server C. Although each entity may have a different processing time at each

server, all entities follow the same sequence (A, B, C). An example of this is a production line in a manufacturing facility or a processing center for getting your driver's license. Sometimes flow lines are paced where all entities must progress in unison; an example of this is an automotive assembly line.

Job Shop. Finally, we consider the more general case where each entity follows its own sequence through the system, and may visit any subset of the servers. In the example on the right, the red entities visit server A then C, then B, and the green entities visit C and then B. This is the classic job shop manufacturing system, but also is representative of patients flowing through a hospital. Note that all these systems may have a primary resource for each server (such as a machine or exam room), as well as secondary resources such as workers, tools, etc., that are either dedicated to a specific server or are shared between multiple servers.



Parallel Servers Flow Line Job Shop

In addition to the basic structure of the system we have set up, we also must make decisions about how to size and operate these systems. In a flow line or job shop, we typically have to allocate limited input and/or output buffer space to servers in order to avoid excessive starvation and blocking between servers. What strategy should we use in allocating this limited buffer space to servers? In a flow line, we also may have choices on the order in which we place our servers in the flow line. Should we put our fastest servers at the front, middle, or end of the flow line? When operating the system, we may have choices on which entity we process next, or which server we assign to perform a specific task. If we want to maximize our throughput, what entity should we select to process next? In the case of secondary resources such as operators, we have decisions to make in terms of how much cross training to employ for sharing operators across multiple servers. The production principles presented here are targeted at helping to make these decisions to improve the design and operation of these systems.

To illustrate our production principles, we will focus on three KPIs—Work in Process (WIP), system throughput, and on-time delivery. WIP refers to work that has entered the system, but is not yet complete. WIP can be the number of items in the system, or converted to time-based measures of congestion such as average waiting time or time in system using Little's Law. System throughput is

number of entities completed or moving through the system, and timely delivery is the completion of product according to parameters set by customer considerations.

The relevance of each of these KPIs depends upon our environment; in a make-to-stock manufacturing environment, we are typically focused on throughput, whereas in a make-to-order manufacturing environment, timely delivery is paramount. A production strategy that reduces WIP typically also reduces waiting time for each process and total time in system. In a make-to-stock environment, our primary KPI is throughput because this increases the number of products produced or customers served for a given investment in resources; decreasing WIP without compromising throughput is a secondary KPI. In a make-to-order environment, throughput is an input (i.e. produce all committed orders) and our primary focus is on timely delivery because this increases demand for our product; reducing WIP without compromising on time delivery is a secondary KPI.

Each of the process improvement principles we discuss will target one or more of the three KPIs. Although we may like to simultaneously improve all three KPIs, this is not always possible, and tradeoffs may be required. For example, a production strategy such as minimizing setup times to maximize throughput may perform poorly in terms of on-time delivery. However, the one overriding principle that does target improving all three KPIs is reduction of variation in the production process.

This book does not propose or present a process improvement methodology such as Six Sigma or Lean. Instead it presents and illustrates basic principles for process improvement that can be used with any methodology within your process improvement projects. For example within a Six Sigma project methodology involving the five phases of Define, Measure, Analyze, Improve, and Control, simulation can help with all these phases.

This book is targeted at busy managers and therefore is intentionally short in length. The goal of this book is to concisely convey the process improvement principles in as few words as possible. Let's begin.

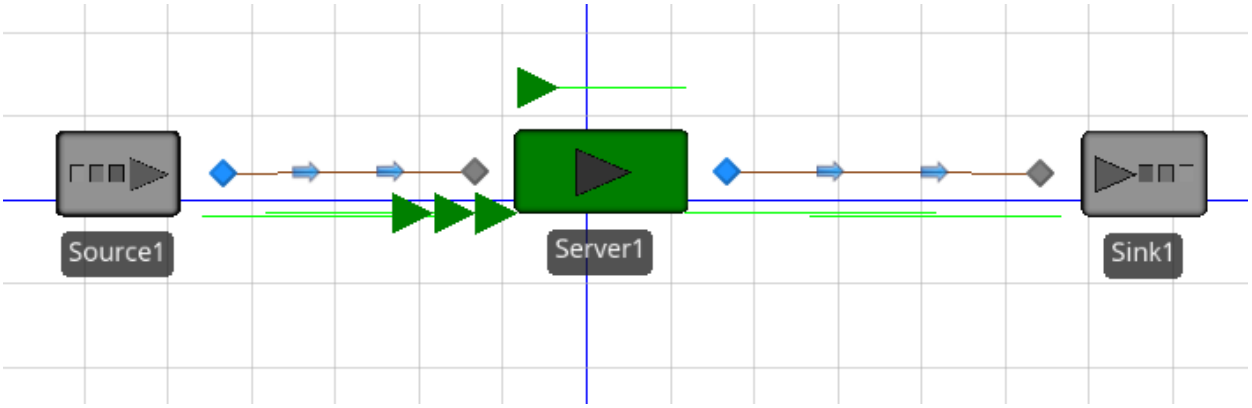
Performance Improvement Principles

Principle #1: Variation degrades performance.

Many managers underestimate the degrading impact that variation has on system performance. Variation is the single most important driver of system behavior. It is the primary reason projects have cost overruns, factories have late orders, and emergency departments have long wait times. Understanding and managing variation is vital to the proper design and running of production systems.

To explore this principle, let's consider a model of a simple production system in which a single server processes entities. Each entity arrives to the system, waits its turn in the input queue (i.e. waiting area) to be processed by the server, is processed, and then departs the system. The entities arrive to the system every hour, and the processing time is 55 minutes. Let's consider the performance of this system, and the role that variation plays.

The figure below shows a snapshot of the animated Simio simulation model of this system. It is comprised of a Source where entities enter the model, travel along the inbound Connector (indicated by blue arrows between the blue and gray diamonds) to the Server, and then travel along the outbound Connector to the Sink. The Source, Connectors, Server and Sink are referred to as objects in Simio, and are used to model the flow of entities through this simple system. In the snapshot below, the Server is green to indicate that it is busy, and the triangular green symbol above the Server is the entity being processed. The three green triangular symbols to the left of the Server are the entities queued up for processing. These objects have properties that control their behavior: that is, the Source has properties that specify the arrival pattern, and the Server has properties that specify the input buffer size and processing time.



We will run three scenarios with this model. In the first scenario, we assume no variation in the system. Entities will arrive every hour, and processing will take exactly 55 minutes. In the second scenario, we keep the processing time constant at 55 minutes, but add variation to the arrival pattern. The average time between arrivals remains one hour, but the arrivals occur randomly by selecting the time between arrivals as a random sample from an exponential distribution with a mean of one hour. In the third scenario, variability will be added to the processing time by selecting this time as a sample from an exponential distribution with a mean of 55 minutes.

The results for this simple model are summarized in the following table, where we have made 100 replications of the simulation for a period of 1024 hours for each replication. There are two controls that define the inter-arrival time to the system and the processing time at the server. The responses include the average throughput, entity waiting time at the server, and utilization of the server.

Scenario		Replications	Controls		Responses		
☑	Name	Completed	InterarrivalTime (Minutes)	ProcessingTime (Minutes)	ThroughPut	WaitingTime (Hours)	Utilization
☑	ConstantConstant	100 of 100	60	55	1024	0	0.916667
☑	RandomConstant	100 of 100	Random.Exponential(60)	55	1018.44	4.57785	0.912119
☑	RandomRandom	100 of 100	Random.Exponential(60)	Random.Exponential(55)	1014.55	9.39315	0.913334

For the first scenario (ConstantConstant) with a constant time between arrivals of 60 minutes and constant processing time of 55 minutes, we have no waiting time and server utilization of 92%. This system performs great. However, when we add variation into our arrival pattern (RandomConstant), we see that the performance degrades substantially, with the average waiting time jumping from 0 to 4.6 hours. If we also add variation into our processing time (RandomRandom), the waiting degrades further, doubling to 9.4 hours. If this were a medical emergency department that we were managing, we would have very unhappy patients. .

This simple production system illustrates the degrading impact of variation on our performance measures.

The action item from this process improvement principle is simple:

Reducing variation in the system might mean doing one or more of the following:

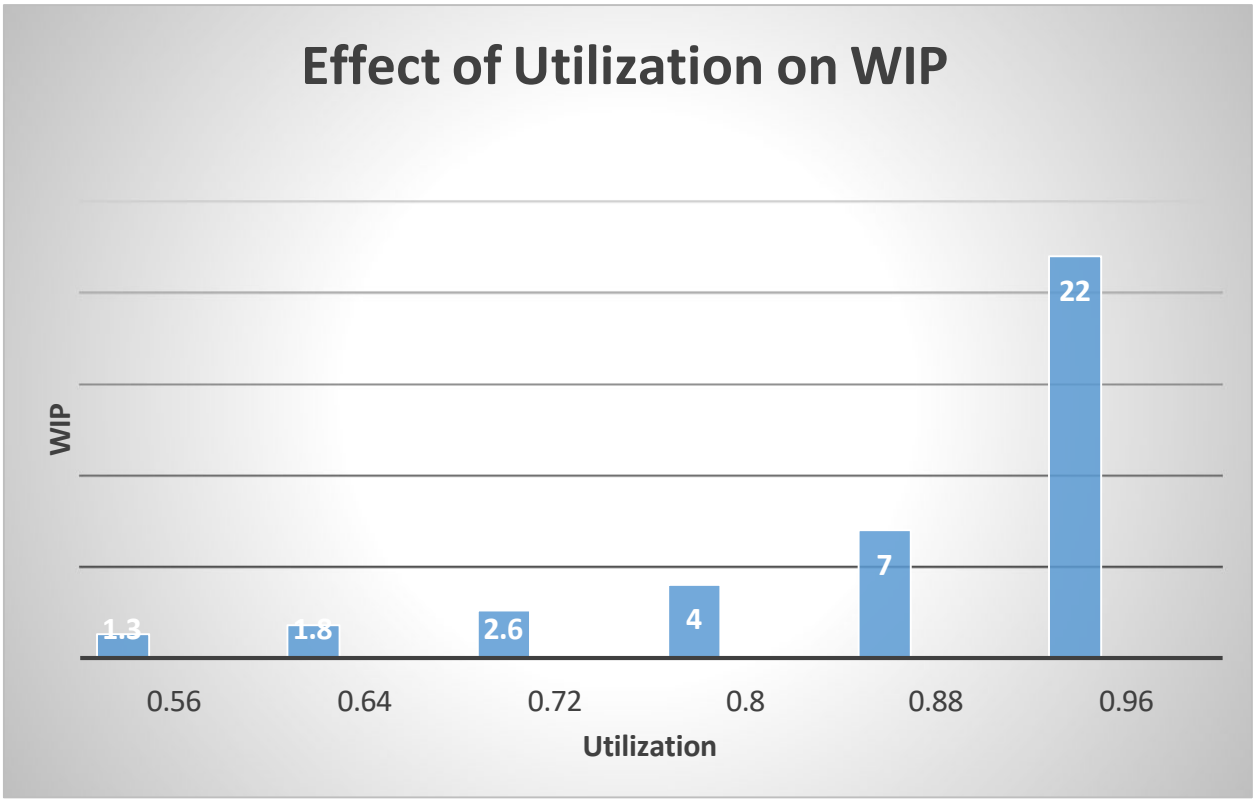
1. Design task aids (e.g. jigs/fixtures, checklists, or standardized procedures) to reduce cycle time and variability for highly variable tasks.
2. Redesign a part to make it less complex to manufacture.
3. Eliminate marginally profitable products or customers from the mix that require highly variable tasks.
4. Add new equipment to automate certain highly variable tasks.
5. Reduce the number of product variations or options.
6. Reduce variability of demand.

As we will see, a number of the following principles also target reducing variability as a way to improve performance.

Principle #2: Increasing utilization increases WIP/Waiting Times.

In our first model, our average utilization remains essentially the same for all three scenarios, with the server being idle about 8% of the time. Even with a 9.4-hour wait time for our third scenario (random inter-arrival times, random processing times), our server is idle 8% of the time. Although variation has a direct impact on performance indicators such as waiting time, cycle time, work in process, queue length, etc., it does not have a significant impact on server utilization. In many systems, the long-term utilization of servers is determined by the entities that enter the system and not the specific policies for operating the system.

Let's say as a manager you would like to increase utilization of the server to 95%. Our second principle states that in the presence of variation, increasing utilization will increase WIP, and therefore wait times at the server. We can increase server utilization to approximately 95% by changing our mean time between arrivals to 57 minutes. If we make this change and rerun our model, we see that the average waiting time increases to more than 15 hours. Hence, a slight increase in utilization from 92% to 95% causes a dramatic increase in wait times. Any further increase in utilization will have an even more dramatic impact on waiting times. In fact, as utilization approaches 100%, WIP and waiting time will approach infinity, as illustrated by the following graph of simulation results plotted for utilizations ranging from 56% to 96%.

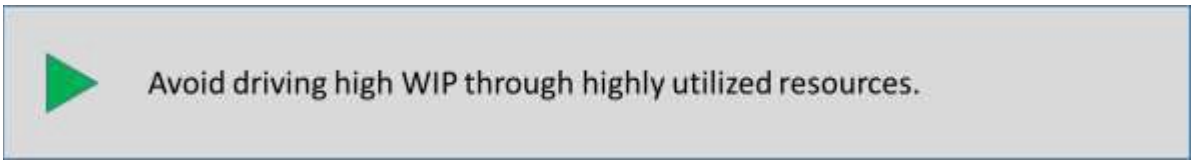


Note that the large WIP and associated wait times (shown in white on the bars) that occur with high utilization is the price paid for having variation in the system, and any attempt to have highly utilized

resources in the presence of variation will result in large wait times, large queues, and significant WIP. We can only eliminate the large wait times for highly utilized resources by removing variation from the system.

Managers often strive for high utilization of resources as a primary measure of performance. However in the presence of variation, a high utilization drives WIP (and therefore wait times) to extreme values. In real systems, managers are often forced to accept utilizations of 80% or less in order to achieve other critical KPIs for the system. For this reason we have elected to focus on throughput, timely delivery, and WIP as our primary KPIs in analyzing our performance improvement principles. In make-to-stock environments, we will focus on throughput and WIP, and in make-to-order environments we will focus on timely delivery and WIP.

The action item from this principle is:

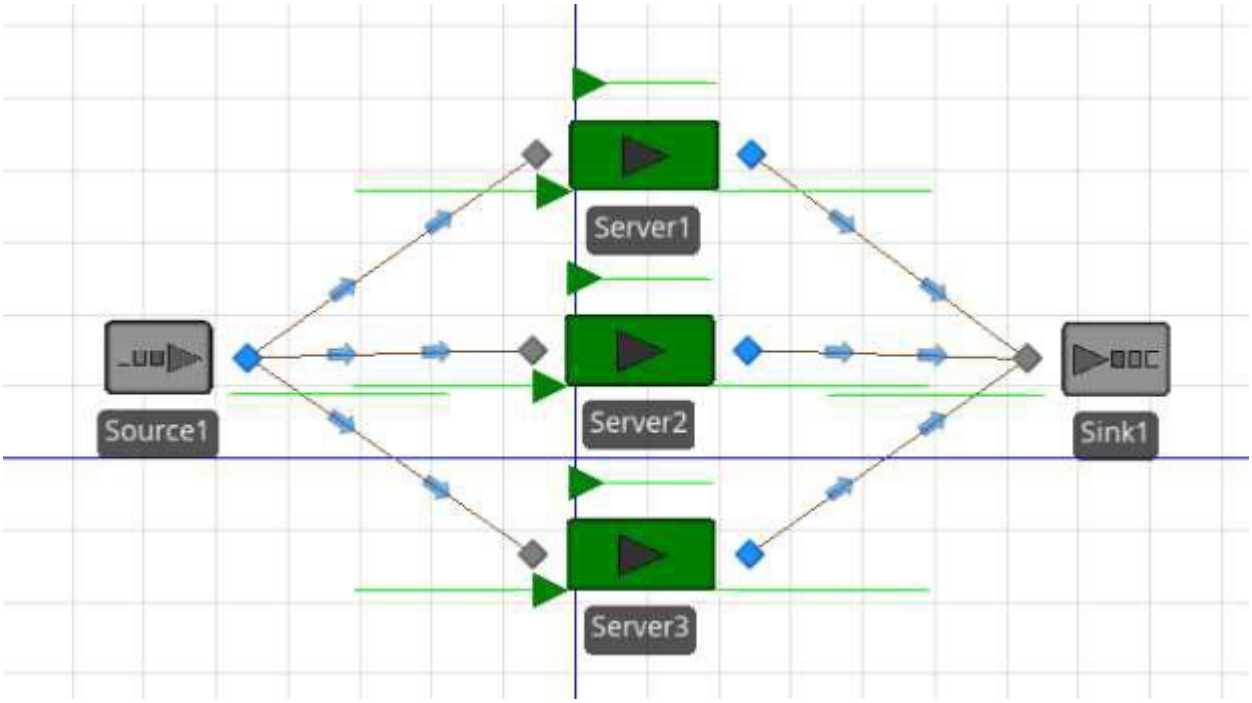


There is one area where it is necessary to focus on short-term utilization; that is, in situations where we have one or more bottleneck servers that limit production. In a manufacturing setting, a bottleneck server is typically an expensive but highly utilized machine. In a healthcare setting, a bottleneck server might include the operating rooms or expensive imaging equipment. In the case of a bottleneck server, we strive to keep our server busy to avoid losing short-term production capacity. This situation is discussed and illustrated in Principle 10. However for non-bottleneck servers, utilization is typically not critical to reducing WIP, maximizing throughput, or meeting on-time delivery dates.

Principle #3: A CONWIP strategy has less WIP for the same throughput.

One strategy for controlling variation – and therefore improving performance – is called CONWIP (constant quantity of work in process). This is a pull oriented strategy (as opposed to a push strategy), in which the completion of processing for one entity triggers the arrival of a new entity. In a manufacturing setting, we can implement a CONWIP strategy by releasing a new order to the shop each time an order is completed and exits the shop. In a service system, we can approximate a CONWIP strategy by scheduling customer appointments or reservations based on expected completion times for earlier customers. In either case, our goal is to keep a constant number of entities in the system.

To evaluate the impact of a CONWIP strategy, consider the following Simio simulation model, in which entities arrive and travel to the least busy of three servers. The entities enter the model at the source and travel along one of the paths to the selected server. After processing at the server, the entities depart the system at the sink.



The entities queue in front of each server and wait their turn to be processed. We will compare two cases for this system. In the first case, we have random arrivals that are completely independent of the departures. In the second case we keep a constant number of entities in the system, and create a new arrival each time we have a departure from the system. In both cases we have 3 scenarios where we initialize the model to specific WIP levels of 3, 6 and 9 entities respectively. Note that in the CONWIP case the WIP level will remain at this initial value throughout the run; however, in the Random case, the WIP will vary as entities arrive and depart the system. The servers have a random processing time with a triangular distribution and a minimum of .1, mode of .2, and maximum of .3 minutes. For the random arrival scenario, entities arrive randomly with an exponential distribution with a mean inter-arrival time of 4 seconds. Note that this is the arrival rate required to achieve a maximum throughput with three servers.

The results are summarized below for both the Random and CONWIP system for an initial WIP level of 3, 6, and 9 entities:

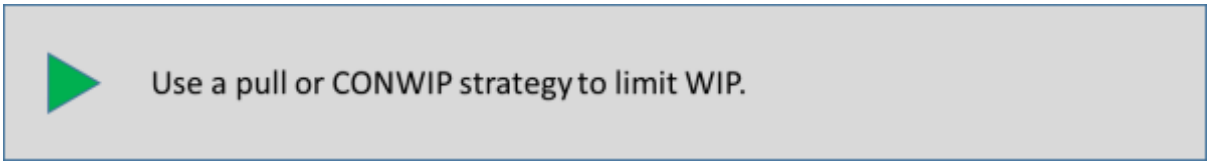
Scenario		Replications	Controls	Responses			
☑	Name	Completed	InitialWIP	RandomThrougput	RandomWIP	ConWIP	ConWIPThroughput
☑	Scenario1	10 of 10	3	21480.6	90.1405	3	15688.7
☑	Scenario2	10 of 10	6	21486.8	89.6011	6	21591.3
☑	Scenario3	10 of 10	9	21525.4	91.8081	9	21601.2

As we can see from these results, the random arrivals has average WIP of approximately 90 entities, whereas the CONWIP system drops the WIP to 3, 6, or 9. Although with a WIP of 3 the throughput for the system is much less, with a WIP of 6 the throughput for the CONWIP case matches the throughput of the random case with a huge saving in the average WIP.

As these results demonstrate, our CONWIP strategy produces the same throughput, but with much less work in process, and much shorter average cycle times through the system. This is an example of how managing the variation in the system can improve performance without having to increase capacity.

Although we are applying a CONWIP strategy in a very specific example, this principle applies across a wide range of systems. Any time you can eliminate or control a source of variation in the system (WIP in this case) the overall system performance improves.

The action item from this principle is:

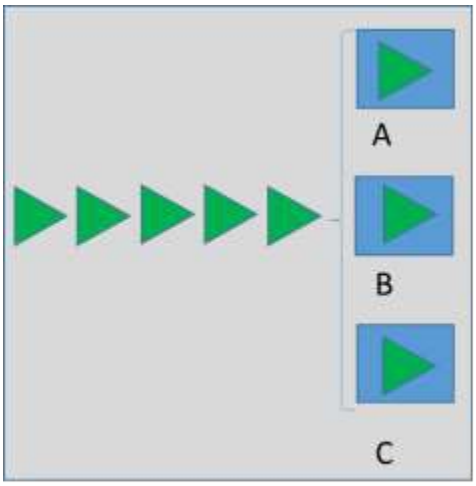


A manufacturing pull strategy can often be implemented without expensive information technology changes using manual systems such as cards (Kanbans) or empty totes sent back to upstream servers to trigger replenishment of component parts for downstream servers. Pull strategies can also be implemented in service systems; for example, a ski slope could show waiting times at lifts so that customers can select ski runs with short lines. Hence the arrivals to the various lifts are self-adjusting based on the congestion at the lifts. In a similar way, a health system might display current wait times on their website so that patients can self-direct to facilities with the shortest wait times. In both of these cases, we are striving to reduce arrivals during busy periods.

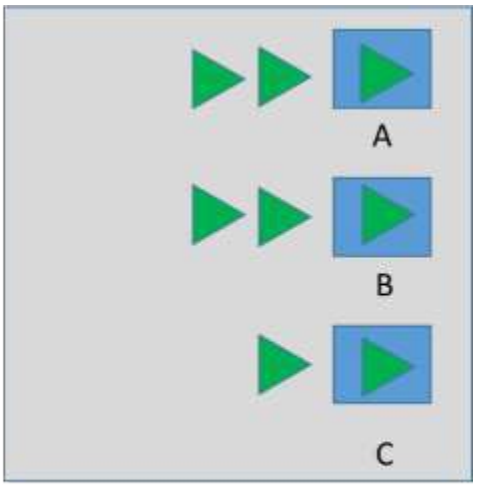
Principle #4: A single queue decreases WIP.

In the last example we used a system involving parallel servers, each with its own input queue. This arrangement is common in checkout lanes in retail stores. This principle states that in the presence of variability, it is better to a have a single queue for a set of parallel servers.

To demonstrate this principle, we compare the results of three servers with a single queue, to three servers with independent queues, as illustrated below:



Single Queue



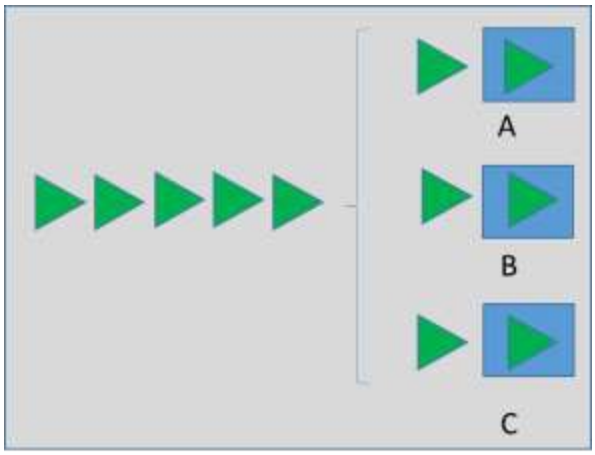
Independent Queues

The results for these two models are summarized below. As we can see, our single queue model has a smaller average WIP than the parallel queue version of the model.

Scenario	Replications	Responses			
Name	Completed	SingleWIP	SingleThroughput	MultiWIP	MultiThroughput
Scenario 1	10 of 10	6.77312	204921	7.11817	204729

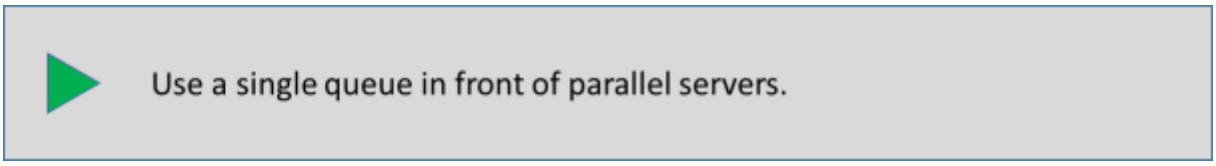
In addition to having better waiting time performance, the single queue version also ensures that customers are processed in the same order that they arrive. This gives customers a sense of fairness and predictability.

The single queue design is most practical when the travel distance between the front of the queue and the individual servers is small, so that the travel time from the front of the queue to the server is negligible. When this time is not negligible, a hybrid design can be used. A small queue with one or two places is placed in front of each server, and a single queue is then used to feed the smaller queues, as illustrated below:



Hybrid Queue

The action item from this principle is:



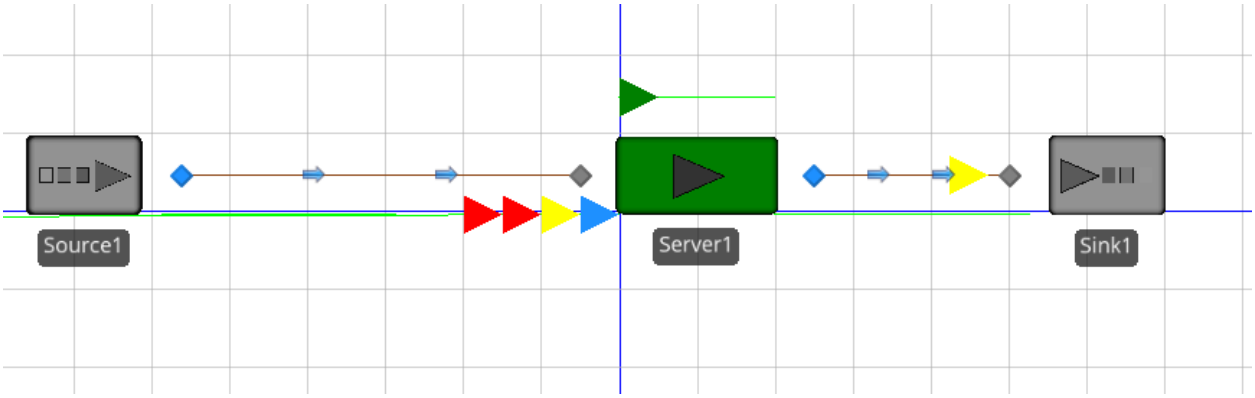
If the travel time from the head of the queue to the servers is large, consider adding small queues in front of each server that are fed by the larger, single-service queue.

Principle #5: Shortest Processing Time (SPT) first decreases WIP.

In many systems such as the preceding, the entities are processed on a first-in-first-out (FIFO) basis. In the case of service systems, customers may insist on this processing rule. However, if our goal is to reduce WIP (and therefore waiting times), this principle states that we can do so by processing orders with the smallest processing time (SPT) first.

To demonstrate this principle, we will model a simple production system with a single server, which processes four types of entities; green, blue, yellow, and red. The entities are randomly and equally distributed between the four types. Entities randomly arrive to the system with an inter-arrival time that is exponentially distributed with a mean of .25 minutes. The processing time for each entity is random with a triangular distribution. The minimum, mode, and maximum differ by entity type and are .0, .1, .2 for green entities; .1, .2, .3 for blue entities; .2, .3, .4 for yellow entities; and .3, .4, .5 for red entities, respectively. We will run this model under two scenarios. In the first scenario, we will process the entities by SPT based on their expected processing time, and in the second scenario we process them in the order of FIFO.

The following is a snapshot of our model executing using the SPT first rule. The server is currently processing a green entity, and four entities are waiting in the queue for processing. Using the SPT first rule, we will process the blue entity next, followed by the yellow entity, and then finally the red entities.



Let's now examine the results from our model under the two scenarios.

Scenario		Replications	Controls	Responses			
☑	Name	Completed	ServerRankingRule	WIP	WaitTime (Minutes)	MaxWaitTime (Minutes)	Throughput
☑	Scenario1	100 of 100	Smallest Value First	26.9446	6.25975	59.2813	5710.97
☑	Scenario2	100 of 100	First In First Out	42.3059	10.2291	24.3918	5691.36

Both scenarios have approximately the same throughput. However, SPT first produces substantially less WIP (by a third), and therefore reduces the average waiting time for our customers. In the case of the SPT rule, the maximum waiting time for a customer is 59 minutes, compared to 24 minutes for the FIFO

case. It is possible with highly congested systems to have some entities wait an excessive time when employing an SPT rule. However, this can be avoided by using a hybrid rule that specifies that an entity that has waited more than a specified amount of time is processed first, and for entities that have waited less than this amount the SPT rule is applied.

Fast service lanes in grocery stores are a simple illustration of SPT. Although this is typically done using dedicated quick checkout lanes, a better performing system in terms of WIP would be to combine principle 4 and 5 and have two queues feeding all servers – one queue for fast check outs – and one for normal checkouts. The server would always process a fast check out customer first if available, otherwise process a normal customer. Note that this is bit more complicated than separate queues in front of each server, but anytime you give priority to processing your "quick" customers first, you reduce WIP and lower the average waiting time for all customers.

Within a healthcare setting, the order of processing is sometimes dictated by the severity of the condition. However, in many cases the patients can be processed in any order, and FIFO may be viewed by the patients as the most fair. However, processing the patients by SPT first will lower the average waiting time for the system.

The action item from this principle is:

Note that this strategy may need to be modified if on-time delivery is the primary KPI, since a time critical entity may have a long processing time. In cases where on-time delivery is the primary concern, the SPT first strategy can still be incorporated into a hybrid rule. The hybrid rule must then also consider remaining production slack time for the entity so that time-critical entities can be processed first when necessary.

Principle #6: Moving variability downstream decreases WIP.

When flow lines are designed, an effort is typically made to balance the line so that the cycle time for each server is approximately the same. However, even if this is achieved, there are often differences in the variability of the cycle time from one server to the next. For example, one server may have a relatively simple set of tasks that take almost the same time each time an entity is processed, while another server might have a more complex set of tasks where the time varies significantly from one entity to the next. Although the average cycle time for each server is approximately the same, the variability is not.

This principle states that by moving variability towards the end of the line, we reduce the WIP in the system without reducing throughput. We have less inventory and congestion in our production system while processing the same number of entities.

To demonstrate this principle, we build a flow line model with three servers in series. Entities arrive to the line with a constant rate of 4 per minute. There is an infinite buffer between each pair of servers to store work in process. Each of the servers has an average processing time of .2 minutes. The servers with low variability have a processing time specified by a triangular distribution with a minimum value of .1, maximum .3, and a mode of .2 minutes. The high variability servers have a processing time that is exponentially distributed with a mean of .2 minutes. The standard deviation for this exponential distribution is X% larger than for the triangular distribution with the same expected value.

We run three scenarios with two low variability servers and one high variability server. The first scenario is named HighLowLow and has the high variability server as the first server. The second scenario is named LowHighLow and places the high variability server in the middle. The last scenario is named LowLowHigh and places the high variability server at the end.

The results show that the highest WIP occurs when we place the high variability server at the start of the line. The greater variability at the first server in the line impacts the performance of the entire line. In the last scenario, moving the high variability server to the end of the line has no impact on the first two servers, but yields a substantial (25%) reduction in WIP.

Scenario	Replications	Controls			Responses
Name	Completed	ProcessingTimeOne (Minutes)	ProcessingTimeTwo (Minutes)	ProcessingTimeThree (Minutes)	WIP
LowHighLow	100 of 100	Random.Triangular(.1,.2,.3)	Random.Exponential(.2)	Random.Triangular(.1,.2,.3)	4.68797
HighLowLow	100 of 100	Random.Exponential(.2)	Random.Triangular(.1,.2,.3)	Random.Triangular(.1,.2,.3)	4.92972
LowLowHigh	100 of 100	Random.Triangular(.1,.2,.3)	Random.Triangular(.1,.2,.3)	Random.Exponential(.2)	3.82732

Another way to state this principle is to try and put your most complex (i.e. variable) task at the end of a sequence, and get all the simple things out of the way first. If the sequence of tasks must be determined by the process itself, this may not be possible. For example, painting may have to be the last step in a manufacturing cycle, even though it has low variability in cycle time. However when possible, placing complex tasks at the end of a production process will improve performance. For example in health system it's not uncommon to have a procedure done (complex) followed by nurse briefing on instructions for proper care and for scheduling a follow up visit. This principle suggests that reversing the sequence would improve the performance of the system.

The action item from this principle is:

Principle #7: Moving fast servers downstream decreases WIP.

The objective in flow design is to allocate tasks to each server in a way that balances the load evenly among all the servers in the line. However, in practice, a balanced line is difficult or impossible to achieve. It is better to move the faster servers towards the end of the flow line because this will reduce the WIP in the system without reducing throughput.

To demonstrate this principle, we model a three-server flow line where each server has a slow, medium, or fast processing speed. The fast server has a processing time that is random, with a triangular distribution with a minimum of .001, mode of .1, and a maximum of .2. The medium-speed server has a processing time that is random, with a triangular distribution with a minimum of .1, mode of .2, and a maximum of .3 minutes. The slow server has a processing time that is random with a triangular distribution with a minimum of .15, mode of .25, and maximum of .35 minutes. Note that the three processing times have different expected processing times but the same variability. Entities enter the flow line at the first server at a fixed rate of 4 per minute.

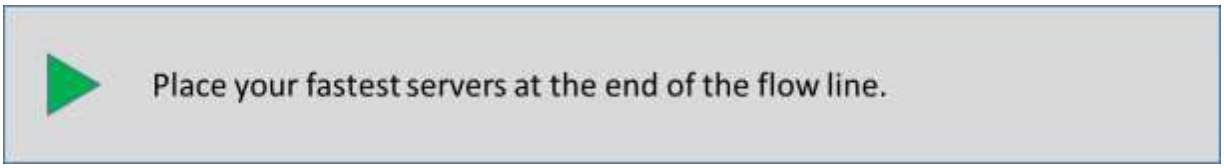
In the first scenario, we evaluate fast, medium, slow. In the second scenario, we evaluate the reverse order of slow, medium, fast. In both scenarios, we have an infinite buffer between servers for WIP. .

The results for the scenarios are shown below:

Scenario	Replications	Controls			Responses	
Name	Completed	ProcessingTimeOne (Minutes)	ProcessingTimeTwo (Minutes)	ProcessingTimeThree (Minutes)	WIP	Throughput
FastMediumSlow	100 of 100	Random.Triangular(.01, .1, .2)	Random.Triangular(.1,.2,.3)	Random.Triangular(.15, .25, .35)	9.2354	5748.54
SlowMediumFast	100 of 100	Random.Triangular(.15, .25, .35)	Random.Triangular(.1,.2,.3)	Random.Triangular(.01, .1, .2)	8.23035	5749.88

As we can see, the order of servers has no impact on throughput, but moving the faster servers downstream in the line reduces WIP (by 12% in this example). The intuitive reasoning that drives this result is that a faster server downstream is able to more quickly clear out work sent to it by a slower server.

The action item from this principle is:



Although this may not always be possible because the process sequence may be dictated by other factors, both manufacturing and service systems would be wise to explore opportunities to exploit this principle to improve performance.

Principle #8: Buffer space increases throughput and decreases WIP.

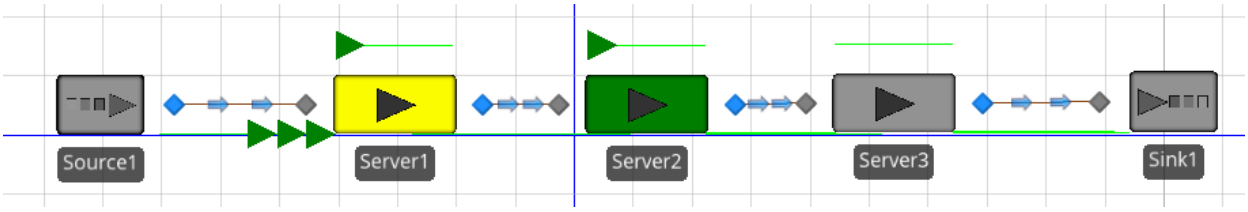
In evaluating the order of servers in a flow line, we have imposed no limits on the input or output buffer size at each of the servers. In theory, a server is never blocked by its downstream server, since it can always move the entity into the input buffer of the next server. In real life systems, though, floor space is limited and there is a finite buffer size for input and output at each server. If the buffers are full, one server may block its upstream server. Increasing buffer sizes will both increase throughput and decrease WIP. Hence, for a production line, we can get more output by adding additional buffer space without investing in more or faster servers. Note that adding buffer space will also decrease our WIP as a secondary benefit.

The intuitive reason that buffer space improves throughput is that it reduces both blocking and starvation. A server is considered blocked when it completes processing of an entity, but cannot move the entity out of the server area because the destination buffer is full. A server is considered starved when it is idle, but there is no entity in its input buffer waiting for processing. In the following figure, where server 2 and 3 have no space allocated for input buffers, server 1 is currently being blocked -- it has completed processing, but server 2 is still busy. Server 3 is being starved (idle) because there are no entities in its input buffer – it will remain starved until server 2 completes processing its current entity.

Note that buffers may be required on both the input side and output side of the server. This is the case when the completed entity at the server must be moved to the next server. Even if there is input buffer space at the next server, and output buffer is needed to temporarily store the entity until the movement device (e.g. a forklift truck) can come and pick up the entity to carry it to its next server. If there is no space in the output buffer, the server will remain blocked until space becomes available.

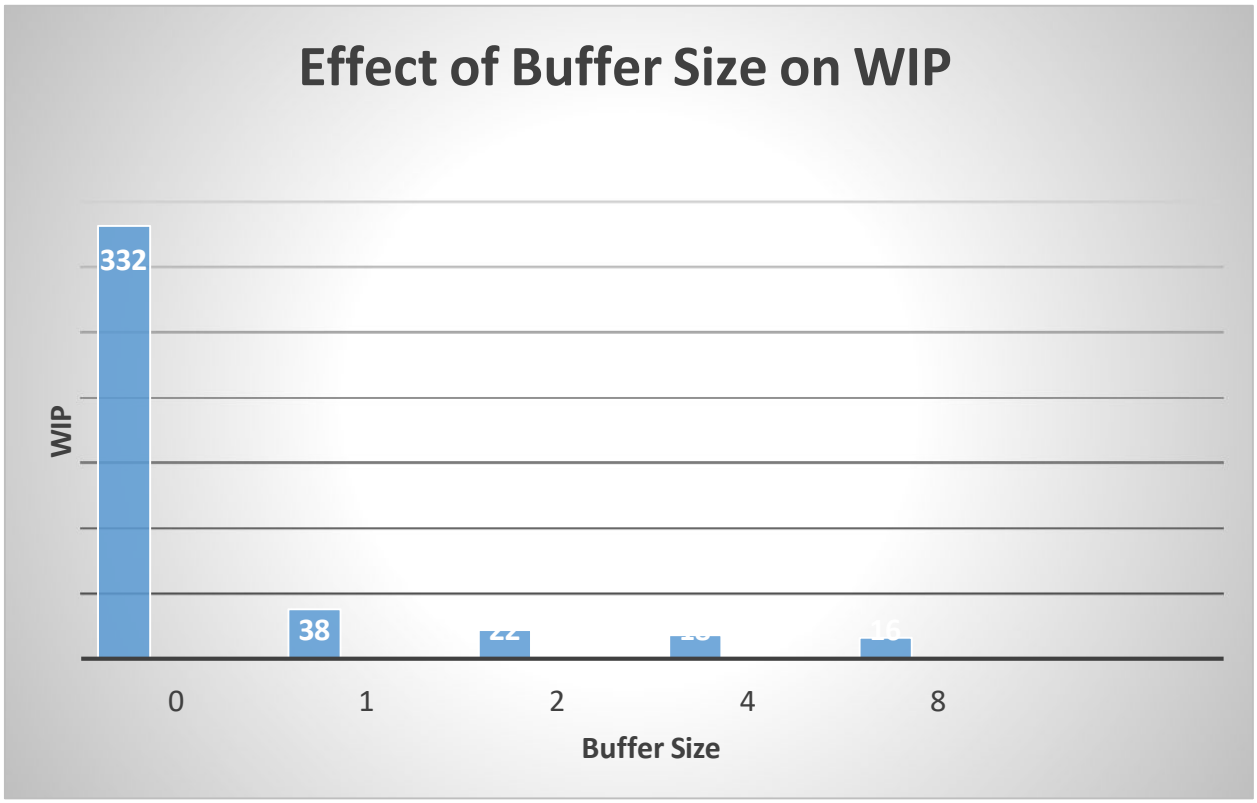
To illustrate the concept that adding buffer space increases throughput and decreases WIP, we model a three-server flow line with an infinite input buffer at server 1, and a fixed buffer size between servers 1 and 2, and servers 2 and 3. Because of the infinite queue in front of server 1, there is no bound on the WIP. Entities move from a server directly into the input buffer of the next server so that no output buffers are required. We will run five scenarios where we evaluate a buffer size of 0, 1, 2, 4, and 8. All three servers have a random processing time with a triangular distribution, and a minimum of .1, mode of .2, and maximum of .3 minutes. Entities randomly arrive to the flow line with an exponential inter-arrival time and a mean of .22 minutes. Note that this arrival rate creates a highly congested system with high server utilization.

A snapshot of the running simulation model with the buffer size in front of servers 2 and 3 set to 1 is shown below. Server 1 has changed color to yellow to denote that it currently blocked. Once server 2 starts work on the waiting entity, server 1 will be able to move its completed entity into the input buffer for server 2, and then start processing its next entity.



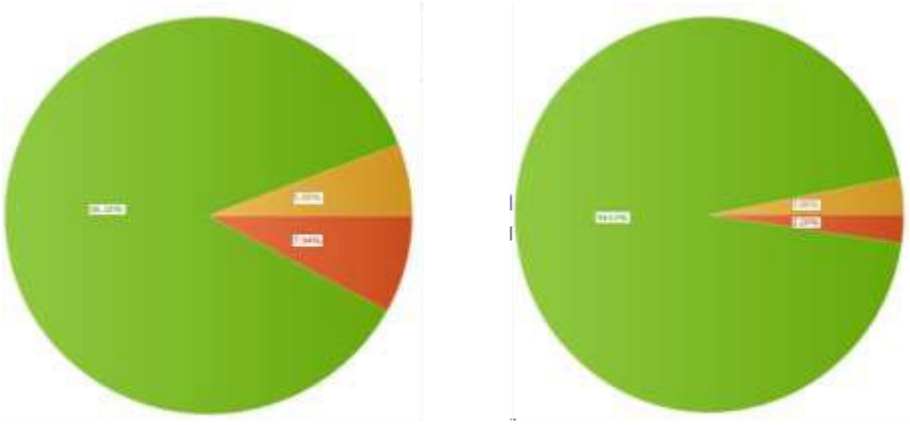
The results for the five different buffer sizes are shown below, along with a graph of the WIP as a function of buffer size. We see that increasing buffer size between server 1 and 2 and servers 2 and 3 has a significant impact on both WIP and throughput. With no buffers the average WIP is 332 entities, with nearly all of these queued in front of server 1. Adding a single buffer between each pair of servers cuts the WIP by nearly 90% to an average of 38 entities. The results indicate that for this simple flow line, a buffer size of 4 in front of each server will mitigate starvation and blocking and achieve good performance.

Scenario		Replications	Controls	Responses	
☑	Name	Completed	InputBufferCapacity	WIP	Throughput
☑	Scenario1	100 of 100	0	332.016	6186.31
☑	Scenario2	100 of 100	1	38.0815	6808.25
☑	Scenario3	100 of 100	2	22.3951	6838.9
☑	Scenario4	100 of 100	4	17.5051	6844.12
☑	Scenario5	100 of 100	8	16.494	6836.47



The following pie charts show the status of the second server in the flow line over time with a buffer size of 0 (left pie chart) or 1 (right pie chart). The green section denotes processing, the yellow denotes starved, and the red denotes blocked. In the case of no buffers, we see that server 2 is starved or

blocked nearly 14% of the time. This represents lost production capacity in the flow line and is the reason for the significant increase in WIP and decrease in throughput. By adding a single buffer to the line, we see that the percentage of blockage or starvation drops to just over 5%.



No Buffer

Single Buffer

The action item from this principle is:

Although this principle states that more buffer capacity is better in terms of throughput and WIP, it does not address the best way to allocate limited buffer space to a system. Rather than assigning an equal size to all servers in the system, our next principle can help us find better methods of buffer assignment.

Principle # 9: Buffering the Bottleneck increases throughput and decreases WIP.

In many production systems, there may be one or more heavily loaded servers that limit the throughput of the system. These servers are referred to as bottleneck servers. In a manufacturing system, the bottlenecks might be expensive machines; in a healthcare system, they might include operating rooms, imaging equipment, and other expensive resources in the system. The theory of bottlenecks has been discussed extensively by Goldratt as part of his Theory of Constraints (TOC).

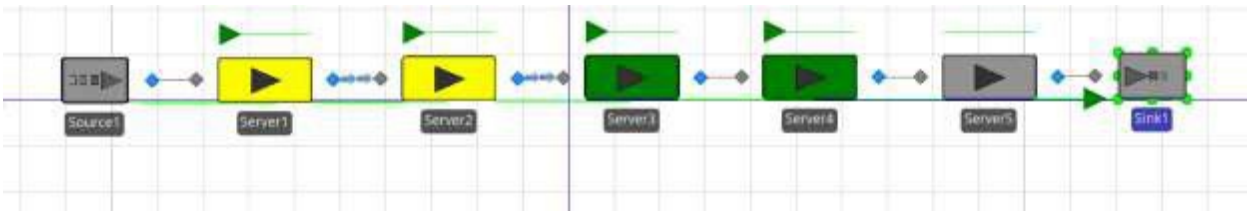
If time is lost at a non-bottleneck server due to blocking/starvation, it may have little or no impact on the throughput of the system, since there is excess capacity to make up for the loss. However, if time is lost at a bottleneck station, there is no excess capacity to make up for the loss. This often has a direct impact on both throughput and WIP.

This principle states that when allocating limited buffer space to a set of servers, it is better to favor the bottleneck servers. This implies not only the bottleneck server itself, but also any adjacent servers that may block or starve the bottleneck. The optimal allocation of buffer spaces to servers is a complex issue, and algorithms have been developed for this problem. This principle does not provide a specific algorithm for allocation buffer space, but simply states that it is generally better to favor the bottleneck when allocating space.

To illustrate this principle, we model a flow line with five servers, where server 4 is a bottleneck station. Servers 1 and 2 are relatively fast, with a random processing time that has a triangular distribution with a minimum of .1, mode of 1.5, and maximum of .2 minutes. Servers 3 and 5 are medium-speed servers with a random processing time that has a triangular distribution with a minimum of .1, mode of .2, and maximum of .3. Server 4 is our slow server, and has a random processing time that has a triangular distribution with a minimum of .14, mode of .24, and maximum of .34 minutes. Arrivals to the line are random with an inter-arrival time that is exponentially distributed with a mean of .25 minutes. Hence, the long-term utilization of server 4 is 96%, making it a critical bottleneck station.

We have no output buffers, an infinite input buffer in front of server 1, and finite input buffers in front of servers 2 through 5, with two buffers to allocate. In the first scenario, we allocate one buffer each in front of servers 4 and 5, and no buffers in front of 2 and 3. Note that scenario 1 provides extra buffer space at the bottleneck server to reduce the chance of starvation from server 3, as well as immediately downstream from this server to reduce the chance of the bottleneck station being blocked by server 5. Based on this principle, we would expect the first scenario (favoring the bottleneck server) to perform better than the second scenario (favoring the non-bottleneck servers). In our second scenario we allocate one buffer in front of servers 2 and 3, and no buffers in front of servers 4 and 5.

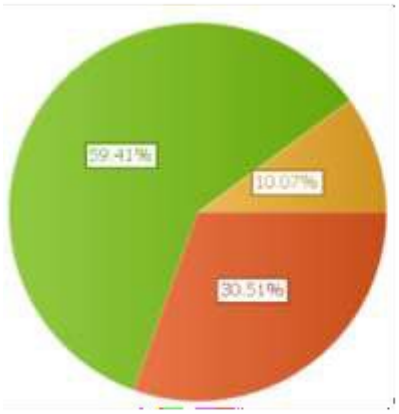
The following is a snapshot at .09 hours into the simulation under scenario 1 (buffering the bottleneck). At this point in the simulation, server 1 and server 2 are both blocked (indicated by yellow) because there are no buffers available and server 3 is busy. In this system design, most of the blocking/starvation occurs at server 1 and server 2; however, since they have excess capacity, they can generally make up for time lost as a result of blocking/starvation. By buffering both before and after server 4 (the bottleneck station), we keep the most critical server in the line productively working.



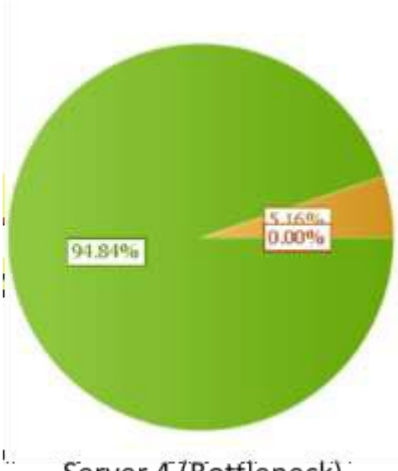
The results from the simulation model are shown below. Note that bottleneck buffered scenario has more throughput as well as one-fourth the WIP of the second scenario. Although we illustrated this principle with a simple flow line, the same concept applies to job shops as well.

Scenario		Replications	Controls				Responses	
☑	Name	Completed	Server2Buffer	Server3Buf...	Server4Buffer	Server5Buffer	Throughput	WIP
☑	BufferTheBottleneck	100 of 100	0	0	1	1	5730.92	14.7018
☑	BufferTheNonBottleneck	100 of 100	1	1	0	0	5645.82	66.4787

The following pie charts depict the processing (green), starved (tan), and blocked (red) resource states for server 2 (one of the fast servers) and server 4 (the bottleneck server). Although we have a small amount of starvation (5%) at the bottleneck server, we have no blocking of the bottleneck server by server 5. In contrast, our fast server has 10% starvation and is blocked more than 30% of the time. However, the fast server can use its excess capacity to make up lost production time; the same would not be true if the bottleneck server lost production capacity. This explains why using limited buffer space to buffer upstream and downstream from the bottleneck improves the system performance.



Server 2 (Fast)



Server 4 (Bottleneck)

The action item from this principle is:

Note that this may require additional space at the input/output buffers of servers upstream and downstream from the bottleneck server. In a flow line, identifying the upstream and downstream servers is straightforward. However, in a job shop, multiple servers may feed the bottleneck based on entity routings through the system. Nevertheless, the principle still applies: any server that sends a significant number of entities to the bottleneck server should be considered for buffering.

An example of applying this principle in healthcare would be proper sizing of the preparation rooms and recovery rooms for the operating room. Too few recovery rooms can block the operating rooms, and too few preparation rooms can starve the operating rooms.

Principle #10: Feeding the bottleneck increases throughput and decreases WIP.

Principle 9 discussed the importance of buffering the bottleneck server for improving throughput and WIP. Buffer space allocation is a decision that we make during the layout and design our production systems. Prioritizing entities is a related decision to be made. This principle states that when selecting entities to process at upstream servers, we should give priority to entities that feed the bottleneck server. The goal for this principle is the same as for the preceding one ---- to ensure that the bottleneck server remains busy (never blocked or starved) so that none of its capacity is lost during production. If we allow the bottleneck server to become starved for work or blocked from operating, the time lost during the idle periods represents lost production indicated by lower throughput and higher WIP.

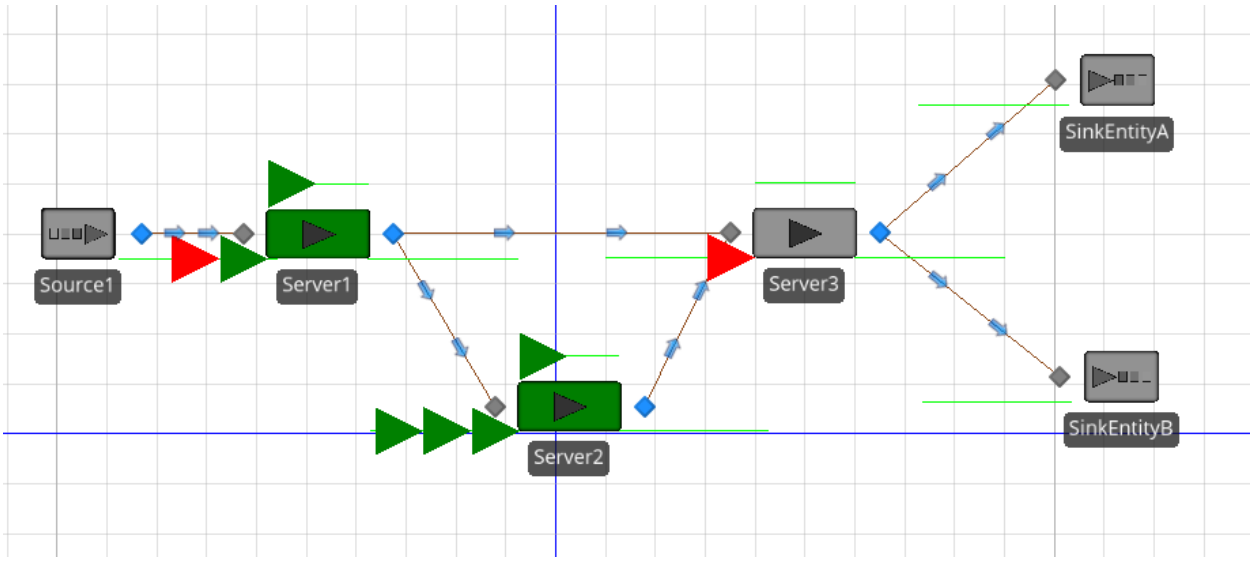
To illustrate this principle, we model a very simple job shop comprised of three servers, where we process two entity types. Entity type A goes first to server 1, next to server 2, and then to server 3. Entity type B goes to server 1 then moves next to server 3, bypassing server 2. Server 2 is our most expensive server and a bottleneck in our process.

Entities arrive to the system with an inter-arrival time that is random with an exponential distribution with a mean of .25 minutes. Fifty percent of the arriving entities are type A, and the remaining fifty percent are type B.

The processing time at server 1 depends on the entity type. Entity type A at server 1 has a fast processing time that is random with a triangular distribution with a minimum of .03, a mode of .04, and a maximum of .05 minutes. Entity type B at server 1 has a much slower processing time that is random with a triangular distribution with a minimum of .34, mode of .44, and maximum of .54 minutes. Note that if we process entities FIFO at server 1 we may delay processing a type A by the slower processing of type B. However, if we always process entity type A before processing entity type B, we ensure that entity type A is quickly processed by server 1 and sent to server 2, which is the bottleneck server in the system.

Server 2 has a processing time that is random with a triangular distribution with a minimum of .39, a mode of .49, and a maximum of .59 minutes. Since entities arrive to this server with a mean inter-arrival time of .5 minutes, and the average processing time is .49 minutes, this is a highly utilized server that is a bottleneck in the system. Server 3 has a relatively fast processing time that is random with a triangular distribution with a minimum of .05, mode of.15, and maximum of .2 minutes.

We compare two scenarios for this system. In the first scenario we process all entities in the order that they arrive to each server, or FIFO. In the second scenario, we give priority to entity type A at server 1 with the aim of always feeding the bottleneck (server 2) with work as early as possible, to reduce the chance that it will go idle and lose capacity that cannot be recovered. The following is a snapshot of the model at .09 hours into the simulation under the second scenario, where entity type A at server 1 is given priority. Note that the type A entity (green) is queued in front of the type B entity (red) at server 1.



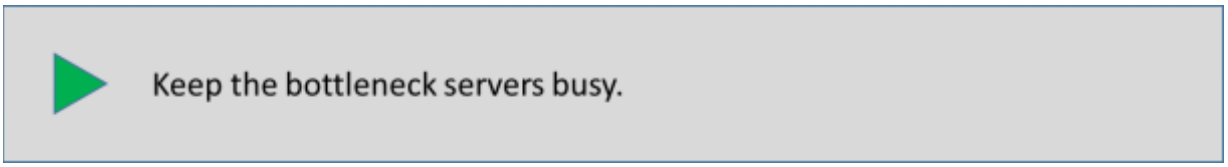
The results from the simulation are summarized below. There is a small improvement in throughput for entity type A, but a significant (25%) decrease in WIP when feeding the bottleneck by giving priority to entity type A at server 1.

Scenario	Replications	Controls	Responses		
Name	Completed	RankingRuleServer1	WIP	Throughput A	Throughput B
FIFORule	100 of 100	First In First Out	39.7539	2842.61	2869.46
TypeAPriorityRule	100 of 100	Largest Value First	30.3776	2862.3	2867.51

The concept of feeding the bottleneck can be adjusted to apply only when the work in front of a bottleneck server drops below some target level. Note that in the previous snapshot, server 2 had three type B entities queued up for processing, so it wasn't necessary to prioritize type A entities at server 1 at this particular point in the execution. We could give priority to entities that minimize changeovers as long as the bottleneck server has plenty of work; otherwise, we can ignore changeovers and select an entity that will feed the bottleneck server.

Note that information flow is crucial to implementing the strategy of feeding the bottleneck when it's in danger of being starved. A system must include a way to inform resources upstream to the bottleneck that the bottleneck is in danger of being starved, so that real time decisions can be made to avoid starvation.

The action item from this principle is:



To accomplish this, make sure that entity selection rules are in place to ensure that bottleneck servers are not starved for work because of entity selection decisions made at upstream servers. The next principle addresses the concept of minimizing changeovers during entity selection.

Principle #11: Minimizing changeovers increases throughput and reduces WIP.

In some systems, there is a changeover cost to switch from processing one entity type to another. The changeover cost is typically expressed as time required to change a server to process a different type of entity. This is common in manufacturing systems where a machine needs tool changes for each part type. This changeover cost can be expressed in several different ways. The simplest form defines one changeover cost for processing the same entity type as before, and a second changeover cost when changing to a different part type. A more complex form provides a From/To matrix of changeover values that defines the changeover cost to change between all pairs of possible entity types. This principle states that regardless how changeovers are defined that operating rules that minimize changeovers will increase system throughput and reduce work in process.

The basic premise behind this principle is that by eliminating changeovers, we increase the available production time at a server. This allows more entities to be processed by the server, which in turn reduces WIP.

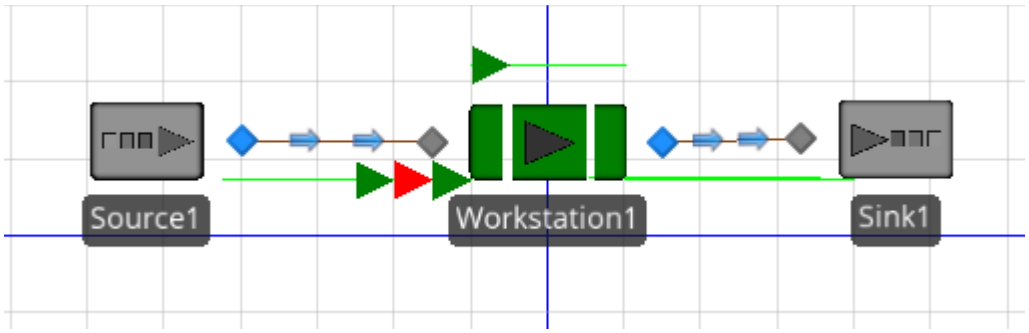
To demonstrate this principle, we model a single server that processes two entity types: A and B. The following matrix shows the changeover times in minutes between these two types:

From/To	A	B
A	.2	.8
B	.6	.1

Entities arrive to the server randomly with an inter-arrival time that is exponentially distributed with a mean of .7 minute. Fifty percent of the arriving entities are type A, and the remaining fifty percent are type B. The processing time for a type A entity is random, with a triangular distribution with a minimum of .3, mode of .5, and maximum of .8. The processing time for a type B entity is random, with a triangular distribution with a minimum of .2, mode of .4, and maximum of .7.

We compare two scenarios for this system. In the first scenario, we process entities in the order that they arrive to the server (FIFO). In the second scenario, we use a dynamic selection where we select the entity with the smallest setup time at the server.

The following is a snapshot of the executing simulation model at .11 hours into the simulation. The workstation is currently processing a type A entity (green), and the input buffer has a type A entity, followed by a type B entity (red), followed by a type A entity. If we process these in the order that they arrive (FIFO), we will do two changeovers switching from A to B, and then back from B to A. However, if we make a dynamic selection based on minimizing setups, we will first process both type A entities in the input buffer, then followed by the type B entity, and as a result only incur a single changeover.



As shown in the results below, the dynamic selection of the smallest changeover time performs dramatically better than the simple FIFO rule. We see a significant increase in throughput (30 %) combined with a dramatic reduction in WIP (96 %).

Scenario		Replications	Controls	Responses	
☑	Name	Completed	DynamicSelectionExpression	Throughput	WIP
☑	FIFO	10 of 10	Entity.TimeCreated	1584.9	245.319
☑	SmallestChangeover	10 of 10	Workstation1.ActualSetupTime	2026.8	8.941

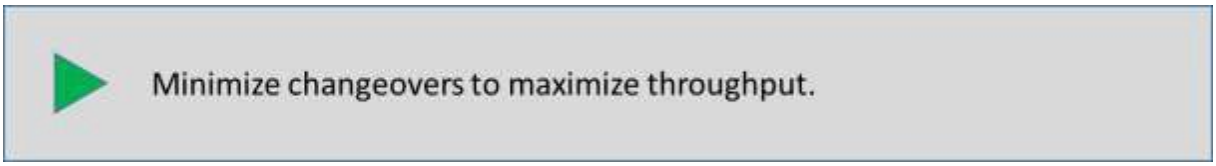
The benefit of minimizing changeovers will be greatest for servers that are highly utilized, where increasing the server availability can directly increase throughput, and in turn lower WIP.

Another way to state this principle is to bunch similar items together. In a manufacturing setting, this means processing all of one type of parts before switching to a second type. In a healthcare system, this might involve processing similar patients as a block before switching the next set of patients.

Note that a strategy that minimizes changeovers has the same issue that the shortest processing time rule has – it may cause certain entities to wait a really long time before getting processed. An entity may sit in its queue for a very long time because it has a long setup time for the server, and new arriving entities may continually jump ahead in priority. This issue can be addressed using a hybrid rule, where we select the entity with the smallest changeover time unless an entity has waited longer than a threshold time.

In some cases, the task for a server can be split up and performed in parallel by two or more servers. In this case it may be beneficial to incur extra setups on parallel servers to be able to spread the tasks across servers and thereby complete the task in less time. We will discuss this aspect of setup times in principle 12.

The action item from this principle is:



When throughput is not a primary KPI, hybrid selection strategies that combine changeover times with other criteria such as maximum waiting time or on-time delivery should be considered.

Principle #12: Task splitting may improve performance.

In Principle 11, we discussed the benefit of selecting entities for processing that minimize changeover times. Under certain conditions, it is better to incur additional setups in order to split a task across multiple parallel servers.

Not all tasks are capable of being split. For example, a machining operation on a work piece or an x-ray for a patient in a health clinic cannot be easily split across two or more servers as a means of completing the tasks sooner. However, if the task is to process a batch of 100 parts, we could easily process 50 parts on one server, and 50 parts on a second server, or split the batch of 100 into four batches of 25 each to be processed by four parallel servers.

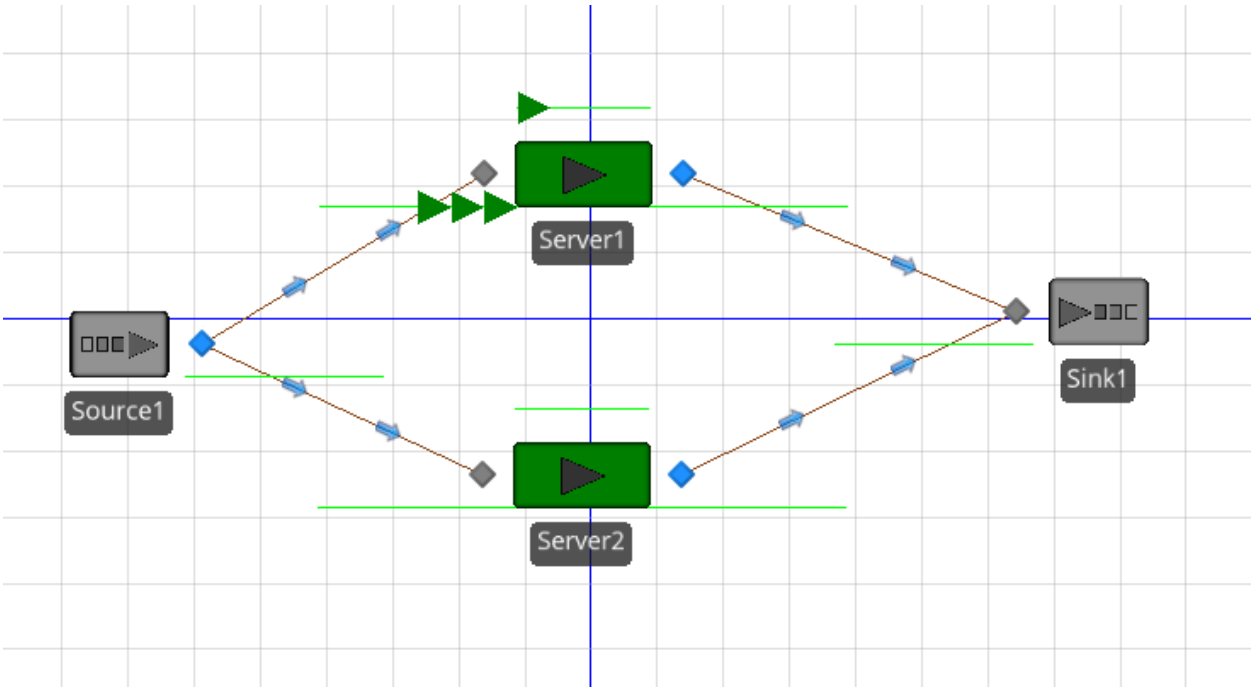
For task splitting to be beneficial, the alternate servers must be non-bottleneck servers. If we perform an extra setup operation or performing additional processing on a bottleneck server, we are impacting the production capacity of the system as a whole. The decision to split a task may be beneficial for the specific entity involved, but detrimental to other entities that must be processed by the servers. On the other hand, if we perform extra processing on a lightly used server, we have minimal impact on the production capacity of the system.

When the conditions are right, task splitting will increase throughput, reduce WIP, and improve timely delivery.

To demonstrate the process improvement for this principle, we will model a simple system of two servers that process two types of entities. Entities arrive to the system with an inter-arrival time that is exponentially distributed with a mean of .15 minutes, and 80% of the arriving entities are type A, with the remaining 20% being type B. The processing time for entity type A on server 1 and entity type B on server 2 is random, with a triangular distribution having a minimum of .1, a mode of .2, and maximum of .3 minutes. If server 2 is idle when processing begins for entity type A on server 1, it is possible to split the work for entity A across both server 1 and server 2. In this case, each server requires 60% of the time on server 1 alone to process the entity.

We will evaluate this system under two scenarios. In the first scenario, we will allow the processing task for entity type A to be split across both servers when server 2 is idle. In the second scenario, we allow no splitting. Type A entities are processed exclusively by server1, and type B entities are processed by server 2.

The following is a snapshot of the executing simulation at time .06 hours into the run for scenario 1, the case where we allow task splitting. Entities enter at the source and then select one of the two emanating paths based on link weights specified as 80 (server 1) and 20 (server 2). The entities sent to server 1 are type A entities and wait their turn to be processed by server 1. The entities sent to server 2 are type B entities and wait their turn to be processed by server 2. We have also added custom processing logic to server 1 to check the status of server 2, and if available, to seize it as a secondary resource for processing entity type A, and we adjust the processing time to 60% of its normal time. If a type B entity arrives to server 2 during this time, it must wait for the server to finish helping server 1 with entity type A. Note that in this snapshot both server 1 and server 2 are colored green to indicate that both are processing the type A entity displayed in the server 1 processing queue, and there are no type B entities at server 2.



The results for the task splitting and no task splitting scenarios for this system are shown below. For this simple example, task splitting increases throughput and has a dramatic (99%) reduction in WIP.

Scenario		Replications	Controls	Responses	
☑	Name	Completed	AllowTaskSplitting	Throughput	WIP
☑	TaskSplitting	10 of 10	☑	9579	2.92934
☑	NoTaskSplitting	10 of 10	☐	9118.9	236.643

The action item for this principle is:

This strategy works best when the parallel servers are not bottlenecks, and there is little or no extra setup times or other complications for splitting the task. This strategy can be effective for both increasing throughput and also meeting timely deliveries. In the latter case, we assign an increasing priority to splitting tasks based on a timeliness measure such as the remaining slack time for the entity.

Principle #13: Worker flexibility improves performance.

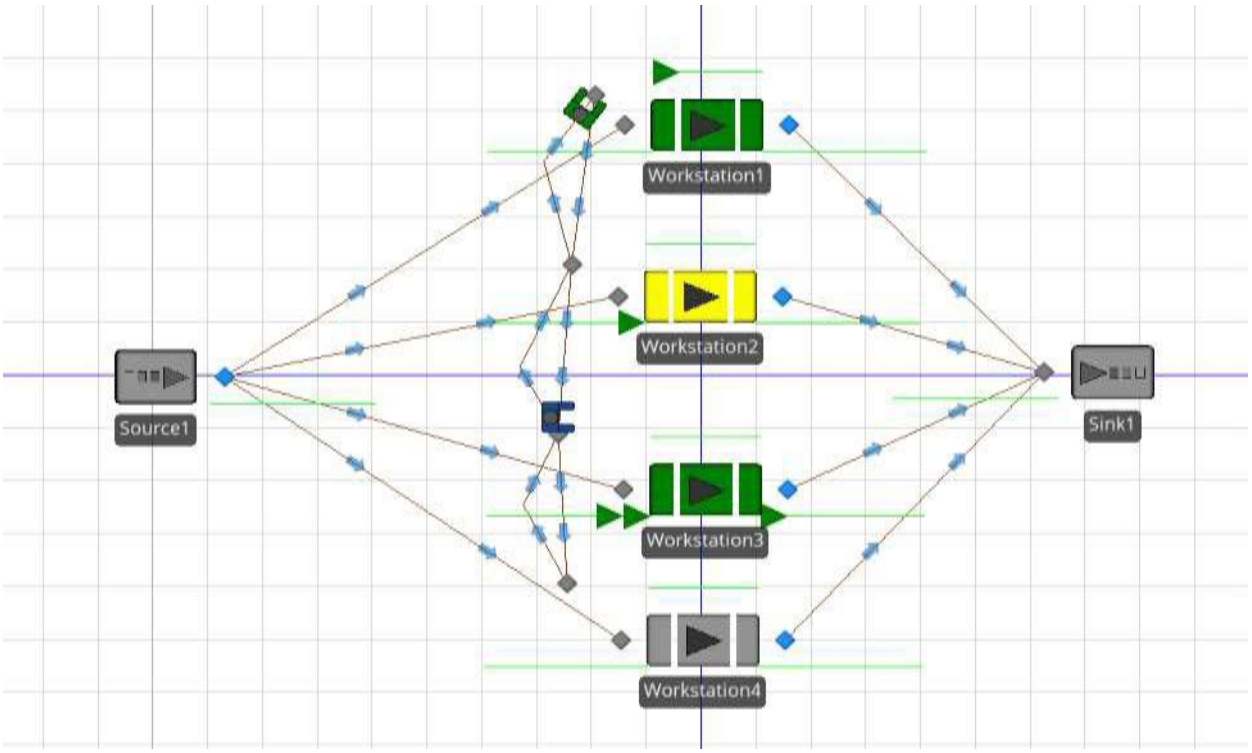
An expensive and critical element of many manufacturing and service systems is the workers in the system. In a manufacturing facility, operators that perform machine setups, assemble components, monitor operations, inspect parts, etc. In service systems, staff members perform some type of service for the customers, such as checking blood pressure and heart rate, or reading an x-ray. The workers may be dedicated to a specific server such as a machine or exam room, or shared between multiple servers.

This principle states that adding flexibility to our workers will improve performance. We should see an increase in throughput, reduction in WIP, and improvement in timely delivery.

To illustrate this principle, we model a simple manufacturing system with four workstations, where entities arrive for processing. Each entity requires a random setup time by a worker at the workstation, which has a triangular distribution with a minimum of 0, mode of .2, and maximum of .4 hours. Processing is automatic (does not require a worker) and has a triangular distribution with a minimum of .1, mode of .2, and maximum of .3 hours. Entities randomly arrive to the system with an exponentially distributed inter-arrival time with a mean of .1 hours, and are sent equally and randomly to one of the four workstations. Two workers perform the setup operations, so an entity must wait for both the workstation and worker before setup can begin.

Two scenarios are being considered. In the first scenario, worker 1 is dedicated to service workstations 1 and 2, and worker 2 is dedicated to service workstations 3 and 4. In the second scenario, both workers are trained to flexibly service all four workstations. In both scenarios, an idle worker will choose the closest workstation requiring service. Based on this principle, we would expect the second scenario (flexible workers) to perform best.

The following shows a snapshot from our model at time .67 hours in the simulation. The entities enter at the source and are sent to one of the four workstations for setup and processing, and then travel to the sink where they depart the system. At the current point in the simulation, worker 1 is busy (green) doing set up operation at workstation 1. Workstation 2 requires a set up, but worker 1 is not available, so the workstation setup is held up (yellow) and cannot start. Workstation 3 is busy processing and has three waiting entities in its queue, workstation 4 is idle, and worker 2 is idle (blue). Note that we have a workstation that requires setup (workstation 2) and an idle worker (worker 2), but because worker 2 is not trained on workstation 2, the setup must wait for worker 1 to become idle. When operating this same system under the second scenario (flexible workers), this situation would not occur; hence, we would expect better performance under the flexible worker scenario. Note that in our first scenario we have two categories of workers, whereas in a second scenario we have a single category. Reducing the number of trained categories of workers by cross-training generally yields improved performance.



The results for this system are shown below. As expected, the flexible workers both improve throughput and lower WIP.

Scenario	Replications	Controls				Responses	
Name	Completed	WS1WorkerList	WS2WorkerList	WS3WorkerList	WS4WorkerList	Throughput	WIP
DedicatedWorkers	100 of 100	Worker1Only	Worker1Only	Worker2Only	Worker2Only	199.2	27.2381
FlexibleWorkers	100 of 100	Worker1And2	Worker1And2	Worker1And2	Worker1And2	204.01	25.5585

The action item from this principle is:

There are obviously constraints that limit the application of this principle. For example, flexible workers may be more expensive, and some tasks (such as brain surgery) may demand a level of expertise that make cross training impractical. However, for many systems reducing the number of worker categories will improve performance.

A recent example of this strategy can be found in the airline industry. Previously, separate cleaning crews were used to clean planes between offloading and on-loading of passengers. However, delays were encountered waiting for cleaners to arrive. By cross training the flight attendants to clean the cabins between flights, there is never a delay waiting for the cleaners to arrive. The net results are faster turnaround time for the airplane, fewer delayed flights for the passengers, and greater utilization and profitability for the airline.

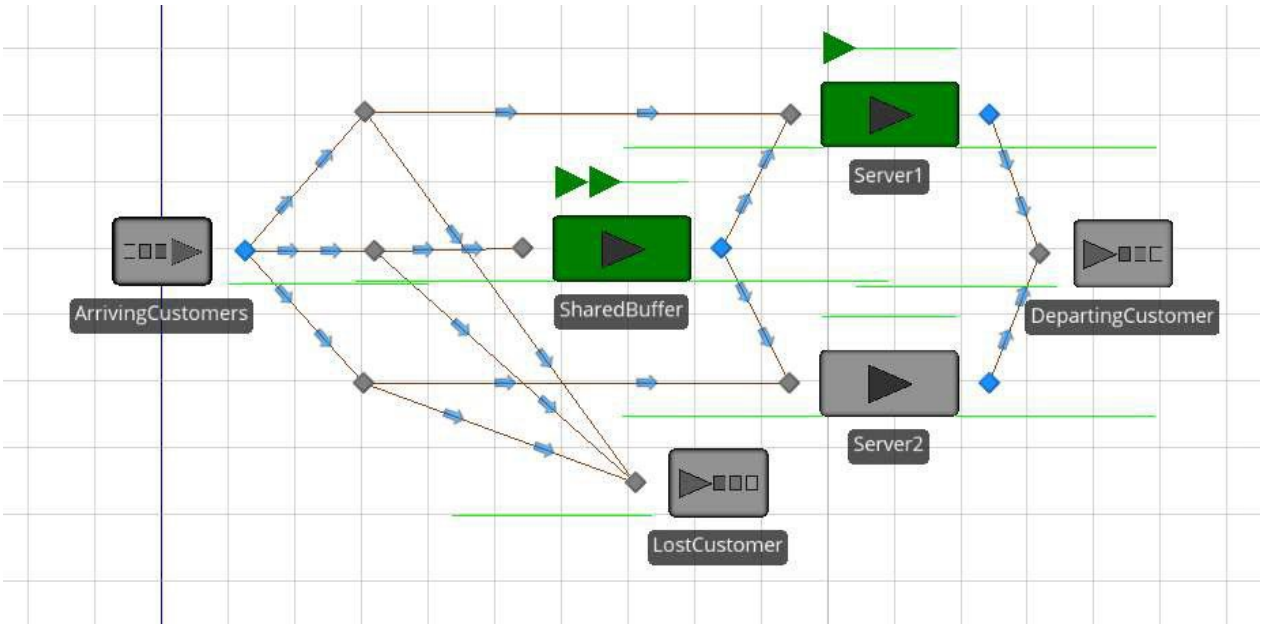
Principle #14: Buffer flexibility improves performance.

Buffers can play an important role in improving system performance. This principle states that employing a shared buffer with the same total buffer space will improve performance over a system with dedicated buffers assigned to each server. Implementing this strategy depends on having the correct layout of the system. In a health clinic, a common waiting room can be provided to support two or more doctors' offices. In a manufacturing cell, machines can be arranged around a central buffer area that is shared by all the machines.

To illustrate this principle, we model a system of two servers and two buffer spaces. One-half of the arriving customers are sent to server 1 for processing, and the other half are sent to server 2. Entities that arrive to the system when the two buffers are full depart immediately and are considered to be lost customers. In the first scenario, we place one dedicated buffer space in front of server 1 and a second in front of server 2. In the second scenario, we have a common buffer area with two buffers that can hold entities moving to either server 1 or server 2, analogous to a shared waiting room in front of two offices. Entities arrive to the system with a random inter-arrival time that is exponentially distributed with a mean of .18 minutes. The processing time for entities on each of the two servers is random, with a triangle distribution minimum of .1, mode of .2, and maximum of .3 minute.

We model this system using a server named SharedBuffer with a capacity of 2 and processing time of 0 to represent the shared buffer area. We use a Boolean control variable named UseCommonArea to route entities to either the dedicated waiting area for each server, or the shared buffer area, depending on which scenario we are simulating. For the case of a common buffer, we send entities to the sink named LostCustomers if the shared buffer is at capacity; in the case of dedicated buffers at each server, we send entities to the LostCustomers sink if the dedicated input buffer at the server is full.

The following shows a snapshot of the simulation at .09 hours running scenario 2 (UseCommonBuffer is True). At this point, server 1 is busy processing a customer, server 2 is idle, and we have two customers waiting in the shared buffer for processing at server 1. Note that in the dedicated buffer case, our second waiting customer would have been a lost customer, since each server would have a single buffer.



The results for these two systems are summarized below. In the scenario 1, we turn off the use of the common area and set the local buffer size to 1. In scenario 2 with shared buffers, , we turn on the use of the common area (which can hold 2 entities for either server) and set the local buffer in front of each server to 0.

Scenario		Replications	Controls		Responses		
☑	Name	Completed	UseCommonA...	LocalBuffer	WIP	Throughput	LostCustomers
☑	DedicatedBuffers	10 of 10	☐	1	1.28108	7068.2	965.4
☑	SharedBuffer	10 of 10	☑	0	1.4764	7352.5	645.8

Note that by sharing the two buffers across both servers, we have fewer lost customers and greater throughput. We also have slightly more WIP; this is the result of our ability to more flexibly handle waiting customers – hence we have more waiting customers on the average rather than lost customers.

The action item for this principle is:

Principle #15: Server flexibility improves performance.

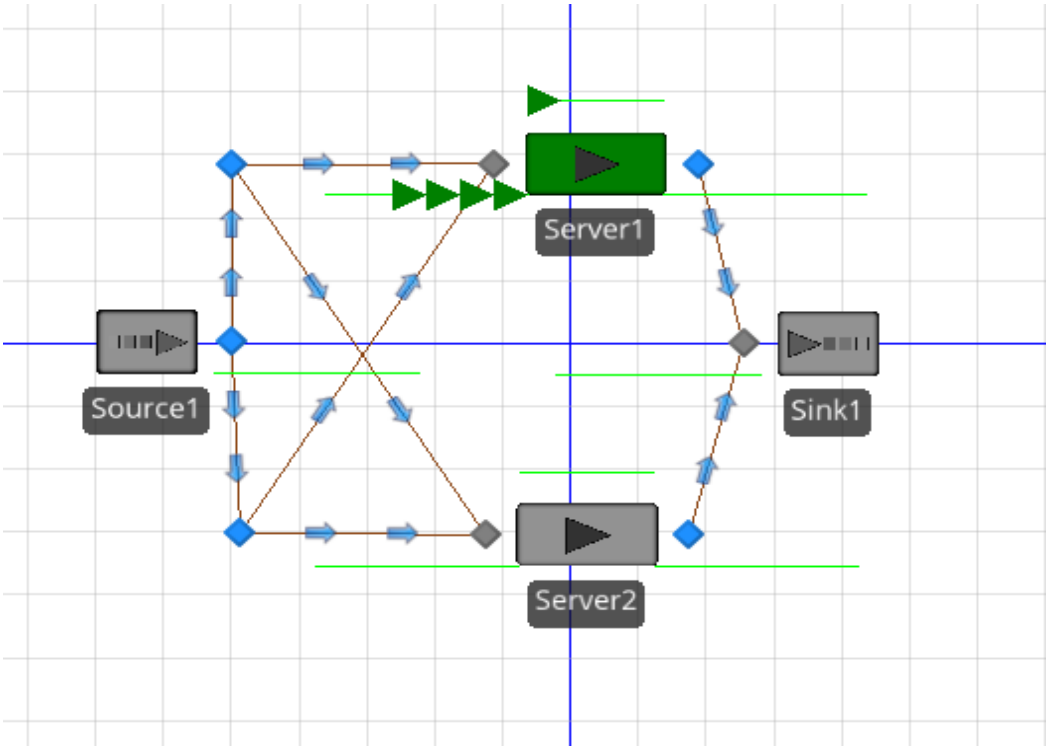
As a general rule, flexibility improves performance. We have previously discussed how flexible workers and buffers improve performance. This principle states that flexible servers will also improve performance.

An example of a flexible server is a numerically controlled machine with a built-in tool changer. This single machine can perform the required setup and processing for multiple types of entities.

To illustrate this principle, we will model a system with two servers that process two types of entities. In the first scenario, servers will be flexible; that is, each machine can process either type of entity. The second scenario involves dedicated machines, where each machine can process only one type of entity. Entities randomly arrive with an inter-arrival time that is exponentially distributed with a mean of .11 minutes. The processing time for entities of either type on either server has a triangular distribution with a minimum of .1, mode of .2, and maximum of .3 minutes. The entities are equally and randomly distributed between the two types.

Within our model, we will employ a Boolean (True/False) control variable named FlexibleServers to control the routing of entities to servers. If this variable is true (Scenario 1), the arriving entity is sent to the least loaded server, since either server can perform the task. However, if this variable is false (Scenario 2), the arriving entity is sent only to the server type that is capable of processing the entity.

The following is a snapshot of the running simulation at time .04 hour for Scenario 2, dedicated servers. At this point in the simulation, server 1 is busy with four waiting entities, and server 2 is idle. Since server 2 cannot process the first entity type, these entities must wait to be processed on server 1.

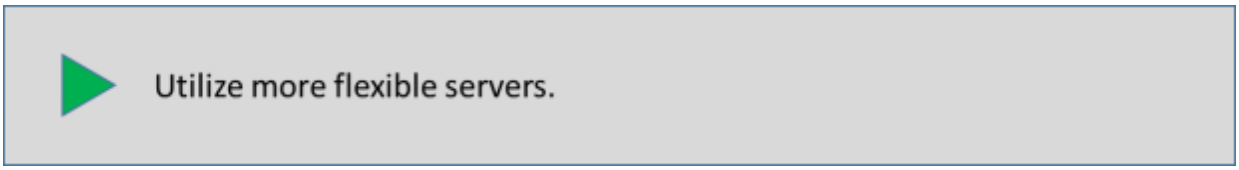


The results for these two variations in the system are shown below. For the same throughput, the flexible servers have significantly less WIP. Note that the differences between these systems would be more dramatic if we had an uneven balance of entity types.

Scenario		Replications	Controls	Responses	
☑	Name	Completed	FlexibleServers	WIP	Throughput
☑	Flexible Servers	100 of 100	☑	6.36162	13081.4
☑	Dedicated Servers	100 of 100	☐	11.1793	13074.2

There are more benefits to flexible servers over dedicated servers than simply improving our KPIs. If a dedicated server breaks in the second scenario, we can no longer process entities requiring that operation until a repair is completed; however, in the case of Scenario 1, a breakdown of one flexible server does not prevent processing of any of the entities. Flexible servers also make it easier to introduce new entities into the system. For example, a flexible set of cooking stations in a fast food restaurant make it easier to introduce new menu items. Finally, flexible servers make it possible to have fewer servers, since the sharing of equipment reduces capacity requirements.

The action item for this principle is:



In a manufacturing system, a flexible server typically implies the purchase of flexible machines, whereas in a service system, a flexible server scenario often involves training personnel, access to online help systems or experts, and perhaps additional equipment. Although flexible servers may be more expensive than dedicated servers, their advantages could offset the added cost.

Principle #16: Transfer batching may improve performance.

Sometimes a server in a production system processes an entity representing a group of N individual items. For example, in a manufacturing system, the entity may consist of many (N) work pieces, where each work piece must be individually processed by the server. In this case, a setup procedure may be required at the start of the lot, but each work piece may not require additional setup. Once the entire lot is completed at the server, it is then transferred to its next server for processing. This principle states that performance may be improved by moving the lot in smaller, multiple sub-lots based on a transfer batch size. For example, a lot of 100 items might be moved in sub-lots of 10 each. That way, the items can begin work at the next server sooner – without waiting for all 100 items to be completed.

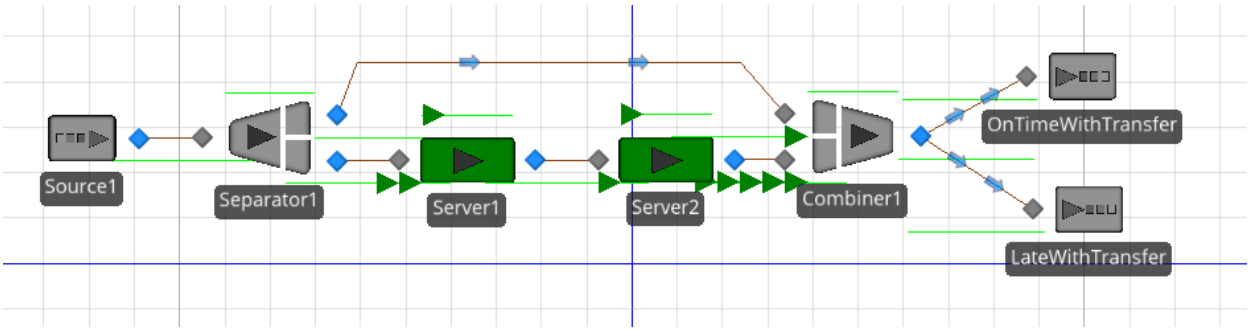
Depending on how transfers are completed (carried by a worker, moved by a fork lift truck, etc.), transfer batching may place an extra load on transfer resources. Hence transfer batching makes sense primarily when the transfer resources are readily available and the next server is also available for processing.

Transfer batching has the most impact on performance in situations where the downstream server is a starved bottleneck. By transferring sub-lots, we are able to "feed the bottleneck" without having to wait for the entire lot to be completed for moving to the bottleneck server.

To illustrate this principle, we model a flow line with two servers in series. Entities representing a lot of 10 parts arrive to the first server, which processes items one at a time until the lot is complete. The processing time per item is random, with a triangular distribution and has a minimum of 20, mode of 30, and maximum of 40 minutes. Once the lot is complete, it moves to the next server where the processing time per item is random with a triangular distribution and has a minimum of 10, mode of 30, and maximum of 50 minutes. Lots randomly arrive to the production line with an inter-arrival time that is exponentially distributed with a mean of 6 hours. Lots have a due date that is 16 hours after their arrival time to the system.

We will evaluate this base system against an alternative which implements a transfer batch size of 1 item; that is, each time an item is complete at the first server, it is transferred for processing at the second server.

The following is a snapshot of the model that implements the transfer batching at time 3.47 hours into the simulation. Each entity representing a lot of 10 items arrives to the separator where it separates into 10 entities that flow through the two servers, and then these 10 entities are re-combined with the original entity once the entire lot is complete. Hence, this model allows entities in the lot to be individually processed on server 1 and server 2 without waiting for the entire lot to be completed at each server.



The results for these two alternatives (No Transfer, With Transfer) is summarized below. As we can see transfer batching has improved both the on time delivery and WIP.

Scenario	Replications	Responses					
Name	Completed	LateNoTransfer	LateWithTransfer	OnTimeNoTransfer	OnTimeWithTransfer	WIPNoTransfer	WIPWithTransfer
TransferBatching	100 of 100	19.55	13.61	20.07	26.88	2.92557	2.45989

The action item for this principle is:

Note that transfer batching requires additional material handling operations to move the smaller transfer batches. In a manufacturing setting, this might involve trips by a forklift truck, whereas in a healthcare setting, it might require more frequent trips between the clinic and the testing lab. These additional transfers will increase costs, but in the process improve performance. The value will depend on the tradeoff of added cost and productivity. This strategy is particularly effective in situations where downstream servers are bottlenecks.

Principle #17: Preventative maintenance may improve performance.

One area where variability can have a highly detrimental impact on system performance is server reliability. Unplanned server shutdowns can create havoc in operational plans. Preventative maintenance (PM) is a strategy for reducing the detrimental impact of server breakdowns by performing preemptive servicing to extend the time between server breakdowns. For example, if a tool that has a random time to failure instead of waiting for the tool to fail we might replace the tool before it breaks, thereby reducing the chance of an unplanned event. By scheduling preventative maintenance, we reduce this source of variability in the system and as a result may reduce WIP, increase throughput, and improve on-time delivery.

Although we do not know when a random server breakdown might occur, we can select the times for performing preventative maintenance. In many cases, we can schedule maintenance at off period times where we do not interfere with the normal production periods. However, even if preventative maintenance must be done during normal production, the server is typically shut down for a much smaller time than during a random failure. One reason for this is that planned maintenance can have all the resources (worker, spare parts, etc.) available and ready at the scheduled time, whereas these resources may take time to assemble for a random failure.

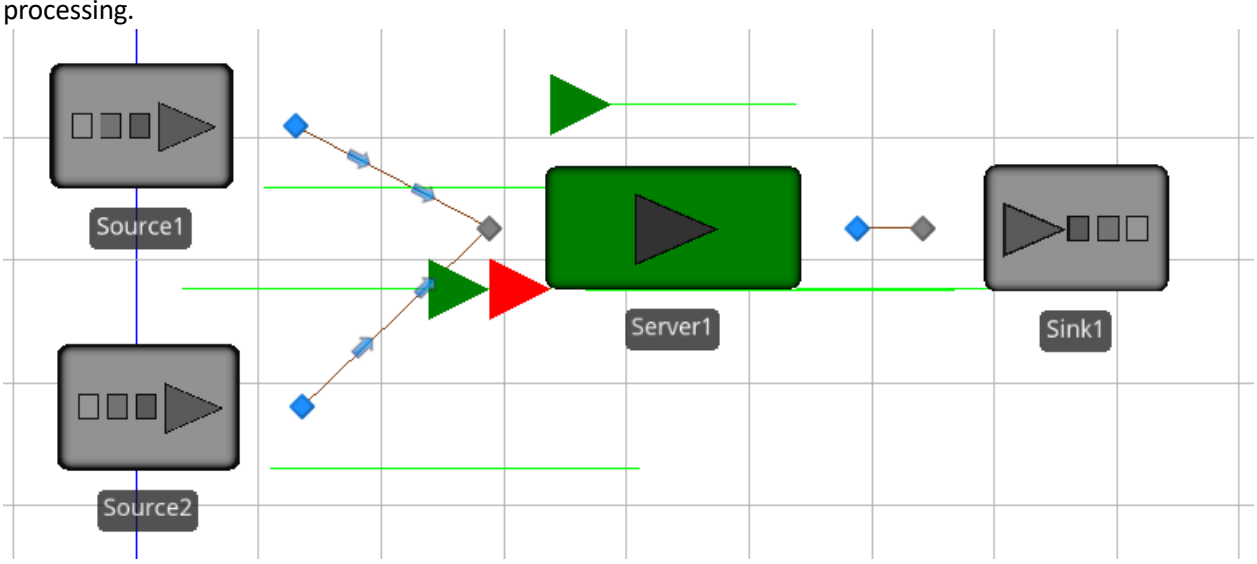
The impact of a preventative maintenance strategy depends on the details of the system; particularly the ability of the maintenance activity to extend the time between random failures. However, in many cases a preventative maintenance strategy can have a significant impact on performance.

To illustrate this principle, we model a single server system with random failures that occur with a triangular distribution with a minimum of 14, mode of 16, and maximum of 18 hours. If a random failure occurs, the repair time has a triangular distribution with a minimum of 2, mode of 4, and maximum of 6 hours. Entities randomly arrive to the system with an inter-arrival time that is exponentially distributed with a mean of .23 minutes, and are processed at the server with a random time that has a triangular distribution with a minimum of .1, mode of .2, and maximum of .3 minutes.

We compare two scenarios. In the first scenario we run the system “as is” with random failures. In the second scenario we take the server down every 6 hours to perform preventative maintenance for 15 minutes, and assume that this maintenance will double our time between failures to a minimum of 28, mode of 32, and maximum of 46 hours. In real life systems it’s not unusual for preventative maintenance to have a much more dramatic impact on time between failures. We also assume that preventative maintenance takes place during normal production periods, whereas in many real life systems we try and perform the maintenance during off-shift periods.

We model the preventative maintenance by introducing a new entity representing the maintenance activity that enters the system at source 2. Under the *Without PM* scenario we limit the number of these entities to 0, and assign the larger uptime between failures. Under the *With PM* scenario we set the limit on these entities to infinity and assign the smaller uptime between failures. This PM entity is given priority at the server and will jump ahead of any normal entities that enter the system at source 1 and may be waiting for service in the input buffer for the server. The following is a snapshot of the model near the start of the simulation run. At this point the server is busy processing a normal entity (green) and a preventive maintenance entity (red) is at the head of the input buffer waiting for

processing.



The results for this simple example is summarized below. The preventative maintenance strategy for this example results in significantly less WIP (95%) along with a sizeable increase in throughput (20%). This improvement is produced by reducing the impact that random failures have on our system through a preemptive maintenance program. How well a preventative strategy will work in your system will depend on the details of your system, but it's often possible to have significant improvements in production through a preventative maintenance strategy.

Scenario		Replications	Controls		Responses	
☑	Name	Completed	MaximumPMS	UptimeBetweenFailures (Hours)	WIP	Throughput
☑	WithPM	10 of 10	Infinity	Random.Triangular(28, 32, 46)	13.6823	6161.6
☑	WithoutPM	10 of 10	0	Random.Triangular(14, 16, 18)	209.953	5447

The value of a PM program is that it removes variation from the system. This is true for both manufacturing systems and service systems.

The action item for this principle is:

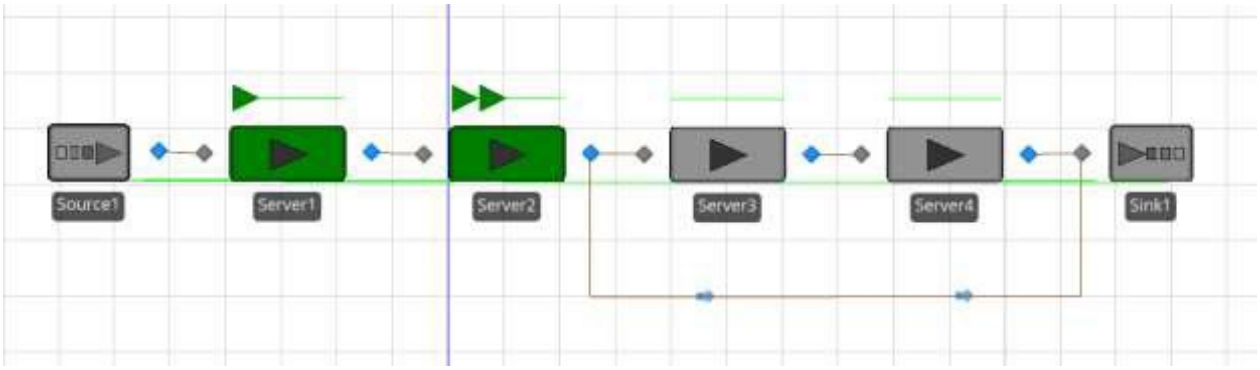
Principle #18: Reducing the number of process steps improves performance.

Reducing the number of steps in a process can reduce process variability, and so improve the performance of a system.

Basic probability theory states that, for a number of independent steps in sequence, the total processing time for all steps has a variance that is the sum of the variances at each step. If we consolidate steps without increasing the variability of the consolidated steps, we will reduce the variability of the total processing time since we are summing fewer variances.

To illustrate this principle, we model a scenario comprised of four steps in sequence, where each step has a random processing time with a triangular distribution with a minimum of .1, mode of .2, and a maximum of .3 minutes. Entities arrive to the system with a random inter-arrival time that is exponentially distributed with a mean of .25 minutes. Four servers process our entities in the system. We compare this to scenario 2, where we have consolidated steps 1 and 2 into a single step, and steps 3 and 4 into a second step, where each are independent with a triangular distribution with a minimum of .3 minutes, a mode of .4 minutes, and a maximum of .5 minutes. We added the means to arrive at the new mean of .4, but kept the same +/- .1 deviation from the resulting mean. To keep the capacity the same in our second scenario, we assign a capacity of two to each of the two servers, yielding a total capacity of four servers – the same as in the four-step sequence.

To model this system we use two different selectable routings. In the first routing, the entity goes to server 1, then server 2, and then travels to the sink. The second routing requires the entity to visit all four servers before traveling to the sink. By changing the routing, the server capacity, and processing time at the server, we can configure the model to each of the two scenarios. The following shows a snapshot of the model configured for the two-step scenario. Note that two entities are being processed at server 2.



The following table summarizes the results for both scenarios.

Scenario		Replications	Controls			Responses	
☑	Name	Completed	EntityRouting	ProcessingTime (Minutes)	ServerCapacity	WIP	Throughput
☑	Four Steps	10 of 10	Four Steps	Random.Triangular(.1,.2,.3)	1	5.85082	5728.3
☑	Two Steps	10 of 10	Two Steps	Random.Triangular(.3,.4,.5)	2	4.75556	5724

In the first scenario, any server failure stops the entire line from producing. The second scenario of two steps provides stability in the presence of server failures. Since there are two identical servers for each of the two steps, a single server failure does not stop production, so failures have little impact on the overall performance of the system. As expected, the two-step scenario has significantly less WIP with the same approximate throughput.

The action item from this principle is:

Principle #19: Decreasing the number of tasks in a complex activity improves performance.

In Principle 18, we demonstrated how reducing the number of process steps performed by multiple servers in a flow line or job shop can improve the productivity of the system. This principle is similar in that it relates to reducing the number of steps in a complex activity that is being performed at a single server. Reducing the number of steps to be performed in a complex activity will improve the performance of the system – even if the total task time is not reduced.

Any assembly operation is a complex activity, since each component to be added to the assembly represents an independent task. An example would be putting a bicycle together. An example in the medical field would be treating a patient in an emergency department for a laceration; the tasks would include cleaning the wound, numbing the area, stitching the cut, putting on an antibiotic, and then covering the area with a bandage.

Since a complex activity is a sequence of independent tasks, as in Principle 18, probability theory tells us that the variability of the activity time is the sum of the variability for each task in the activity. Hence, even if the total time for all tasks remains the same, having fewer tasks means less variability, as long as we don't introduce additional variability into the individual tasks.

Reducing the number of tasks in an activity involves redesigning the activity. In an assembly operation, it might mean having fewer components, or having fewer mechanical/electrical connections to make for each component. In an activity such as treating a patient for a laceration, it might involve combining two tasks into one; pre-medicated bandages might be used so that the antibiotic and bandage are placed on the patient as a single task.

Activity redesign to reduce the number of tasks can often produce smaller total cycle times by simplifying and eliminating tasks. However, even if the total cycle time remains the same, reducing the number of tasks yields performance improvements.

To demonstrate this principle, we model a single server performing a complex activity comprised of three tasks, where each task has a triangular distribution with a minimum of 1, mode of 2, and maximum of 3 minutes (scenario 1). Note that the expected time for this complex activity is 6 minutes, and we have entities arrive to the server randomly with an exponential distribution and a mean inter-arrival time of 6.5 minutes. We then compare this to a scenario where we combine the three tasks into a single task that has a triangular distribution with a minimum of 5, mode of 6, and maximum of 7 minutes. Although the new combined task is three times as long as each of the original individual tasks, it has the same +/- 1 deviation for the maximum/minimum around its mode.

The simulation results comparing these two systems are summarized below.

As we would expect the combined task produces less WIP with the same throughput.

Scenario		Replications	Controls	Responses	
☑	Name	Completed	ActivityTime (Minutes)	WIP	Throughput
☑	Individual Tasks	100 of 100	Random.Triangular(1,2,3)+Random.Triangular(1,2,3)+Random.Triangular(1,2,3)	6.29803	2354.55
☑	Single Combined Task	100 of 100	Random.Triangular(5,6,7)	5.9639	2346.08

The action item for this principle is:

Redesigning a complex activity such as an assembly can yield not only less variability as a result of fewer tasks in the activity, but also a smaller total cycle time. The combination of a smaller cycle time and less variability can provide even greater improvement.

Principle #20: Slack-based rules can improve on-time delivery.

In a make-to-order environment, the most important KPI is often on-time delivery. The shift to custom made products, zero or low inventories, and just-in-time production has made the ability to deliver by a specified date a critical element of competing in today's market. Fast and reliable delivery is an important and recognized value that can support premium pricing for products and services.

On-time delivery is achieved when all entities in the current planning period are processed by the production system by their specified due date. When one or more entities are completed after their due date, we have a late production schedule. The quality of a late schedule can be measured by the number of late entities, and the average lateness. Note that it is often possible to reduce the number of late entities at the expense of a large average lateness. Hence focusing solely on the number of late entities may not be an appropriate strategy when striving to achieve an on-time schedule.

This principle states that slack-based rules can improve on-time delivery. Slack is defined as the time remaining before the entity deadline less the remaining processing time required. For example, if an entity should be completed in 8 hours, and 3 hours of processing time remains, the remaining slack is 5 hours. Note that the slack does not account for waiting time or availability of resources; hence, a positive slack does not guarantee that an entity will be on time, only that it's possible to be on time. On the other hand, negative slack indicates that an entity will be late.

One commonly used measure of slack is called critical ratio, which is the ratio of the time remaining before the entity deadline to the remaining processing time required. In our previous example, the time remaining before the entity deadline is 8 hours, and the remaining processing time needed is 5 hours, resulting in a the critical ratio of 1.6. A value of 1 indicates that the entity must start to be processed immediately to be on time, a value > 1 indicates that some slack is available, and a value < 1 indicates that the entity will be late. The critical ratio provides a normalized value that makes it easier to compare entities with small and long remaining processing times.

To demonstrate the benefit of slack-based rules, we will model a simple job shop with three entity types and three servers, and analyze the benefit of using the critical ratio rule to manage the flow of entities through the system Entities randomly arrive to the system with an inter-arrival time that is exponentially distributed with a mean of 1 hour. We limit the number of entity arrivals to 40, and run the simulation long enough for all 40 entities to be processed. This is important to insure that our KPI measures are not distorted by entities that remain in the system at the end of the simulation. The entities have a due date that is their arrival time to the system plus a lead time that varies by entity type. Each entity has a routing sequence that ends at a shipping station. The product mix, lead time, routing, and processing time is specified in the following table for each of the three entity types:

Type	LeadTime (Minutes)	Mix
1	600	40

Routing

	Sequence	Type	ProcessTime (Hours)
1	Input@Server1	1	Random.Triangular(.5,1,1.5)
2	Input@Server3	1	Random.Triangular(.5,1,1.5)
3	Ship	1	0
*			

Type	LeadTime (Minutes)	Mix
2	750	30

Routing

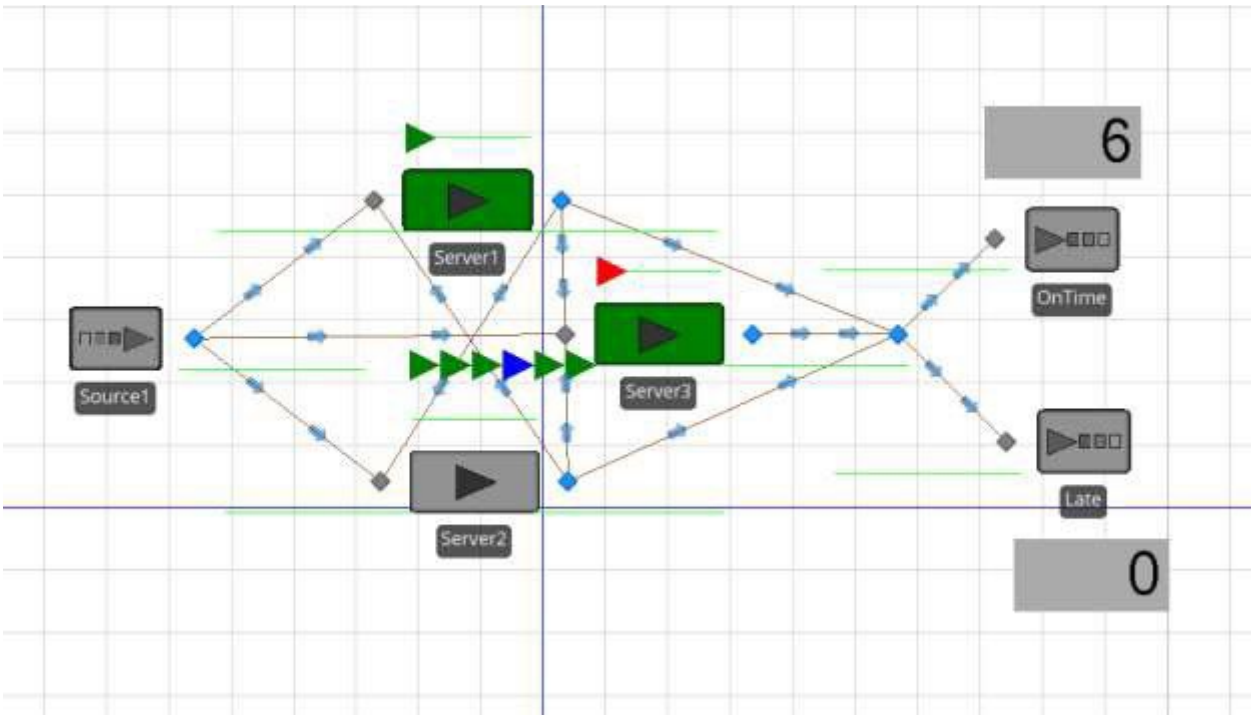
	Sequence	Type	ProcessTime (Hours)
1	Input@Server2	2	Random.Triangular(.5,1,1.5)
2	Input@Server3	2	Random.Triangular(1,2,3)
3	Ship	2	0
*			

Type	LeadTime (Minutes)	Mix
3	450	30

Routing

	Sequence	Type	ProcessTime (Hours)
1	Input@Server3	3	Random.Triangular(2,3,4)
2	Ship	3	0.0
*			

The following is a snapshot of the executing model for this system. Entities of type 1 are green, entities of type 2 are blue, and entities of type 3 are red. At this point in the simulation, servers 1 and 3 are busy with entities waiting in the input buffer for server 3, and server 2 is idle. The entities that complete on time are sent to the sink labeled *OnTime*, and the count shows that 6 have been completed on time. The entities that are late are sent to the sink labeled *Late.* None have been late at this point in the simulation.



In this experiment, we are able to change the dynamically evaluated expression that is used for selecting the next entity to process at the three servers based on the smallest value first rule. In Scenario 1, we specify this value as the critical ratio for the entity, so that entities with the smallest critical ratio are given priority. Scenario 2, the base case, uses the order of release, which is the time each entity entered the system. In the third scenario, we specify this value as the critical ratio for this entity, but only if the critical ratio is greater than or equal to 1; otherwise, we use the time the entity entered the system. In the third scenario, therefore, we are treating an entity that is going to be late as a "lost cause", and instead selecting an entity that is critical to work on but can still ship on time. Once an entity is late in this scenario, it will be continuously passed over for entities that still have a chance of being on time. We would expect this last rule to reduce the number of late jobs, but possibly at the expense of a larger average lateness.

The results are shown below. Based on the number of late jobs with this scenario, processing the entities in the order of release has slightly better results as processing them based on the most critical ratio rule. However, using the critical ratio rule reduces the average lateness by 40%, and also reduces the WIP. Scenario 3 using the modified critical ratio rule does much better than the other two rules in terms of number of late jobs, but has a significantly higher average lateness. Hence, this produces fewer late jobs, but the jobs that are late are much later.

Scenario	Replications	Controls	Responses			
Name	Completed	DynamicSelectionExpression	OnTime	Late	WIP	AverageLateness
Most Critical	10 of 10	Entity.Sequence.CriticalRatio	16.4	23.6	4.37...	3.94466
Order of Release	10 of 10	Entity.TimeCreated	17.8	22.2	4.9961	7.17584
Most Critical of Good Candidates	10 of 10	Math.If(Entity.Sequence.CriticalRatio >= 1, Entity.Sequence.CriticalRatio, Entity.TimeCreated)	29.8	10.2	5.30...	21.2264

There are many other measures of slack-based rules that can be used. Entity priority could be combined with the critical ratio so that low priority entities are allowed to be late to accommodate higher priority entities. The best rule will depend on the specific application, but incorporating slack (or critical ratio) as part of the rule can improve on-time performance.

The action item for this principle is:

Principle #21: Smaller batch sizes can improve on time delivery.

In a make-to-order production system, the enterprise and materials planning (ERP/MRP) system will group together items from multiple customer orders to generate production orders for the system. A single production order may be for a batch of entities that are tied to several different customers. In many cases, the entire batch is produced as a single order, and the due date for the order is the earliest due date of all entities in the batch. The principle states that having a flexible batch size can improve on-time delivery.

Replacing one larger batch with several smaller batches provides greater flexibility in arranging the sequence of work across the available servers. Processing one large batch ties up the server for the entire time required. Smaller batches space the production of the batch over the planning period, and we have greater flexibility in meeting due dates for all orders. Entities can be grouped within a given batch based on similar due dates, so that we avoid completing orders sooner than needed and as a consequence make other entities late.

One of the main drivers for combining entities into a single batch is to have a single setup for all entities in the batch. If we break up this batch into smaller batches, each batch may need a new setup. If the setups are large and occur on a bottleneck server, this may have a detrimental impact on performance. However if the setups are relatively small and/or can be done off-shift, or are done on non-bottleneck servers, we will typically gain much more in terms of schedule flexibility than we lose in terms of extra setups.

To demonstrate the benefit of smaller batch sizes in meeting on-time delivery, we will use a slightly modified version of the job shop model that we used to demonstrate our last principle on slack-based rules. In this variation of the model, we break each production order into a set of batches, where the number of batches is a control used for experimentation. We assume that there is no significant penalty from additional setups from multiple batches. We compare three scenarios with the number of batches being 1, 5, and 10, all using a critical ratio selection rule. The results are shown below, where the number of on-time and late orders and WIP are scaled by the number of batches for easy comparison.

Scenario			Replications		Controls		Responses			
☑	Name	Status	Required	Completed	DynamicSelectionExpression	NumberOfBatches	OnTime	Late	WIP	AverageLateness...
☑	Large Batch Size	Idle	10	10 of 10	Entity.Sequence.CriticalRatio	1	13.2	26.8	5.43872	7.00305
☑	Medium Batch Size	Idle	10	10 of 10	Entity.Sequence.CriticalRatio	5	17.68	22.32	4.80509	6.06844
☑	Small Batch Size	Idle	10	10 of 10	Entity.Sequence.CriticalRatio	10	19.29	20.71	4.03106	4.05603

As we can see having the greater flexibility through smaller batches improves on-time delivery and reduces WIP. The action item for this principle is:

As noted before this strategy works best in situations where setups are relatively short or occur on non-bottleneck servers.

Principle #22: Increasing capacity early in a schedule improves overall performance.

When operating a production system in a make-to-order environment on-time delivery is often a critical KPI. The ability to reliability deliver products to customers on or before a promised date can be "make or break" for a business.

In a typical production environment a collection of manufacturing orders are released to the factory at the beginning of a production period. This period might be weekly, monthly, or longer depending on the nature of the business. Each manufacturing order in the collection has a specified due date, and the job of the production manager is to manage the flow of work during the production period to ensure that all entities are processed by their due date. At the end of the production period a new collection of orders are then released to the production system.

The production manager's job is complicated by many disruptions that may occur in the production system. Machines break, workers call in sick, component materials arrive late, tasks take longer than expected, last minute orders may be added to the collection, a due date gets moved up, etc. All of these disruptions are challenges to meeting the required schedule of due dates without adding overtime into the system.

When overtime becomes necessary, the question that is faced is how much overtime is required, and where in the schedule should we add the overtime.

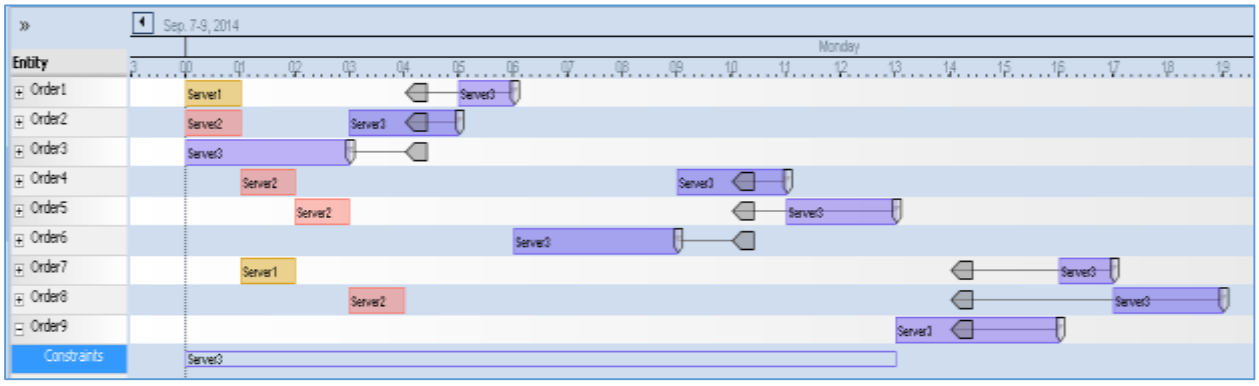
What often occurs in practice is that a specific customer order to a very important customer that is going to be late results in management approval for overtime to avoid the late order. The overtime is directed at improving the delivery of the important order, and often occurs late in the production schedule. This principle states that instead of using overtime late in the schedule to address specific late orders, it is better to use overtime early in the schedule to impact a wider range of orders. Another way of stating this is that it's more efficient to get ahead early in the schedule than to try and catch up late in the schedule.

To demonstrate this principle we will again use our same small job shop introduced in principle 20 with the same entity types and routings. The system is comprised of three servers and produces three different entity types based on different routings through the system. We will employ the modified critical ratio rule for maximizing on-time delivery.

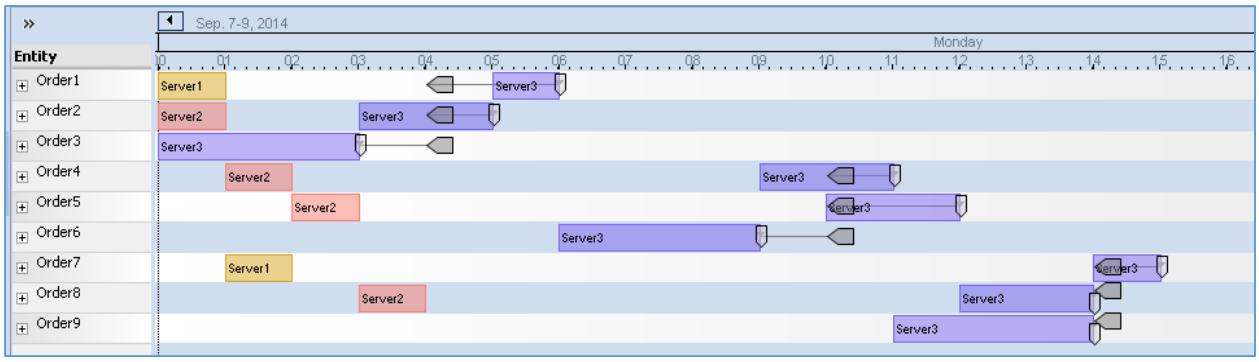
In place of randomly generating orders to the system we will instead simulate the flow of a specific list of orders, each with a specified release date and due date. The list of orders for our system is shown below, where our planning period is a single day:

Order	Entity Type	Release Date	Due Date
Order1	1	9/8/2014 12:00:00 AM	9/8/2014 4:00:00 AM
Order2	2	9/8/2014 12:00:00 AM	9/8/2014 4:00:00 AM
Order3	3	9/8/2014 12:00:00 AM	9/8/2014 4:00:00 AM
Order4	2	9/8/2014 12:00:00 AM	9/8/2014 10:00:00 AM
Order5	2	9/8/2014 12:00:00 AM	9/8/2014 10:00:00 AM
Order6	3	9/8/2014 12:00:00 AM	9/8/2014 10:00:00 AM
Order7	1	9/8/2014 12:00:00 AM	9/8/2014 2:00:00 PM
Order8	2	9/8/2014 12:00:00 AM	9/8/2014 2:00:00 PM
Order9	3	9/8/2014 12:00:00 AM	9/8/2014 2:00:00 PM

We can generate a planned schedule for this system by simulating the flow of entities through the model in deterministic mode, where the random processing times are replaced with the expected processing time. This generates a planned schedule for the nine orders processed by the three servers as depicted in the following Gantt chart, where the completion of each order is denoted by the symbol and the due date for each order is depicted by the symbol . The connecting line between these symbols indicates the amount of slack or lateness. As we can see in this case, only Order 3 is on time and all other orders are late.

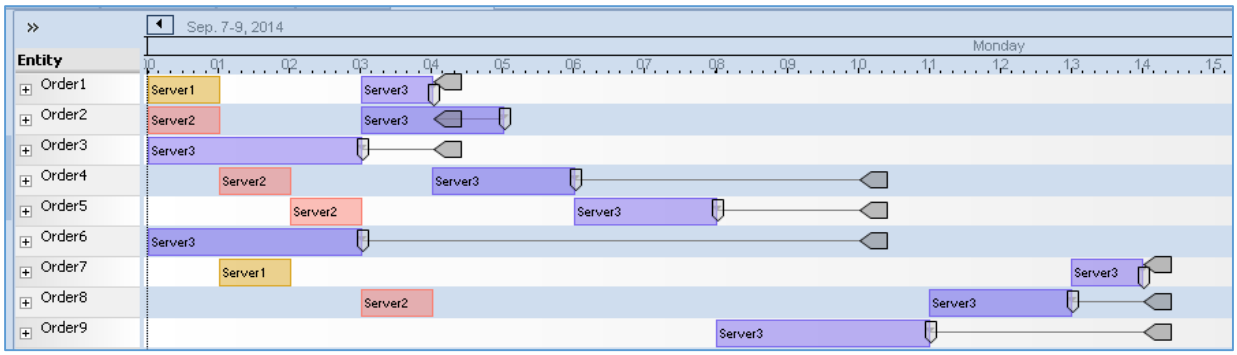


Let's assume that Order 9 is an important order that we must complete on time. We have expanded the row in order 9 above to show the non-value added time in the system for this order, and in this case we see that the order is constrained by Server 3, and this is the reason the order is late. To get this order completed on time, we can bring a second Server 3 online to process this order in time to complete Order 9 by its due date. If we make this server available from 10 am to 2 pm (since Order 9 is due at 2 pm) we get the following new schedule that completes Order 9 on time.



Note that by adding extra capacity to impact Order 9, we have completed Order 9 on time, and also Order 8 is completed on time. We have also reduced lateness for Order 7.

Our current principle states that if we are going to increase our capacity as we did here for a 4 hour block from 10am to 2pm, we would typically be better off moving that capacity earlier in the schedule. To see the impact of this let's examine the planned schedule based on a second Server 3 from midnight to 4am instead of from 10am to 2pm. The following shows our new schedule.



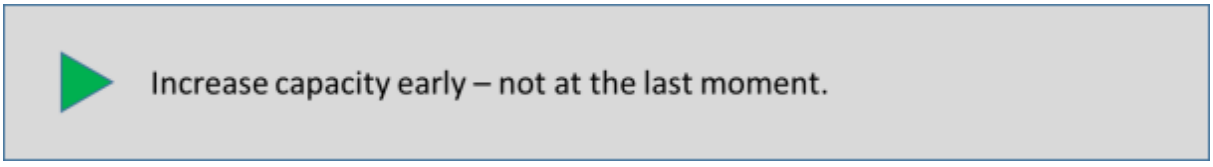
Although our total added capacity is the same (an additional Server 3 for a 4-hour period), moving this extra capacity to the start of the schedule dramatically improves the results. We now only have one late order (Order 2) and have substantial slack time for our critical Order 9.

To further demonstrate this principle, we run this model within an experiment comprised of four scenarios; no excess capacity, early extra capacity (midnight to 4am), midterm extra capacity (8am to 12pm), and late extra capacity (10am to 2pm). We will make 10 replications of each scenario with randomly sampled processing times for the three servers, and record the number of late orders. The results for this experiment are shown below.

Scenario	Replications	Controls	Responses
Name	Completed	WorkSchedulePattern	NumberLateOrders
No extra capacity	10 of 10	ContinuousWeek	6.3
Early extra capacity	10 of 10	EarlyOvertimeWeek	2.5
Midterm extra capacity	10 of 10	MiidOverTimeWeek	3.6
Late extra capacity	10 of 10	LateOverTimeWeek	5.2

As expected this experiment demonstrates that the earlier additional capacity is added, the greater the reduction in lateness. Adding resources near the end of the processing cycle only helps the last entities in the system, whereas adding resources near the beginning will clear congestion throughout the processing cycle. It's better to employ additional capacity to avoid falling behind as opposed to trying to catch up at the end once you have fallen behind. Managers should have a bias toward early action when it comes to increasing capacity through overtime, outsourcing, or other methods. This principle applies to both manufacturing and service systems.

The action item for this principle is:



Principle #23: Schedules are optimistic and over-promise.

Anyone involved in planning and scheduling tasks often finds that the work ends up falling behind schedule. This is often seen as the fault of the scheduler or of the people executing the plan, but in fact, it is the basic nature of deterministic schedules that they are optimistic and over-promise.

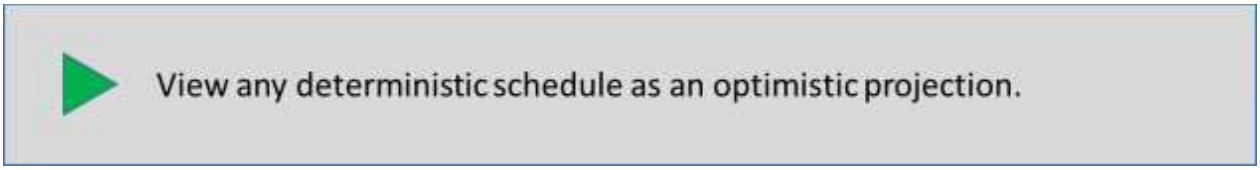
To generate a schedule, the first step is to replace all variable times with expected times. Since variation degrades performance (Principle 1), a schedule that assumes no variation ignores one of the primary drivers of system performance. In addition, unplanned events such as workers not showing up to work or machines breaking down are usually not accounted for, since we have no idea if and when they will occur in advance, and cannot include them in our schedule. Instead, we wait for these events to occur, and generate a new schedule that incorporates the unplanned event. This new schedule will typically be worse than our original schedule because it now includes the unplanned event. No matter how much effort and time we devote to generating the original schedule, the fact that the schedule is based on deterministic times and ignores unplanned events makes the schedule optimistic.

The inability to meet schedules often leads to tactics such as artificially inflating processing times, or building in fudge factors to make the schedule less optimistic. Of course, how big to make the fudge factor is a big guess, and the choice may generate a schedule that either over-promises on delivery or underutilizes the system.

The results from our previous experiment illustrate the optimistic nature of deterministic schedules. When we added a second Server 3 from midnight to 4 am and generated a deterministic schedule, we had only one late order (Order 3). However, when we simulated the schedule with variation in the processing times, we had an average of 2.5 late orders. This 250% increase in the number of late orders was due to the variation in the system.

The optimistic nature of deterministic schedules applies to both manufacturing and service systems. Regardless of your application area, you should view any deterministic schedule as an optimistic projection of what might happen. The schedule will typically only be met if things go exceptionally well and no unplanned events occur.

The action item for this principle is:



Principle #24: Risk-based planning and scheduling (RPS) improves performance.

Principle 23 identified the problem that deterministic schedules are optimistic and over-promise. As we run a real system, unplanned events occur that require us to reschedule the work, and the real schedule unfolds. The real schedule will often perform much worse than our original schedule because it reflects the actual events and variation that occurs.

When our real schedule doesn't meet planning expectations, we are often forced to add capacity to the system in terms of overtime, subcontracting work, or other costly measures to get back on schedule. Principle 22 says that it is better to add capacity early in the planning period, but at this point, it is too early to know if our schedule is going to be met or not. A risk-based based approach to scheduling can inform you early in the schedule if the schedule is likely to be met, so that action can be taken early enough to provide the most benefit on the resulting schedule.

Risk-based planning and scheduling is a simulation approach to scheduling that incorporates variation and unplanned events to generate multiple schedules that can be used to compute risk measures on the original deterministic schedule. These risk measures include the probability of completing each order on time given the variation and typical unplanned events in the system. The scheduler can then use this information early in the planning process to decide if capacity needs to be increased, or other actions are needed. This allows these decisions to be made in time to have the greatest impact on system performance.

To demonstrate this concept, we return to the example in Principle 22, where we generated a schedule with an additional Server 3 added from midnight to 4am. The resulting Gantt chart shows the deterministic schedule with risk measures added.

Note that the target ship date for each order is colored red (high risk), yellow (medium risk), or green (low risk) and includes a percentage that specifies the likelihood of shipping the order on time. For example, our plan shows that Order 1 will ship on time, but our risk measures show that it only has a 43% chance of shipping on time. Order 8, which has positive slack, has a 64% chance of shipping on time.

Being able to model added risk measures gives us information at the beginning of the schedule period, so that action can be taken early in the scheduling process if the risk is too high. For example, if it's crucial that Order 8 be completed on time, we could add additional capacity at the start of the processing period to get the risk down to an acceptable level.

By properly accounting for variation and unplanned events in planning and scheduling using RPS, problems can be identified earlier. It's then possible to take action sooner when they are less expensive and more effective for improving on-time performance. This basic principle applies equally well to both manufacturing and service systems.

The action item for this principle is:

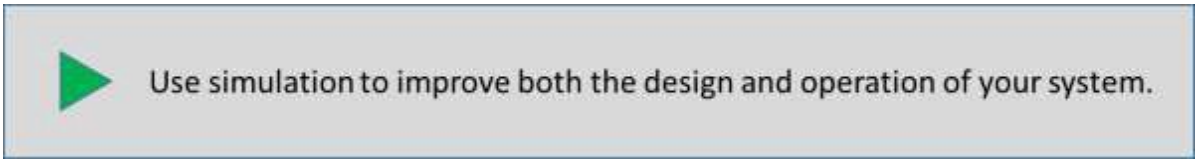
Principle #25: Simulation models can improve performance.

The preceding 24 principles all used simulation models of simple systems to illustrate performance improvements in throughput, WIP, and/or timely delivery. These principles can serve as guidelines for improving a wide range of systems. Of course, the best system to model in your evaluation of performance improvement ideas is your own!

Simio offers you the ability to quickly model complex production systems and fully capture the influence of randomness on the dynamic behavior of your system. You can then use the model to vary buffer sizes, introduce new entity types, modify server characteristics, etc., and see the impact of the changes on your system. The 24 principles we have discussed provide ideas for changes to consider in your system, and this principle simply states that testing those ideas in a model of your actual system can demonstrate the resulting performance improvements that you can expect.

It is important to realize that the impact of changes to large and complex production systems can sometimes be counter-intuitive. For example adding additional resources at a bottleneck station may yield little additional throughput because the bottleneck shifts to a different part of the system. By making proposed changes to a model of the actual system the impact of proposed changes can be fully explored.

The four case studies in the next section illustrate the breadth of application of simulation in both manufacturing and services for improving both the design and operation of complex systems, and illustrate our last principle:



Case Studies

Four recent case studies illustrate how simulation has helped management evaluate their systems and improve productivity. You can find additional information on these and other case studies at www.simio.com/.

The first case study, Nissan, used simulation to improve its automotive production facility in Barcelona, Spain. The second case study at Nebraska Medical Center helped improve the design of the campus-wide medical center. The third case study by Vantage Airport Group at Vancouver Airport looked at airport design, and illustrates the value of simulation for avoiding unnecessary capital expenditures. The last case study (BAE) applied simulation for risk-based planning and scheduling of a manufacturing facility.

Case Study 1: Nissan Motor Company

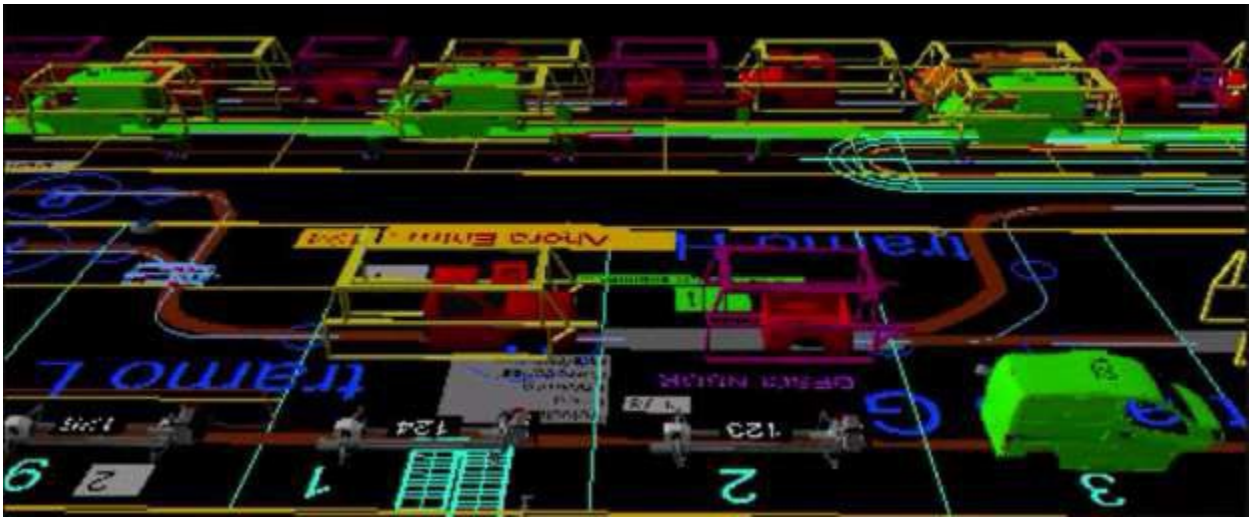
Nissan Motor Co., Ltd., is one of the world's leading automakers, and produces a diverse array of cars, minivans, trucks and SUVs. Their plants are located in every inhabited continent. To aid in the production of its new NV200 Van, Nissan Europe Engineering relied on discrete-event simulation software to validate the layout of assembly lines in the automaker's Barcelona plant.

Over the course of a project developed with the support of a local Simio partner, the team completed a number of models in a multi-phased approach aimed at obtaining rapid answers to specific issues related to deployment of the new lines. Among the challenges facing the team, several aspects were targeted as critical in terms of cost and time:

- Number of hangers required to meet projected vehicle throughput
- Validation of alternative product mixes
- Evaluation of the docking process for various converging lines (bodies, chassis) paced at different times

Simio's ease of use and ability to address the consecutive challenges faced throughout the project produced favorable results for the engineering team at Nissan Europe.

"We find that the use of objects to quickly build a working model of a plant layout very appropriate. Moreover, the possibility to add additional logic in the case of movement and synchronization of different lines through a set of instructions is easy to implement and expanded our capability to solve more complex problems," said Nissan Europe Project Manager José Vilar.



The 3D graphic objects allowed Nissan Europe to create a high quality representation of real elements found on the shop floor, such as vehicles, carriers, monorails, and others.

"We expect to continue the use of Simio to try to understand and resolve, through simulation, the potential problems of synchronization of the flow of vehicles on our other lines," Vilar said. "Simio is seen as an effective complement to our other engineering tools."

Most of the process improvement principles discussed in this book apply to a typical manufacturing system such as the Nissan factory. These same principles also apply to the suppliers of sub-components that feed our manufacturing system to enhance their just-in-time delivery of components parts to our line. Having a model of our factory allows us to test out these principles in the context of our manufacturing system. For example, Principle 1 (reducing variability improves performance) might suggest looking at adding automation or jigs/fixtures to simplify a specific step in the line and make it less variable. It might also suggest reducing the number of product variants by standardizing on fewer option packages to be processed on the production line. Regardless of the parameters, we can use the model of our factory to test out all of our process improvement alternatives.

Look here for more information and an animation: http://www.simio.com/case-studies/Nissan/

Case Study 2: The Nebraska Medical Center

The Nebraska Medical Center saw an opportunity during the programming phase of their future Comprehensive Cancer Center to rethink the surgery platform campus wide.

The consulting firm HDR was engaged to model future state scenarios in order to compare and identify situations in which variation might have the greatest impact on operations and facility planning. Simio served as a subcontractor for the project.

The simulation study produced a well-tested projection based upon the actual operations (including variation) at Nebraska Medical, as well as forecasted volumes analysis. Specific process improvements included the following:

1. More efficient use of the operating rooms and their staff during both peak and non-peak hours,
2. Better utilization of equipment-specific rooms, resulting in a need for fewer total operating rooms,
3. Reduced surgeon travel, in both distance and frequency,
4. Reduced patient travel distance, and
5. Reduced turnover times by services by patient type.

The value of this model was summarized by Matthew A. Mormino, M.D., Orthopedic Surgery Residency Director at Nebraska Medical:

"[it was] nearly impossible to make an informed decision without analyzing the various modeling scenarios provided by the [HDR] simulation [in Simio]. We now feel confident that our recommendations are well thought out and reflect our likely utilization of these operating suites."

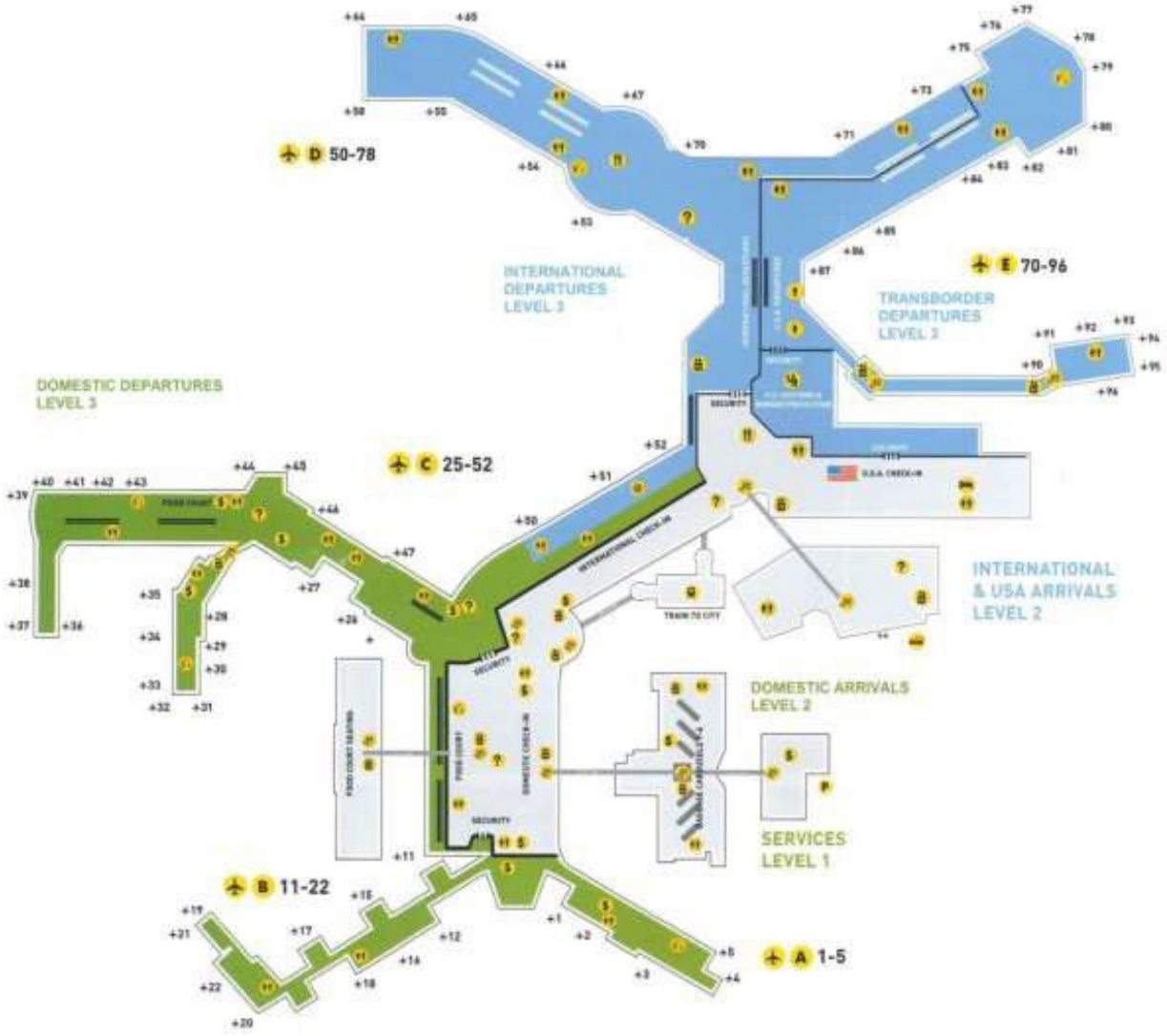
Nearly all of the principles discussed in this book can be applied in healthcare applications such as the Nebraska Healthcare model. For example, Principle 1 might focus the healthcare system process improvement team on reducing process variability. This might be done by universally applying best-

practice protocols by all caregivers to standardize procedures and methods. Although the motivation for standardized best-practice protocols is sometimes driven by a focus on quality, it has the added benefit of reducing variability, and thereby improving performance. Many healthcare systems also have very expensive resources such as high-end imaging equipment or operating rooms. Principles 9 and 10 on bottlenecks would suggest looking at buffering these resources as possible improvements. For example, ways to improve handling of pre-op and post-op patients might be found to better buffer the operating room. We can use our model of the healthcare delivery system to assess and measure the impact of any proposed changes.

Case Study 3: Passenger and Baggage Flow at Vancouver Airport

The Vancouver Airport Authority developed a simulation model to analyze the flow of passengers and baggage that arrive, depart, and connect through Vancouver International Airport (YVR). The modeling approach is part of the airport's initiative, efficient YVR (eYVR), aimed at optimizing airport operations.

The purpose of the simulation was to assess demand at various processing points, such as check-in, security screening, customs declaration, and baggage claim, and in turn determine the capacity required to meet defined service levels (LOS) with stated maximum wait times. Using an extensive database of passenger and flight profiles, the model was designed to forecast passenger demand and determine capacity requirements using interchangeable flight schedules.



Process results provided information on the following:

- Utilized resources required to meet the LOS
- Wait times (average, maximum, and standard deviation)
- Queue lengths
- Periods of time above the LOS threshold
- Success rate.

The model continues to generate results that are used by various airport contractors, including passenger screening, baggage handling, border protection agencies, and customer service.

Mike Lazzaroni, Senior Planning Analyst, at Vancouver Airport Authority, summarizes the use of simulation in airport design as follows:

Airport simulation models, although quite complex and time-consuming to build, provide invaluable insight in the planning and operating of terminals. When building new terminals or expanding existing ones, simulation models can determine capacity requirements and provide guidance in scoping a project. In situations where increased capacity is required, simulation models can be used to re-engineer processes in order to make them more efficient. In such cases, the refining of airport processes can lead to millions of dollars in savings as a result of deferred capital costs. One such example at Vancouver Airport has been the introduction of kiosks for the customs declaration of returning residents. The need for more capacity in the customs hall was driving a project to expand the terminal and create a ripple effect such as the relocation of aircraft gates. The use of kiosks provided the added capacity, reduced strain on the facility, and saved close to $100 million.

Finally, simulation models are also invaluable in the day to day operations of the airport. They are useful in assessing the required amount of staffing to run smooth operations with wait times that are contained within established levels of service.

Many of the principles discussed in this book have obvious application to airport applications such as the Vancouver Airport model. For example, the principles would suggest using a single line to feed multiple check-in counters, as well as a special line for handling baggage check-in for passengers with tickets to give priority to shorter processing time customers. A critical bottleneck at most airports is the gate used for loading and unloading airplanes; hence, our bottleneck principles (9 and 10) suggest looking at ways to buffer/improve both the passengers deplaning on arrival to the gate, and passengers boarding the plane for departure. The deplaning process might be enhanced by designing the gate to have a larger, dedicated departure lane that is less congested and easier/faster to exit. The boarding process might be improved by having all tickets pre-checked and scanned, and all gate-checked bags completed before the boarding process begins. These types of proposed changes might require design changes in the gate area that have positive or negative impacts on the system as a whole. A simulation model can be used to test out the impact of these or other changes on gate turn times.

Case Study 4: Forecasting Production Resources at BAE Systems.

Defense contractors need to reliably plan and predict production resources to meet the military's needs on time and within budget. Contract managers seek more effective production resource risk mitigation methods. They demand accurate and timely key risk indicators (KRIs) for materials, labor, and equipment.

BAE Systems (BAE) used Simio's Enterprise Edition software that includes a patented risk-based planning and scheduling (RPS) functionality. The feature integrates traditional planning and scheduling features with stochastic modeling for risk analysis.

Simio's scheduling software provided planners and schedulers with a customized interface for generating schedules, performing risk and cost analysis, investigating potential improvements, and viewing those parameters in 3D animations. Gantt charts made it easy to see the timing of processes, and to explore how changes in equipment or employees affect that timing.

Simio users such as BAE can run simulations whenever system downtime, employee availability, or other factors change, resulting in a "finger on the pulse" awareness that enables quick adjustments and aids confident decision-making.

Simio Enterprise Edition software with scheduling functionality helped BAE Systems meet production deadlines. BAE now uses Simio to meet a variety of forecasting and scheduling challenges, including decreasing overtime, developing training goals, preparing proposals, and evaluating capital investments.

Simulation-based scheduling applications such as BAE Systems can benefit from a number of the principles discussed in this book. Although many of the principles focus on improving system design, several are specifically targeted at operational improvement. One of the most important of these is the value of identifying operational issues such as risk of tardiness at the beginning of the planning period, and the added benefit of taking actions such as overtime action as early as possible in the planning period.

You can find additional case studies at https://www.simio.com/case-studies

Glossary

Bottleneck –A resource or step in a process that causes the entire process to slow down or stop. The bottleneck is often an expensive resource that cannot be expanded without significant investment.

Cycle Time –The period required to complete one cycle of an activity, or a function, job, or task from start to finish.

Discrete Event Simulation (**DES**) – A type of simulation modeling that deals with events that happen in a chronological sequence.

Entity –The item or person that moves through a process, such as a work piece in a factory, passenger in an airport, or patient in a health clinic.

Key Performance Index (KPI) –An important business measure for judging the performance of a system, such as throughput or work-in-process (WIP).

Lean Production –A set of techniques and tools for process improvement that seeks to eliminate any aspect of production that does not contribute directly to the creation of value.

Material –A supply of items or other assets which are required and consumed in a processing step. Examples of materials are nuts and bolts in a manufacturing line, drugs in a healthcare system, or fuel at an airport.

Process –A sequence of steps for performing some action to arrive at a particular end result– such as performing a service or producing a part.

Processing Time Variation –A phenomenon where the time required to perform some step in a process is different each time that step is performed.

Production Batch –A collection of entities that are processed as a group.

Resource –A device, person, or a collection of devices and people that is required to perform some activity on an entity in a step in a process. A resource can be a single person or machine, or a combination of people and equipment – such as an MRI and its operator.

Risk-based Planning and Scheduling (RPS) –Risk-based Planning and Scheduling is the dual use of a simulation model to plan and schedule a system, and to assess the associated risks with that plan.

Simulation Replications –A simulation model run multiple times with different random values to estimate system performance in the presence of variation approximated by using statistics across replications.

Six Sigma –A set of techniques and tools for process improvement that has five phases: Define, Measure, Analyze, Improve, and Control.

Throughput –The number of entities produced or processed in a specified time.

Transfer Batch –A collection of entities that are processed as a batch, and therefore transferred from one location to another as a group.

Unplanned Event –An event that occurs where the time of the event is not known in advance, such as the breakdown of a machine.

Work-in-Process (WIP) –The number of entities of a specific type that are currently being processed within the system.

The Next Step

The principles in this book have been illustrated with Simio simulation software. To see how these principles can improve your own operations, take the next step! Produce a model and 3D animation of your own facility to see how design or flow problems might be creating delays and poor performance for you.

There are many compelling reasons why Simio is your best choice of simulation software. It is based on ideas from an industry leader with over 30 years of modeling experience, and implemented using the latest advancements in technology. Its Microsoft™ ribbon-based interface helps you to get up to speed quickly. The included library of objects lets you get results twice as fast as other leading products. And its patented processes remove the brick walls that could prevent you from modeling your system to the required accuracy – all with no programming required!

Full product information is available at www.simio.com.

Watch our sample application videos on the home page to see what people in your industry have done. Then get started by downloading the evaluation software. Don't miss all the helpful tools on the support ribbon, including free e-books, free training videos, and much more to help you get results sooner.

To contact Simio email sales@simio.com or call Simio toll-free at 1-877-297-4646 (or direct at 1-412-528-1576).

www.simio.com
sales@simio.com
+1-412-528-1576