A Complete Guide to Lean Agile Project Management

Playbook

This Guide walks you through Lean Agile project management and why it's the business opportunity of the century including:

What is Lean Project Management? Page 3

What is Lean vs Agile Project Management? Page 3

What is the goal of Lean Agile Project Management? Page 4

Lean Agile Project Management principles and methods that work Page 4

How do you measure ROI for Lean Agile Project Management? Page 5

Which Lean project management initiatives have the biggest ROI? Page 6

Visual work management Page 6

Effective management of resource utilization Page 12

Push vs. Pull work management Page 20

Clear and correct priorities across projects Page 27

Shared Project Buffers Page 38

Decentralized Planning Page 48

What is Lean project management?

Lean is any process improvement that substantially improves the throughput of your system, sustainably.

So, it is with this definition in mind that we turn our focus to hardware development and find the ways to ***substantially*** improve throughput in these systems, ***sustainably***.

What is Lean vs. Agile project management?

Lean focuses on minimizing waste to maximize efficiency. Similarly, Agile focuses on minimizing unnecessary documentation, and maximizing communications through continuous delivery of product and daily meetings.

Both Lean and Agile recognize the need for flexibility in project management and adapting process to fit the size and scale of the project.

What is the goal of Lean Agile project management?

The common goal is to get more out of the system simply by doing things differently -- by working smarter, not harder.

We know that not all Lean techniques are 'smart' in all systems. It's clear that manufacturing systems are different from product development systems and that some Lean manufacturing techniques don't apply to product development.

Similarly, software development systems are different from hardware development systems, which are different from services oriented systems like health care, etc. These systems are different enough that good solutions in one can be detrimental in another, even if they share some things in common.

Lean project management principles and methods that work

At Playbook we have tested many different approaches to project management for hardware teams. Here are the underlying principles which drive success in hardware projects.

Visibility and manageability of resource queues

Effective management of resource utilization in the highly variable world of product development

Pull vs. push in work management

Reduced information flow batch sizes and faster feedback

Clear, correct priorities across projects

Shared project buffers

Decentralized project planning and management

How do you measure ROI for Lean project management?

System throughput in a hardware development system is usually measured in profit (per unit of time).

The ROI of an improvement is can be looked at using this simple equation.

Return on Investment (ROI) = how much it improves throughput divided by how much it costs.

For example, a process change like an investment in knowledge capture might cost $100k/year in labor and tool expense, in some high-innovation environments where everything changes a lot.

After considering the cost of delay incurred by stopping to capture the knowledge, maybe it only generates $400k/year in increased profits (after the $100k cost is deducted) for a 400% ROI. If we could make more than 4-1 hiring more resources and doing another project the old way, then that process change isn’t very Lean.

What is Lean, to me, are process improvements that have a greater ROI than what we could make on another new product.

Which Lean Agile project management initiatives have the biggest return on investment?

So, which of the many schools of thought or potential process improvements deliver a high ROI in typical hardware development systems? Well, according to our analyses, in most companies, the big two process ***improvements with the highest ROI are Visual Work Management and Project Risk Management***.

Visual work management: Visibility and management of resource queues

A fundamental principle of Lean is smooth, continuous flow. Flow is achieved by eliminating turbulence caused when the flow through any process is stopped because a resource is busy doing other things. Such interruptions can cause the flow of work to stop, resulting in work queues.

The following are a few examples of queues in product development:

1. An engineering change order waiting to be reviewed and approved.
2. Parts waiting to be inspected.
3. Information waiting in someone's head to be shared with someone that needs it – e.g., What are the test results?

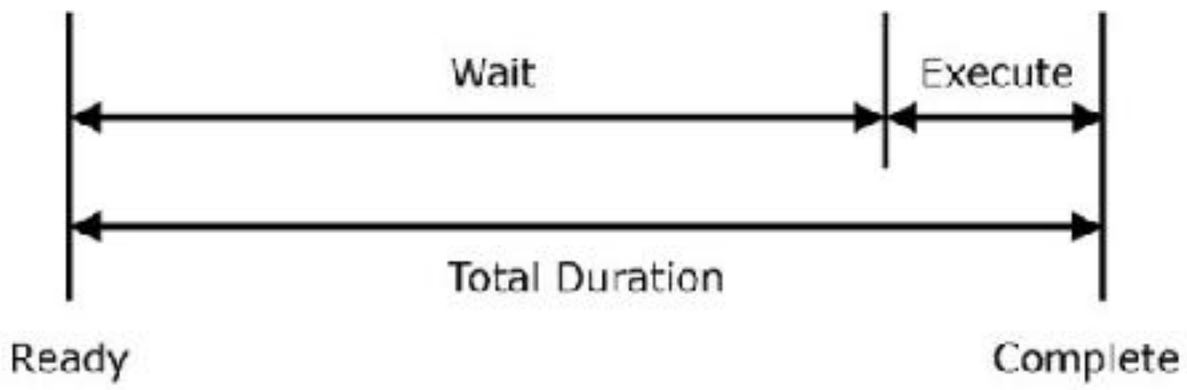
The duration of time when something is ready to be worked on, but is waiting for someone to work on it, is called "wait time" or "queue time." The duration between starting a task and completing a task is called "execute time." In traditional product development, wait time is typically much greater than execute time.

For example, a change order often sits in an approver's inbox for days, when it only takes a few minutes to review and sign. These queues and delays build up across the whole system of tasks and resources, from the beginning of a project to the end, and make our projects take much longer.

Total Duration = Wait + Execute

The longer the total duration of each task, the longer each project takes. Queue time is a big component of the total duration of each task, and the overall lengths of our projects.

Figure 1: Total duration



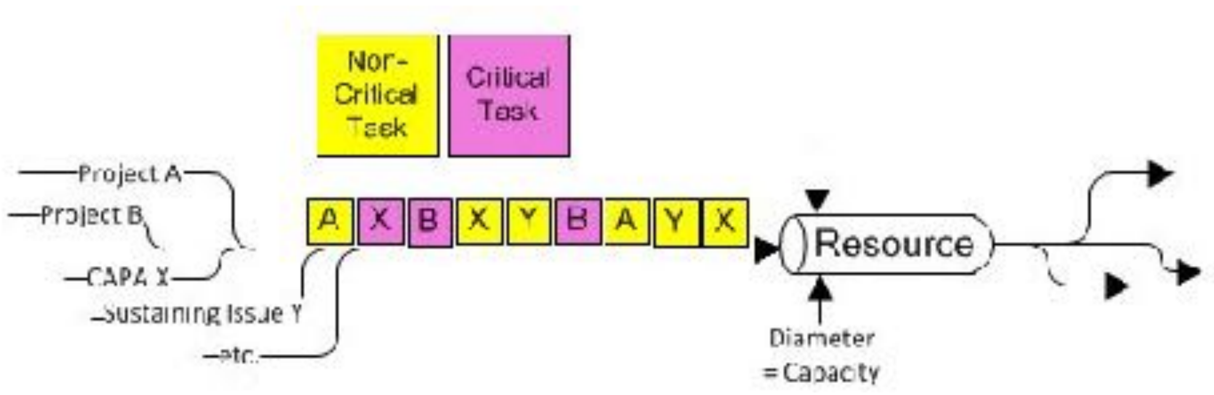
That is NOT to say, and I cannot emphasize this enough, that we should start executing each task as soon as it arrives just to reduce queue time. This is often more detrimental to the throughput, as multitasking easily extends the time to execute well beyond what we save in wait time. We will discuss this more when we get to the post about pull vs. push.

However, wherever we can reduce queue time without extending execute time, we reduce the duration of our project. One of the keys to accelerating projects is finding ways to do this. The first step is to make the queues visible.

The ability to see queues is critical

For the sake of this and future discussions, I'll present a simplified view of a resource in a typical hardware team, where there is a mix of projects currently being executed.

Figure 2: Resource Queue



The diagram above focuses on a single component of our system – a single resource. The work is represented as discrete critical and non-critical tasks in the form of stickies (the yellow and pink boxes). Tasks sit in the resources queue until the resource starts to work on them. They are then placed in the "Resource" pipeline of active work.

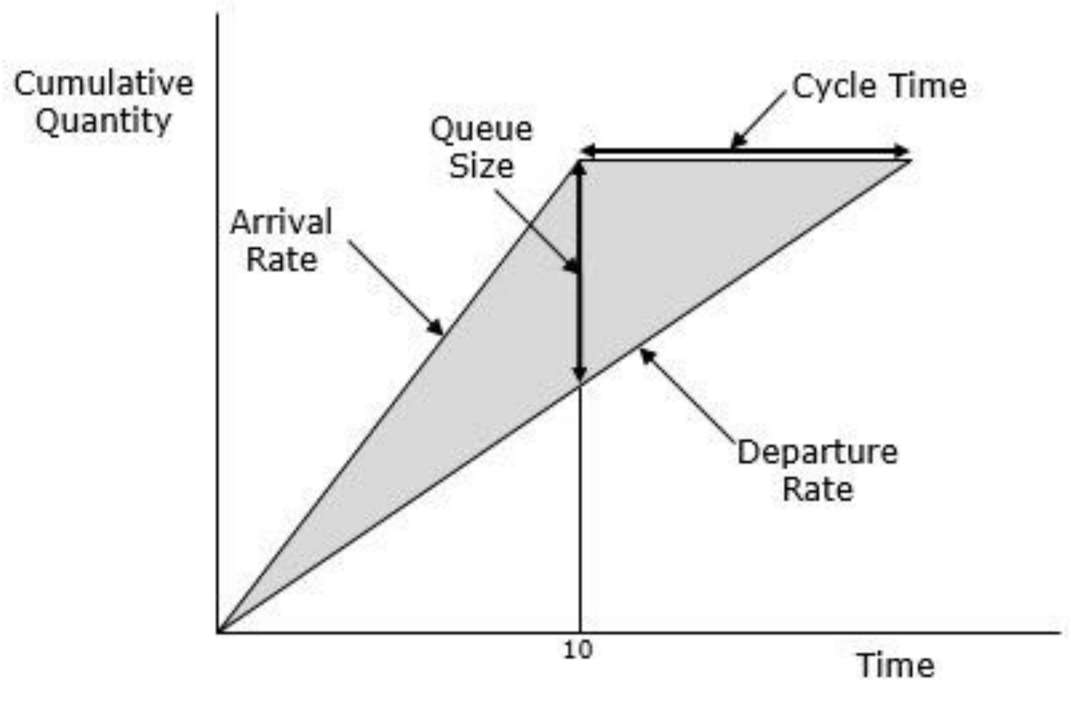
In later posts we will enhance this diagram and look into the pipeline, as well as represent more real life conditions, but we have started the discussion with this simplified view to make one very important point: the real priority of each task varies. We will discuss priorities more in a later post. For now, we'll keep it simple and note that some tasks are 'critical' and some are not. 'Critical' means that a delay of the task will probably cause a delay in the project.

Every day a critical task sits in a queue is another day of project delay.

Again – this statement assumes that the resource doesn't start working on the tasks as soon as they arrive and multitask just to get them out of the queue. Multitasking causes task duration to get much longer while tasks sit in the resource pipeline.

Queues can also be visualized using a Cumulative Flow Diagram (CFD).

Figure 3: Cumulative Flow Diagram



For example, let's say this CFD represents drawings entering document control for release. If the arrival rate is 2 drawings / day and the departure rate is 1 drawing / day, the 20th drawing, which appears in the queue on day 10, will have 9 drawings in front of it. It will wait in the queue for 9 days and be completed 10 days after arriving. We can predict this by seeing the queue.

As the CFD shows, queues are a leading indicator of the completion date of each task, as opposed to the lagging indicators of missed deadlines and project schedules published with the completion dates moved later. When we can see the queues, we get information early enough have A LOT more opportunities to keep the right things moving. When we can't see the queues, delays are unavoidable, which is the world we live in without Visual Work Management.

How do we minimize queue delays?

The good news is that queues, if they are visible, can be effectively managed and doing so will accelerate project execution. In order to clear a queue, we can use a combination of four effective strategies:

1. Slow the arrival rate by throttling demand
2. Increase departure rate by increasing the availability/capacity of the resource
3. Move some work to a more available resource
4. Remove the work entirely – just eliminate and/or cut work from some tasks.

Without some visibility of the work and the queues, how can we even hope to execute these time-saving measures quickly enough to avoid delay? We can't, not compared to how effective we can be when we can see the queues.

Note – that ‘just work faster’ and ‘apply more pressure’ are not listed as options for increasing the departure rate. We cannot clear a queue by increasing the pressure to force the work through the pipes. This directive often backfires by encouraging people to multitask and take too many risks or make more errors. We will discuss this more in a follow-up post on push and pull.

Besides the clearing strategies above, the other tactics to minimize queue time prevent the queues from getting long in the first place. I will summarize these methods here, as we will be diving into these in later posts.

1. Reduce resource overloading
2. Reduce variability in arrival rate and work amount
3. Reduce batch sizes (those stinking batches again)
4. Limit Work In Process & multitasking

What else is important to see in the queue?

There are other important aspects to good queue visibility in hardware teams, which most other formats and tools miss. First is the importance of seeing the queues, not only on a task basis, but also in the amount of work in the tasks. Three tasks in a queue could be three days of work, or thirty days of work.

Unless we can see the amount of work in the tasks, completion dates are difficult to determine and we lose the ability to know when to execute a queue-clearing measure. To make matters worse, inaccurate expectations for task completion dates too often result in additional push from management, which causes a number of problems we will discuss in the push vs pull post.

Additionally, especially with physical boards but even in digital sticky tools, seeing the work in a resource’s queue across *all* activities is difficult. If the tool does not enable us to manage sustaining engineering, CAPAs, early concept design and learning phases, and even training, annual reviews, and any other tasks which are in our queue, we simply can't manage the queues as effectively.

Finally, queue visibility is enhanced by granulizing the work well. Rather than one big, multi-week task with a lot of hidden sub-tasks, we benefit by granulizing and showing the sub-tasks when we can. There are many good reasons for task granularity, and making the resource queues more visible and manageable are only a couple.

However, to get good granularity and effective queue management, the people who know the work must be the ones to plan the work. To achieve optimal queue manageability, the tool must be easily usable by the whole team. It can't just be the project manager's tool.

Effective management of resource utilization

Resource loading has an enormous impact on queue time

In the previous post, we discussed briefly that resource loading has a high impact on queue time, and one way to minimize queue time was to increase the capacity of our resources. This is probably not surprising. We have all experienced many examples of this, such as Black Friday, rush hour, airport security, and the post office at lunchtime in December. When there is too much work for the resources who need to process that work, wait times get very long.

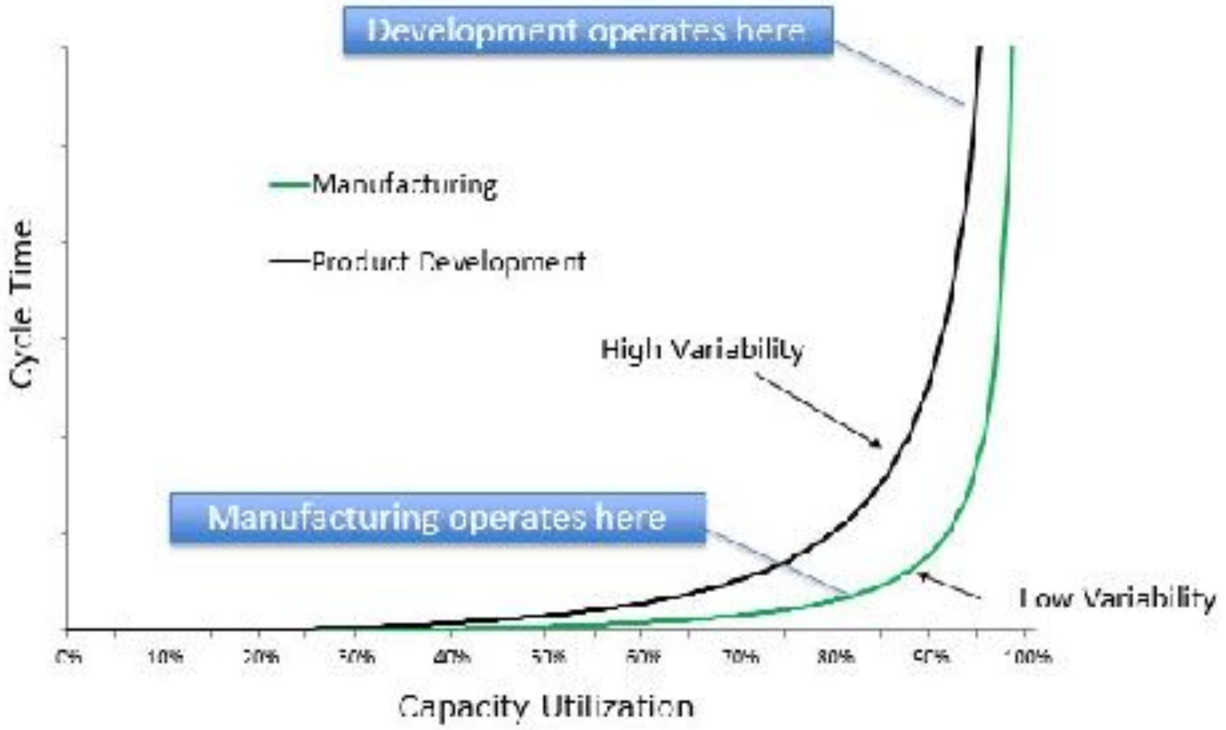
Figure 4: Rush hour traffic

The amount of time it takes to get through a system increases exponentially as capacity utilization (system loading) increases. One system we can use as a reference is the manufacturing floor. Ask a manufacturing manager how highly they load their production system and most will say they plan their machines to operate 80 to 85%. In other words, about an hour and a quarter of every day is left unallocated.

Why leave the production tools so much extra time? Because, even in low-variability environments like manufacturing, ‘stuff’ happens (defects, rework, unplanned maintenance, etc.). (variability) If you load the system too fully, these issues can bring the production line to a screeching halt. In order to keep the system flowing, we need a little unallocated capacity.

So how much unallocated capacity do we leave our development resources to keep the development system flowing quickly? Most often it is much higher than 80%. It may start out early in a project at 80%, but soon thereafter ‘stuff’ happens, the schedule starts to slip and the resources are loaded back up to 100% trying to make up for the slip. Ask the resources themselves and most will say they are loaded well above 100% even at the beginning of a project.

Figure 5: Impact of Resource Utilization and Variability on Project Duration



As figure 5 shows, in systems where variability is higher such as product development, we need even more unallocated capacity, not less. This overloading of the development system is a chronic issue which has an enormous impact on the lengths of our projects.

There is another analogy we can use to describe this issue – a car race. When we plan our resources to high levels of utilization we theoretically have a more ‘efficient’ system, with less wasted money on potentially unused capacity. But ‘efficiency’ is slow, which costs a lot in terms of the cost incurred by delaying a product to market (Cost of Delay). We can’t win the Product Development race by driving an efficient 4-cylinder Honda that gets 40 mpg. We win it by driving the fast Ferrari that gets 10 mpg, and we make a lot more money in the process.

Speed is more profitable than efficiency.

One of the keys to Lean Manufacturing is to reduce variability. However, variability is absolutely necessary in product development to innovate. Customers want to buy innovative, new products, and with innovation comes uncertainty (risk, unexpected defects, rework, etc.) and variability. Simply stated, to add value there must be variability.

Variability causes the duration curve to rise sooner, so for faster projects we must operate the product development system with even more spare capacity. Resource Utilization must be planned at 70% or even lower for high-innovation/ high-uncertainty projects. This drastically reduces the wait time on each task, and, in product development, also reduces execution time.

Resource loading has an enormous impact on execution time, too.

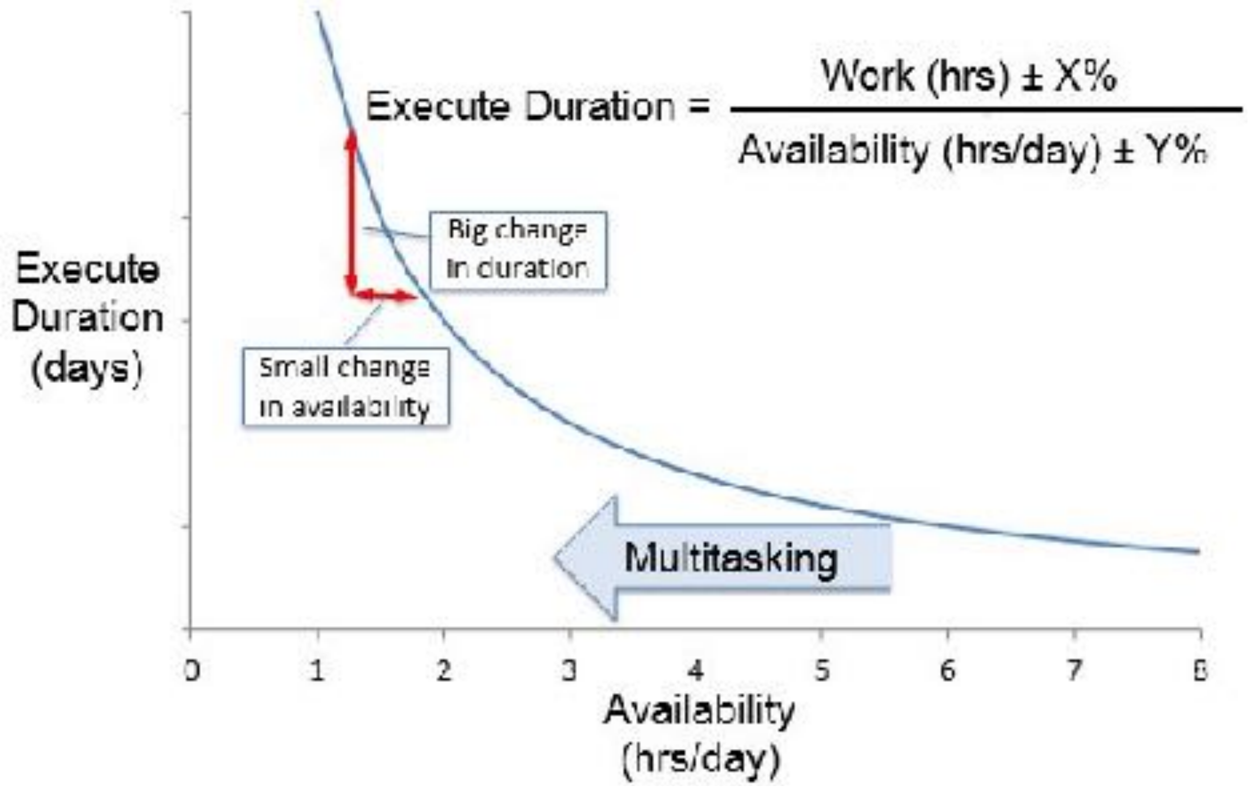
Recall from the previous post that the total duration of each task, from the minute it arrives in the queue to the minute it is completed, can be calculated using this simple formula:

Total Duration = Wait + Execute

Execute time for most tasks has a very simple formula as well.

Work is the total hours of work required to perform the task. Availability is the amount of time a resource has, per day, to execute that task. For example, a 12 hour task performed by a resource with 3 hours a day to work on it will take 4 days to complete.Execution time grows exponentially when our product development resources are overloaded because they are forced to multitask, which decreases their availability.

Figure 6: Multitasking Increases Duration AND Reduces Certainty



When there are are too many active projects and activities, there is always another meeting to attend, email to read, or person knocking at the door to discuss something – all of which reduce our resource's availability to get their work done. When there are too many tasks to work on, resources tend to multitask across several of them which further decreases their availability to each task. The result is much longer execution times for those tasks.

Not only are the task durations longer, they are less certain. As shown in Figure 2, when our resources' availability to each task is already low, little changes to their availability have a big impact on the duration of their tasks.

Uncertainty in execution time comes from uncertainty in the amount of work (±X%) and resource availability (±Y%).

We all experience days when we get nothing done because we spend the whole day in meetings, emails, phone calls, and interruptions. On those days our availability to design, experiment, research, and other sit-and-do-the-work type tasks goes to zero. Availability varies from zero to about seven hours per day, or about 3.5 +/-100%, unless we work more than 8 hours a day.

In contrast, the variance on our work predictions is often 50%, when we spend a little time to think through task or base the estimate on real data from past tasks.

Without a concerted effort to maintain our resource's availability, it is likely to fall below the the 6.4 hours a day we assume when we plan resources to 80% utilization. If actual availability is 4 hours/day and we estimate it at 6.4, our projects become 60% longer (6.4/4) and that assumes we correctly estimate the work, which is also rare.

The result is longer projects, therefore more pressure resulting in more push, and more multitasking. Management can become anxious to start the next project in hopes of completing it done earlier. These reactions just makes matters worse. And as overloads increase, people take more risks, which result in even more work later. It is a vicious cycle produced largely by overestimating people's availability and loading the system with too much work.

Some variability is bad

While variability is necessary for innovation, it is also true that some forms of variability do not create value - some forms are bad. For example, forgetting to connect the wires to record data during a 1 week material fatigue test and not discovering this until the end of the test. Some techniques for reducing bad forms of variability are outlined below. The first four techniques above reduce work or make it more predictable. The last technique increase availability. All of these result in shorter and more predictable projects.

- Reuse designs where innovation doesn't add value (don't reinvent the wheel)
- Standardize processes (e.g., a checklist to remind the test operator to connect the wires used to record test data; drawing release process, etc.)
- Automate tools and processes (e.g., leverage computer software to do repetitive tasks and take human error out of the equation)
- Base work estimates on data, not guesses. Use actual work measured on similar tasks from the past.
- Measure and control Resource Availability using Time Blocking, limiting Work-In-Process (WIP), minimizing multitasking, and removing or deferring less critical work.

Time Blocking is the practice of setting aside a block of time to perform certain tasks. For example, you decide to set aside three hours every morning for the team to get a particularly task done. During this time, we minimize or eliminate status meetings, interviews, performance reviews, interruptions, and other activities which take resources away from completing the tasks assigned to them. Although, these activities are still important, the benefit of time blocking is that average availability is increased and the variance is reduced. By insuring that at least a little work gets done each day, we improve the flow through our development resources.

What do typical visual project management tools do to control resource loading and availability?

Like showing the queues in our system, typical VWM tools and formats only do a little to help control loading and increase resource availability. Most accomplish this by limiting Work-In-Process: the number of active tasks any one resource can be working on at any one time.

Unfortunately for hardware teams, because of procurement times and other causes of forced wait time, the resources have upwards of three active tasks at any one time. This reduces multitasking somewhat, but not as much as it should, because daily multitasking remains invisible.

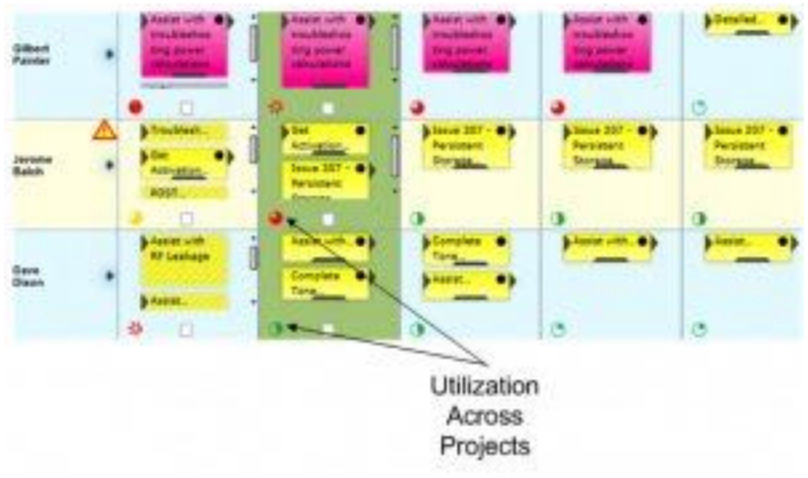
In addition, especially when using physical VWM project boards, it is impossible to apply work to the stickies and have it roll-up to an overall utilization level for the resources. Instead, this is approximated in other tools which are separate and out-of-sync with the day-to-day tasks.

Playbook ensures resource overloads are highly visible

In Playbook, daily multitasking is highly visible, as are short term and long term overloads.

When resources are loaded above 50%, the capacity dial turns yellow, and when they are loaded above 70%, it turns red. This loading updates immediately as tasks are resized, added, and removed from the resource, providing instant feedback.

Figure 7: Resource Loading



With this visibility, we can help resources know what priorities to focus on and what tasks to defer. We can take actions to increase the availability of the resources who are most critical, such as removing them from a meeting to focus on their critical tasks. Lastly, we insure there is room for the stuff that happens (variability), so that stuff doesn't kill our project schedules.

Push vs. pull work management

I'll use an analogy to describe the difference between pull and push at the system and resource level. This analogy is depicted in the pictures below. If you live in the United States, you can probably guess what day this is:

For those who don't reside in the U.S., this is a picture of your typical department store on Black Friday – the day after Thanksgiving Day, when most of the stores in the U.S. attract a flood of customers with low prices on many of the Christmas presents people need to buy.

Generally, the shoppers' experience goes like this: They rush to the store the evening after Thanksgiving dinner, and maybe stand in line with a few hundred other people waiting for the doors to open. When the doors finally open, everyone squeezes through clogged-up doors and aisles to get to the bargains they are after.

The few square feet of floor space within reaching distance of a good bargain becomes especially overloaded, and everyone stops moving while they wait or fight their way through the clog to get their hands on the product they want.

Figure 8: Overloaded Resources inside the System

And so it goes, through many clogged aisles, busy intersections, and other piles of great bargains, slowing down at each one as the shoppers squeeze through. Eventually they find their way to the cashier line. Then they really wait - for hours – literally. They eventually make it out the door again and on to the next store, or home because who wants to go through all of that again?

The ‘push’ at the system level is generated by the customers walking in the door. More customers are added to the store "system" with no regard to how many are already there. There is no controlled limit to the number of shoppers allowed inside the system simultaneously. The physical limit is the only limit, and it is reached only when system so full that pretty much everything in it stops moving.

Inside the system, many of the resources (cashiers, aisles, intersections, etc.) are overloaded. Each shopper who must go through an overloaded resource slows down at that resource, and because there are so many overloaded resources, each shopper's time inside the system is much longer.

The shoppers on Black Friday are ‘the work’ in the system: the projects, sub projects, and tasks in our product development system. The cashiers, aisles, and intersections are our engineers, designers, test technicians, and other scarce product development resources.

Wherever our resources are overloaded with too much work, the work is slowed down. The more projects, sub projects and tasks we simultaneously load onto the resources, the more clogged up they become, and the longer it takes to get the projects through.

Black Friday illustrates the cost of ‘push’ at the system level. But it could be even worse, if we didn’t use a pull system at the most critical resources in the system – the cashiers.

Push and pull at the resource level

To illustrate ‘pull’ and ‘push’ at the resource level, we look at a single resource. Let’s start with an ‘aisle’ resource where there is no controlled limit to the number of shoppers sharing that aisle. Once it reaches a threshold of only a couple of shoppers, every new shopper added without another one leaving clogs up the aisle a little more, which slows down some or all of the shoppers a little more. By the time there are ten shoppers sharing a ten meter aisle, they are close to gridlock.

A resource processing multiple tasks simultaneously and thereby delaying the completion of some or all of those tasks is called ‘multitasking’. Note, our definition of multitasking has at its core the necessary condition that some tasks are slowed down by the presence of other tasks. When a resource simultaneously processes multiple tasks, but none of those tasks are slowed or degraded in any way, that is not multitasking by our definition.

In contrast to the multitasking aisles and intersections, cashier resources use a pull system and Work In Process (WIP) constraints to single task (work on one task at a time). When the current shopper is done, they pull the next one in line and process the next shopper, and so on.

Figure 9: Pull System at the Critical Resources

This picture indicates a long wait because the critical resources are so overloaded, but it could be a whole lot worse. Can you imagine what it would be like if the cashiers operated in ‘push’ mode, and multitasked across all of the shoppers in line? They would scan one product from the first shopper, then one from the second shopper, then one from the third, and so on through everyone in line. Then they go back to the first shopper and scan his next product. Every time they switch shoppers, a little time would be spent in the process of switching. How long would our shoppers wait in the checkout line line then?

Imagine if every time we went to a store, for example, the local grocery store, the cashiers multitasked? If, on average, there are 3 people in line at a time, our average check out time would triple, from 5 minutes to 15 minutes: and that is before adding the switching costs. In the picture above, with twenty people in line, every hour a shopper spends in line in normal 'pull mode' would become a whole day in 'push mode'. Black Friday would need to be renamed Black Friday, Saturday, and Sunday.

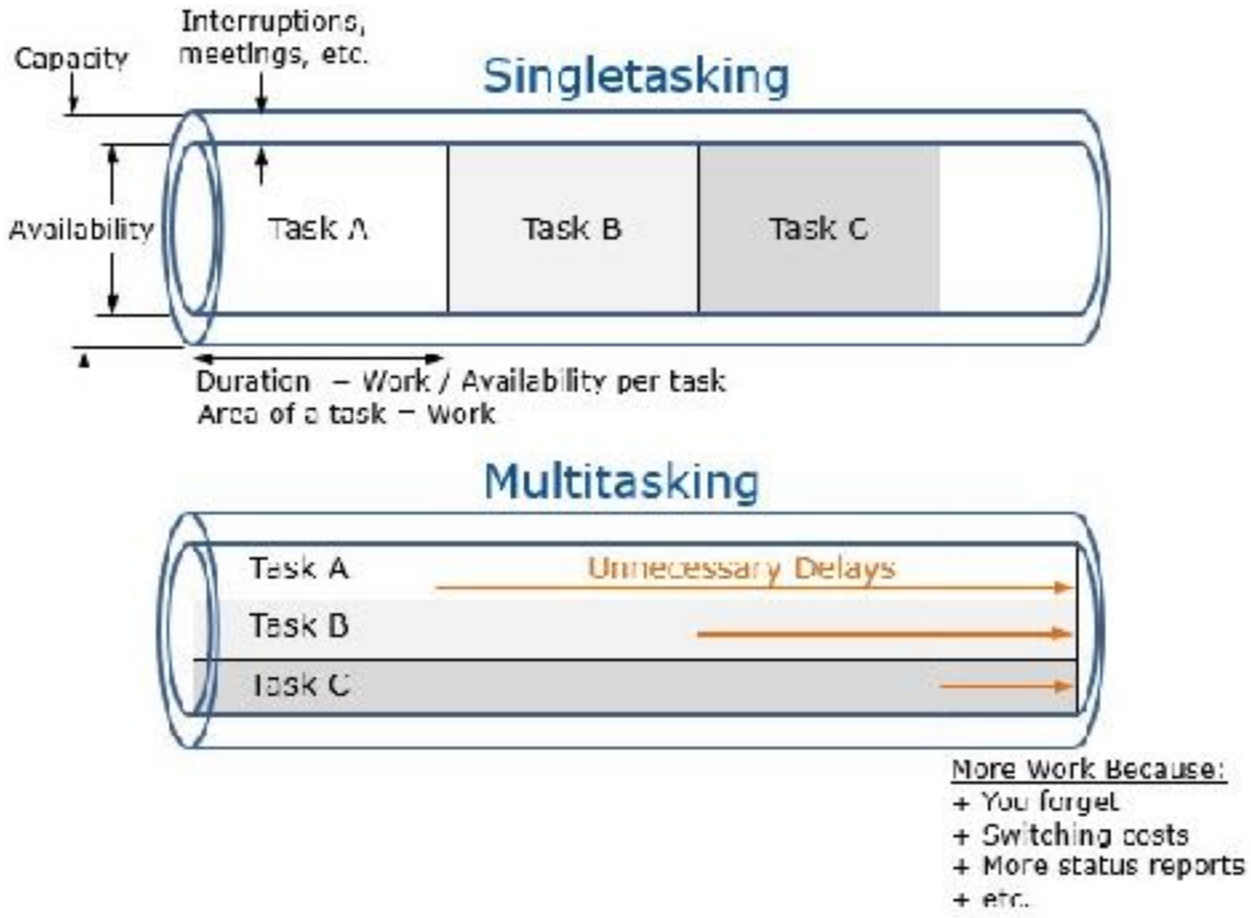
The first person in line really gets delayed, and the last one is also delayed because of all of the wasted time spent switching between everyone in front of him. Everyone would be frustrated – the shoppers, the cashiers, and the managers. At least until they get used to it and start to believe “that’s just the way its done.”

Fortunately, when we can see the work – like we can see the shoppers and their purchases – we more naturally converge on using pull system. But when we can’t see the work, the opposite often happens and resource-level ‘push’ systems very often result.

This is what is happening in our typical, push-based development systems. The work in the system is very difficult to see without some form of Visual Work Management tools, and even then, unless we have the really good tools, we only get to see *some* of the work.

The result is that the resources are overloaded and multitasking, and the durations of their tasks are unnecessarily extended. Instead of the ‘single piece flow’ as demonstrated by the cashier example, we have multitasking as depicted in the image below.

Figure 10: The Costs of Multitasking

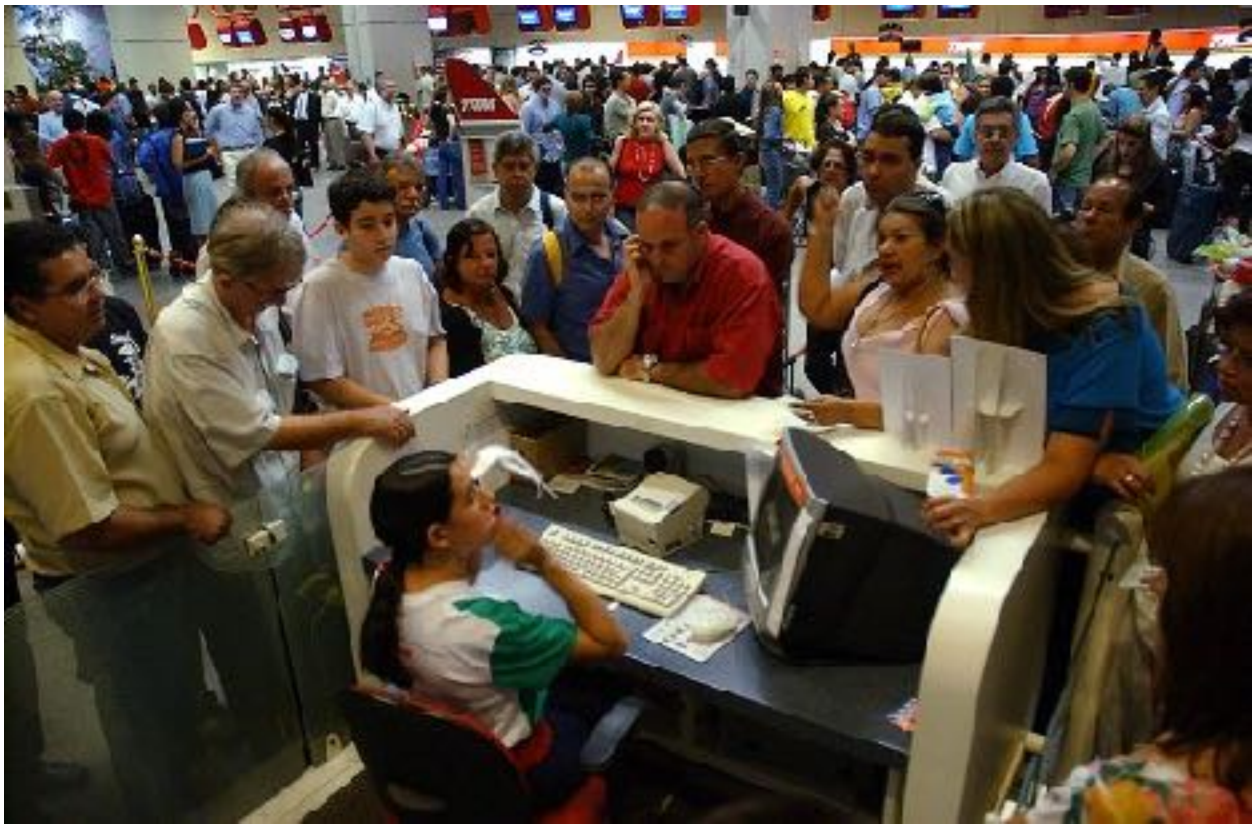


There is more to discuss about this image, and multitasking in general, which I will cover in the next post. In the meantime, let’s review the other key ingredients in a good resource-level pull system.

Key ingredients in a resource-level pull system

We've already touched on two key ingredients of an effective resource-level pull system – Visible work, and work in progress (WIP) constraints. Cashiers have a WIP constraint of 1. However, simply saying "only work on one at a time" is not sufficient. Consider this busy airport counter.

Figure 11: Working on one task at a time



How easy is it for this resource to work on only one at a time? How often are they coerced to multitask? To achieve pull, we must also have tools that help make it happen. The cash register at the grocery store can only process one order at a time, and the checkout lane is only 1 person wide. These tools help force WIP constraints and limit costly multitasking.

But even Visible Work, WIP Constraints, and tools that encourage pull are not sufficient. These tools must also work in tandem with a supportive culture. The way people react to the situation can either support the pull system or it can degrade it. For example, if our shoppers expect or demand to be served when they arrive, rather than patiently waiting their turn, the cashiers and counter workers will multitask at least a little. It is human nature to want to help someone who is clamoring for it. But it usually slows everyone down.

The same is true of our product development resources. Visible work is the first step, and at least some form of WIP constraint must be applied, which requires the right tools to really make it happen. Last, but definitely not least, the culture must support the desired behavior.

Clear and correct priorities across projects

Multitasking

A lot has been said over the last several years about the fallacy of increased productivity when a person multitasks. Lean, Agile, and the Theory of Constraints each recommend reducing or eliminating it, as do many scientists, and sociologists. Just Google search for 'multitasking' and you'll see what I mean.

Much has also been said, especially by people in hardware development companies, about the impossibility of eliminating multitasking. Many see the large wait times associated with common activities like drawing release and component procurement as a waste of people's capacity if multitasking is not allowed. Many see the need for development resources to spend some of their time sustaining production. And many see tight integration between what they think of as "tasks," or example 'design part X' and 'design part Y', when the parts impact each other.

In the next part of this discussion, we'll find common ground that allows us to see and eliminate the costliest multitasking without going overboard and wasting capacity, delaying production issues, or creating a lot of rework.

In this first picture, note the area of each task rectangle is the total work required to complete the task. Its height is the resource's availability to perform the task, and the width of each task is its duration. Work is generally the "given," availability is the input variable, and duration is the output, determined by work and availability.

These tasks can be thought of as a malleable, but non-compressible fluid, like Play-Doh, flowing through the resource's pipeline. The pipeline's total diameter is the resource's capacity, which, in a sustainable (Lean) system must be considered mostly fixed. Our control is in how we choose to divide up that capacity.

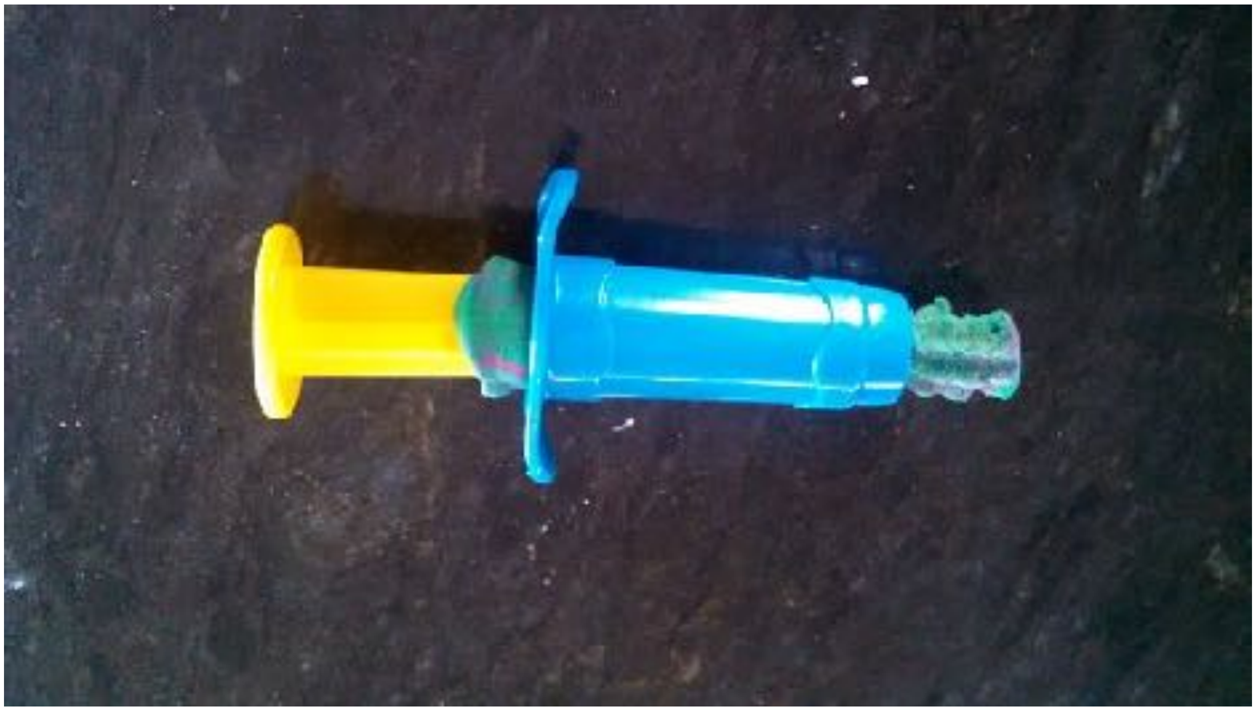
Notice that availability is less than capacity, because of meetings, bathroom breaks, interruptions, miscellaneous emails, and many, many other things. These additional tasks choke the flow of work through the pipeline. The remaining availability, is then divided across the resource's in-process tasks (WIP). The availability applied to each task determines the duration of that task.

Notice, also, that the area (total work) of each task is greater in the multitasking scenario. This is because of multiple factors, including switching costs and the time spent remembering what we did and why we did it. Also, the longer each task takes, the more time we spend reporting the status of that task. In short, multitasking creates more work.

Another cost of push

There is an important difference between the Play-Doh analogy and real tasks through a resource pipeline. With Play-Doh, we can increase the rate of flow by increasing the force we use to push it through, almost without limit. However, imagine hitting the plunger in the picture below with a hammer - what would happen? The same thing that happens in product development - either the pipe breaks, or stuff comes out the wrong end, or both.

Figure 12: Excessive pressure forces some work to escape (not usually a good thing)



Excessive pressure has this effect even more easily on real tasks. Pressure often causes some work to be omitted, rather than forcing it through faster. Some of this escaping work is otherwise known as 'cutting corners' and 'taking risks' and very often it comes back later, in the form of a lot more work and costly delays.

Definition of multitasking

Note, figure 12 depicts “sit and do the work” type tasks, such as; draft a document, review a document, run a test, set up the analysis model and so on. For these tasks every hour not applied to that task delays completion of that task by one hour. If your availability is one hour a day, each hour not applied to the task is one whole day of delay.

“Sit and do the work” tasks are in contrast to monitor tasks like receive parts and release documents, where the task duration is not so directly dependent on the resource’s availability. Monitor tasks generally take priority over work tasks, and therefore they leave less availability for the work tasks. Monitor tasks become much like meetings, emails and interruptions, in that they line the walls of the resource pipeline and leave a narrower passage for the work tasks to fit through, stretching them out and creating more work due to switching costs, etc.

In addition, these are three independent tasks. For example, work on task C does not impact tasks A or B. If the tasks are dependent enough that they must be done together to reduce rework, then we don't call that multitasking. Instead, we segment the work to better reflect what we must do together and what we can do separately. If we can't do much separately, we make it one task.

Lastly, there are sometimes blockages on these tasks. If for some reason we cannot move forward on one task and instead work on a different task, that is not 'multitasking' in our definition. This includes mental blockages, where you just need to get away from a task for a little bit to clear your head.

At PLAYBOOK, our definition of multitasking is when two or more independent work type tasks are in process at the same time and not blocked. Instead of picking one to focus on, the resource works on more than one. Even with this limited definition of multitasking, there is a lot of multitasking every day among typical resources.

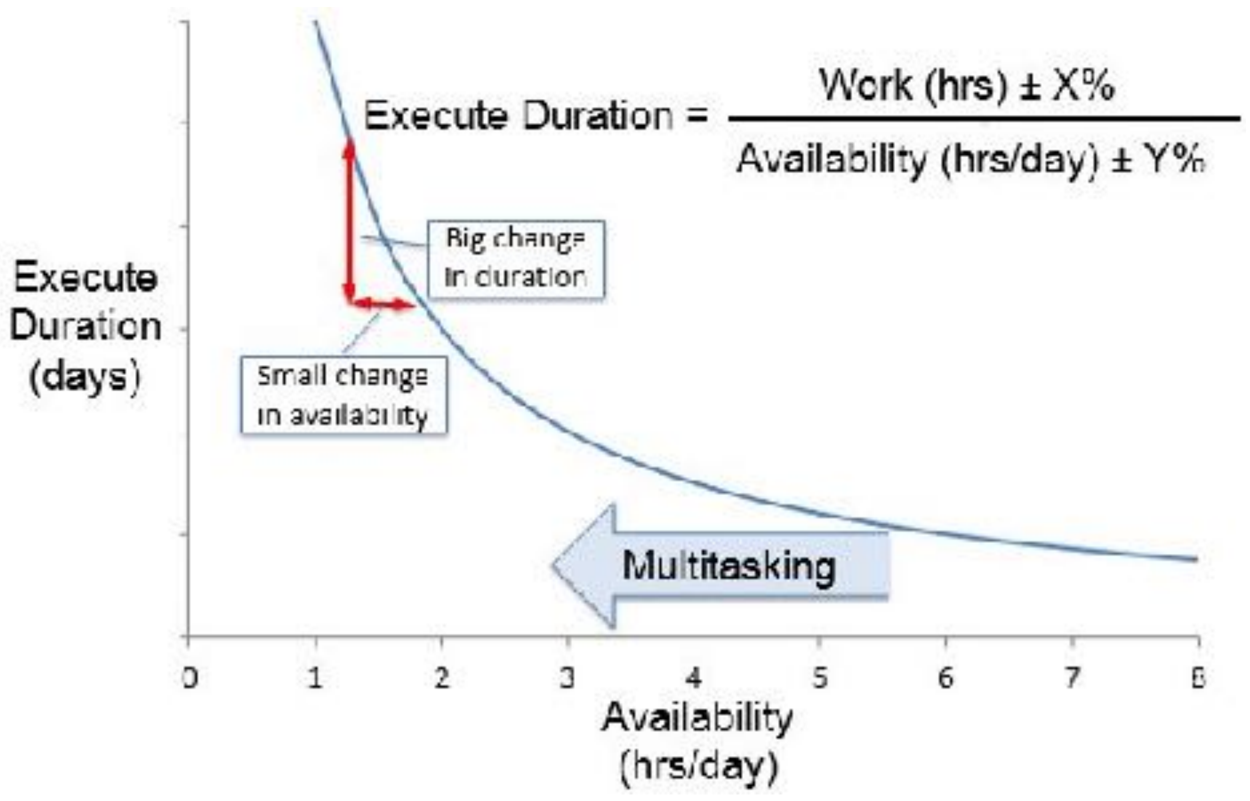
So, returning to Figure 1, let me ask these key questions:

1. How much earlier did we complete task C by starting it earlier?
   Answer: We don't complete C any earlier – we complete it later.

2. How much did we delay A and B by starting C earlier?
   Answer: A LOT.

We simply delay some tasks without accelerating other tasks in the process. In our definition of multitasking, it is all downside. And there is a lot of it in our systems right now.

Figure 13: Multitasking Creates Uncertainty and Turbulence



To make matters worse, when we spread our availability more thinly across the tasks, we lose a lot of confidence in the duration of those tasks. This uncertainty causes turbulence in our product development systems, which further slows the flow of projects though the system.

Multitasking is a natural outcome of a push system

So why do people multitask so much, and what can be done to reduce it? There are many reasons but in this post we will discuss a critical contributor to reducing multitasking – having clear priorities. In most hardware development environments, the team members are required to work on multiple projects, and multiple activities within a project simultaneously. Along with activities related to developing a new product, resources often have multiple sustaining engineering activities which require their attention. Very often different people have different opinions about what a given resource should focus on. In a typical matrix scenario, people have multiple managers come to them asking for the status reports on various tasks across projects. What's a resource to do? Tell them, "Sorry, I'm busy with someone else's tasks." This response is not in our nature. It is uncomfortable for most people to say, "No." Instead, people will just multitask so they can say, "I'm working on it."

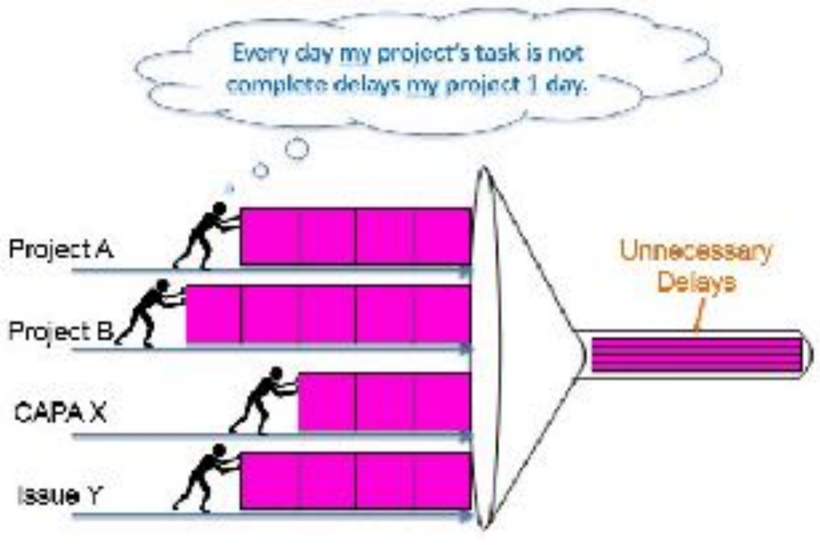

Every day my project's task is not complete delays my project 1 day.
Project A
Project B
CAPA X
Issue Y
Unnecessary Delays

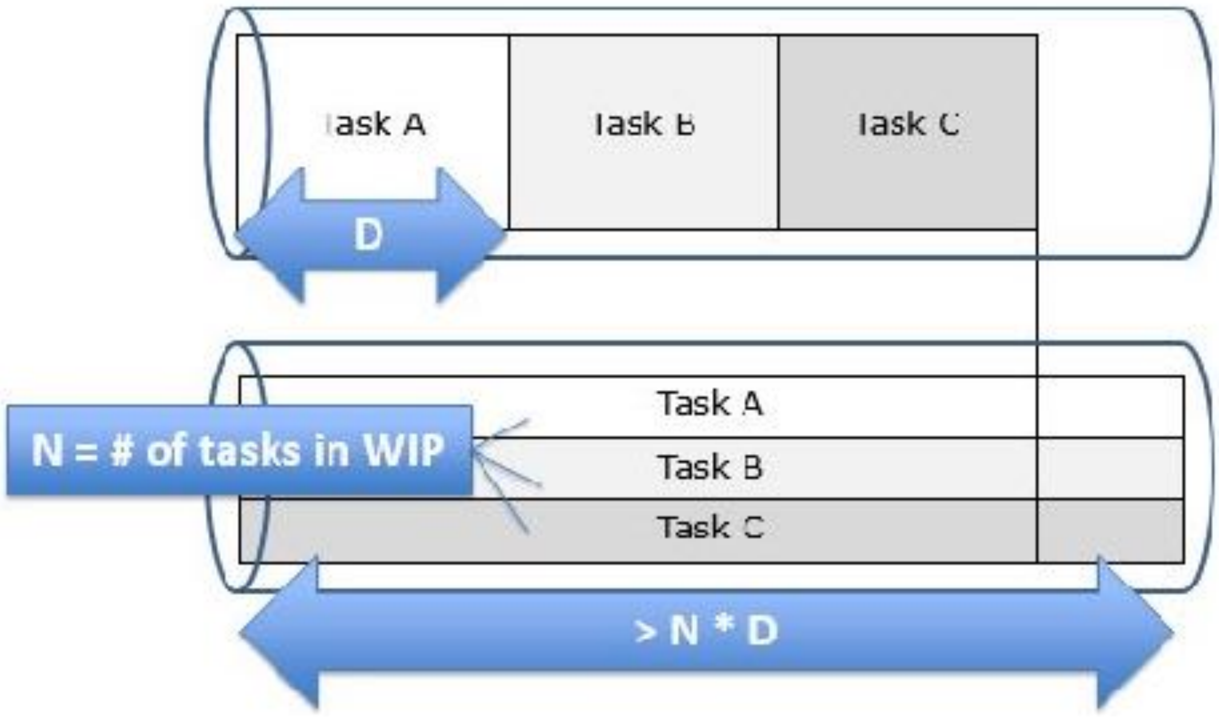
Figure 14: Push Creates Delays

Making matters worse, even on a single project there are often multiple activities which require a resource's time. Imagine if the four tasks on Project A in the image above were stacked vertically and all pushed in simultaneously, rather than lined up horizontally in succession. That is what it usually looks like to the resource anyway.

The value of clear priorities

The result is that all of the tasks get very stretched out, and a significant amount of extra work is created. If our resources, on average, work on 3 tasks at a time, each task takes more than 3 times as long to complete.

Figure 15: Value of clear priorities



If they average 5 tasks in WIP, each task takes over 5 times as long. How many tasks do each of your resources have in WIP right now? Is it any wonder that completed work just trickles out?

Note, the directions of the flow in the pictures above are reversed. In figures 2 and 4, we depict flow in the traditional direction: from left to right. What happens on day 1 is on the far right. In Figures 1 and 5, however, we show a calendar-day, Gantt Chart view, where flow goes from right to left and what happens on day 1 is on the far left. The figures below also use a right-to-left calendar-day view. Regardless of how it is depicted, however, the effects are the same, so please ignore the differing directions.

Achieving clear priorities

In order to reduce costly multitasking, establishing clear priorities both within a project and across multiple projects is absolutely necessary. To establish clear priorities it is very helpful if priorities are common and project team members and management are on the same page with what each resource should be doing each day.

In order for priorities to be clear and common across multiple projects, there must be clear prioritization of the projects themselves. Project A, Project B, CAPA X, and Issue Y in the pictures above must be prioritized relative to each other.

Common priorities across projects is imperative, but not sufficient. If the work is not visible, then priorities aren't very clear to everyone, and push and multitasking will return. Visible work is an absolute must to achieve clear priorities.

From the previous post on Pull vs. Push using the Black Friday analogy, what if, instead of standing in line with clear visibility to the people (work) in front of you,

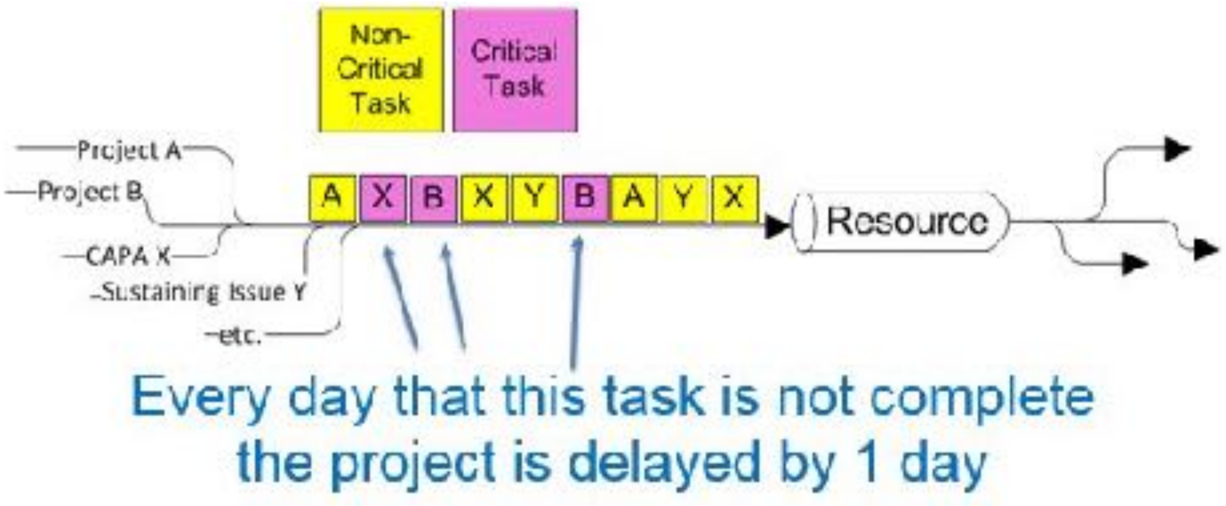
you were asked to sit in a closed room all by yourself until it was your turn? How long before you start clamoring for someone to multitask on you? Having recently been in the Emergency Room for seven stitches in my thumb, invisible work is a lot like that. I sat a long time, waiting for the doctors and nurses to help me out. Had I been able to see them busy with someone else's more urgent needs, I would not care. As it was, I was pretty annoyed. Sound familiar?

The value of correct priorities

With clear priorities, even if they aren't the 'correct' priorities, the throughput of the system will improve. Tasks A, B, and C are all completed earlier when multitasking is reduced. Maybe C would have been a better task to start first, but even if we didn't realize this, as long as we just picked something, we complete C earlier than we would by multitasking.

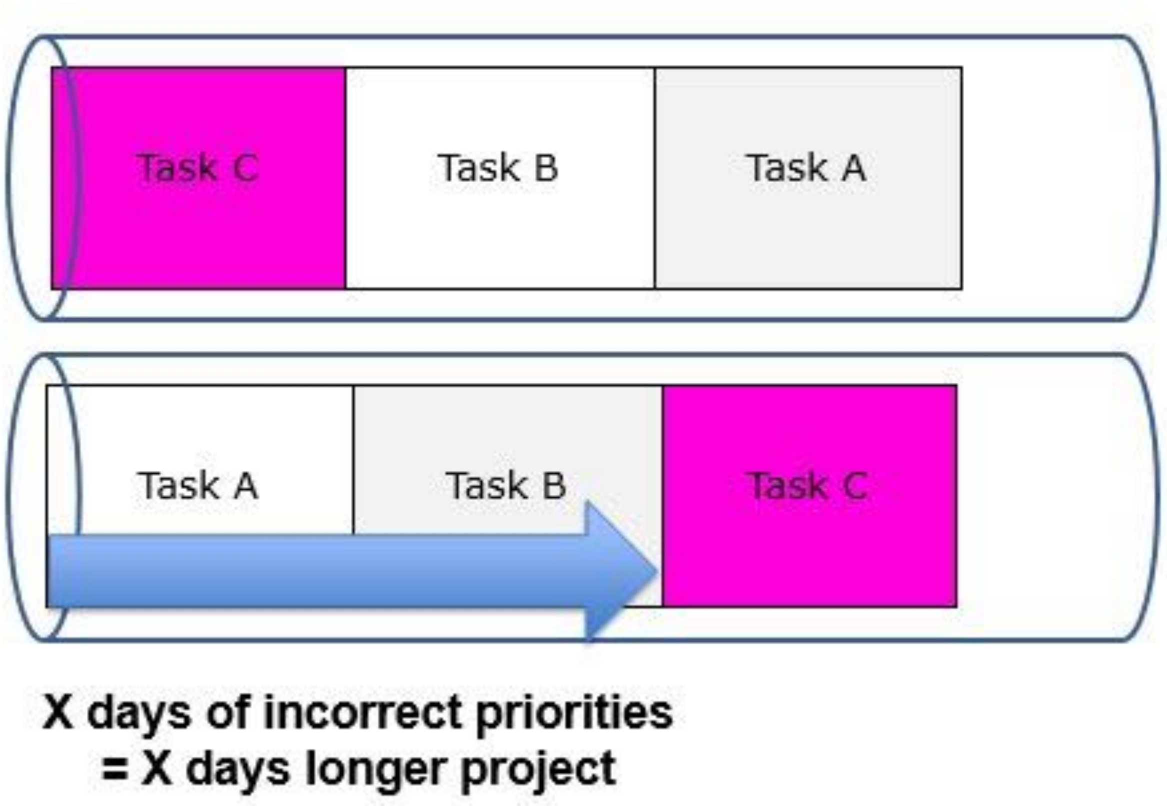
However, how much would it accelerate our project to recognize and work on 'correct' priorities as well? What if C were actually the task on the critical path that we should have worked on first, but we couldn't tell and we guessed wrong?

Figure 16: Value of Correct Priorities



As figure 6 indicates, every day we defer completing a critical task is one day later the project will be completed. This applies whether those days are lost while the task is inside the execution pipeline, or while it is sitting in a queue waiting to get in.

Figure 17: More Value of Correct Priorities



Every day we set the priorities of critical tasks correctly instead of incorrectly, every day we make the better decision about what to work on, is one day earlier our project will be completed. There is a huge opportunity to accelerate projects by getting the priorities right more often.

How to ensure people are working on the right thing more often

Correct priorities within a project are easily determined with a good model of the project (a good plan) and the criticality calculations that the model enables. Without a good model of our project, we must use gut feel to make priority decisions.

In the complex, dynamic system of product development, establishing correct priorities using gut feel is a lot harder than most people think. Importantly, each day we work on the wrong priority task moves our end date further out. Unfortunately, typical project plans fall far short of ‘good’ plans and don't help us make great priority decisions.

Good models of our projects are not difficult to develop, if you have the right tools. There are several very important ingredients which many tools do not contain, including good criticality calculations, the ability to see and limit multitasking, the ability to see, measure, and control resource availability, and many more.

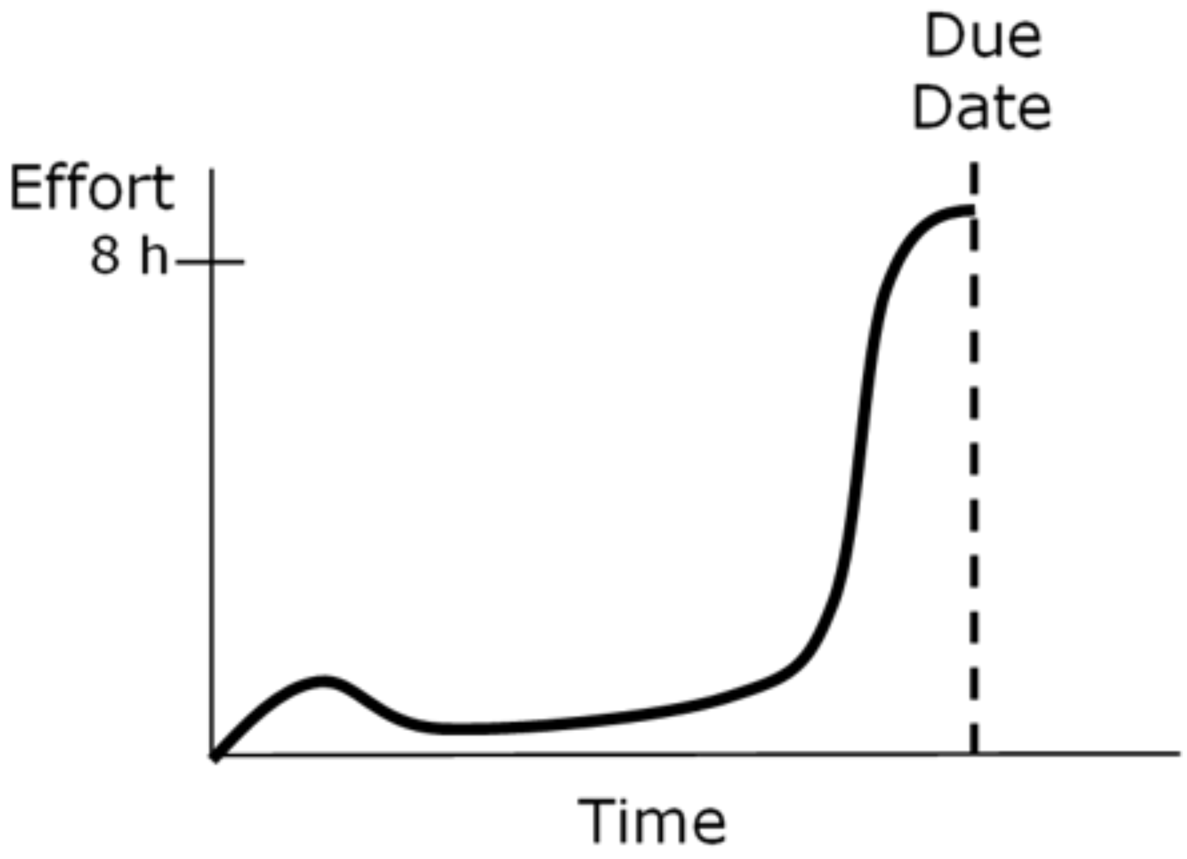
In a future post on Decentralized Project Management, we’ll dive deeper into the key aspects of a good model of a project and how to attain and maintain the model. In the meantime, there is one more key to reducing multitasking: the elimination of Task Buffers, and the use of Project Buffers instead.

Shared project buffers

Student syndrome

To explain individual task buffers, we'll start with a discussion about Student Syndrome.

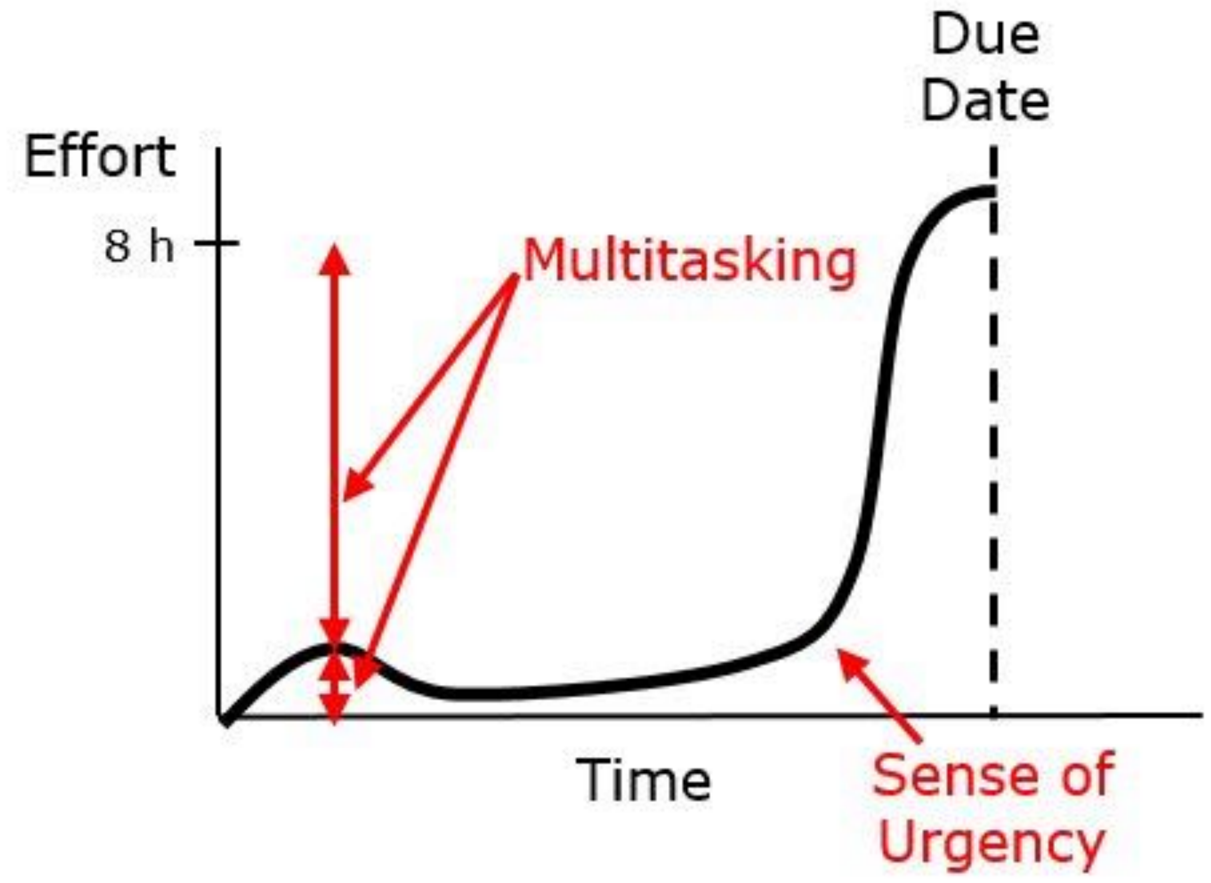
Figure 18: Student Syndrome



Student Syndrome refers to people's tendency to procrastinate - only starting to focus one's effort on an assignment at the last possible moment before its due date.

What happens is that we multitask, unknowingly causing more delay, until a task’s due date approaches, which invokes a sense of urgency followed by a more concentrated effort to complete the task.

Figure 19: Student Syndrome and Multitasking



Sometimes the task is completed by its due date and sometimes it takes longer.

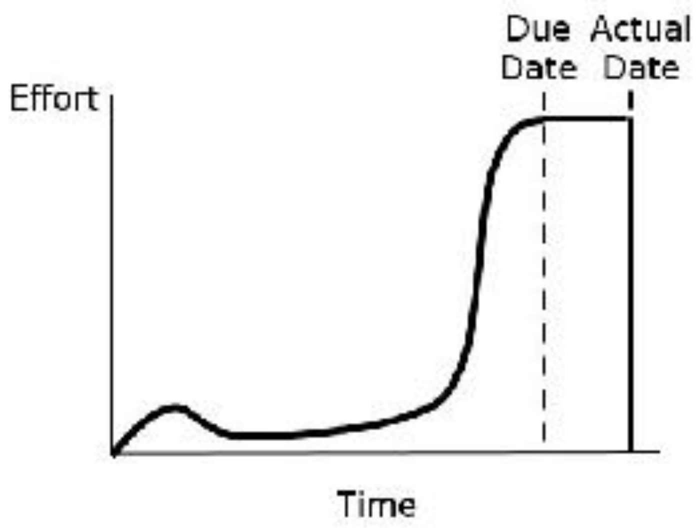


Figure 20: Student Syndrome, Multitasking and actual date

Student Syndrome consumes potential safety margins, and puts people under stress and pressure, and often requires heroics to get the task done. The impacts of Student Syndrome is that each and every task takes longer, causing the project to take longer, and the resource may cut corners because they're late.

Parkinson's law

But that's not all. There's another phenomena affecting how long it takes us to get the work done – Parkinson's Law. Parkinson's Law is the adage that the "*work expands to fill the time allotted*."

For example, the larger the suitcase we bring on a trip, the more stuff we pack, right?

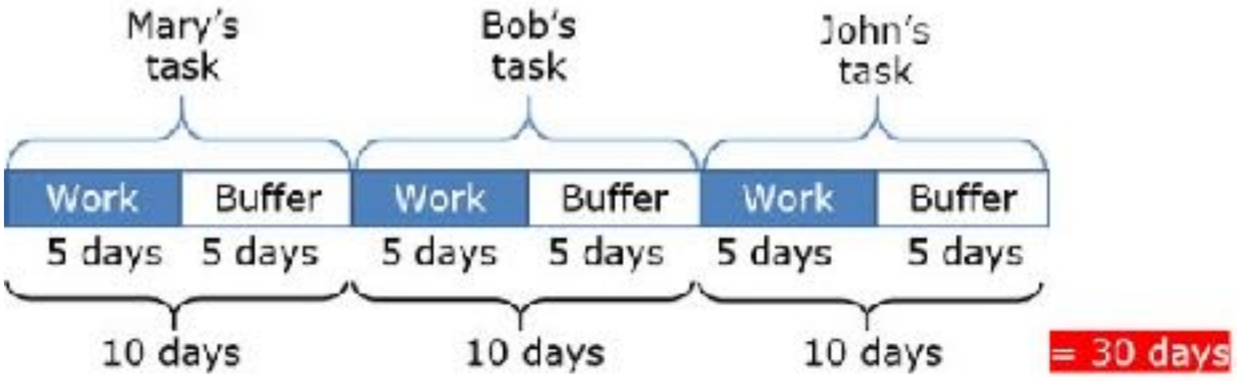
So, the more time we have to work on a task, the longer we work on it, forever trying to improve or perfect it.

“Perfect is the enemy of good enough.”
-Voltaire

The prevailing culture that feeds into Student Syndrome is one in which task duration estimates are treated as commitments and being late is unacceptable, therefore it’s better to under-commit and over-deliver.

For example, if Mary, Bob and John think their tasks will take them 5 days each to complete, they might commit to have each complete in 10 days, for a total of 30 days. This is a safe bet. Mary and Bob have high confidence they’ll complete the tasks no later than the commit date. Estimates like these are referred to as 90/10 estimates, 90% confidence the task will be done by the estimated completion date and a 10% chance it won’t be.

Figure 21: 9/10 Estimates



Since we’ve been measured on our ability to hit due dates and being late is unacceptable, we’ve been conditioned to pad our estimates (90/10 estimates). This padding is what we refer to as individual task buffers.

In fact, because of Student Syndrome and Parkinson’s Law, the individual task buffers typically get used and often the time to complete the task extends beyond the buffer.

The bottom line impact of Student Syndrome and Parkinson’s Law is every task takes longer causing projects to take longer.

"Insanity == Doing the same thing over and over again expecting different results."
- Albert Einstein

Nothing will change unless we do something differently. If we could adopt a new way of executing our work, we could get projects done sooner.

Using shared team buffers just makes sense

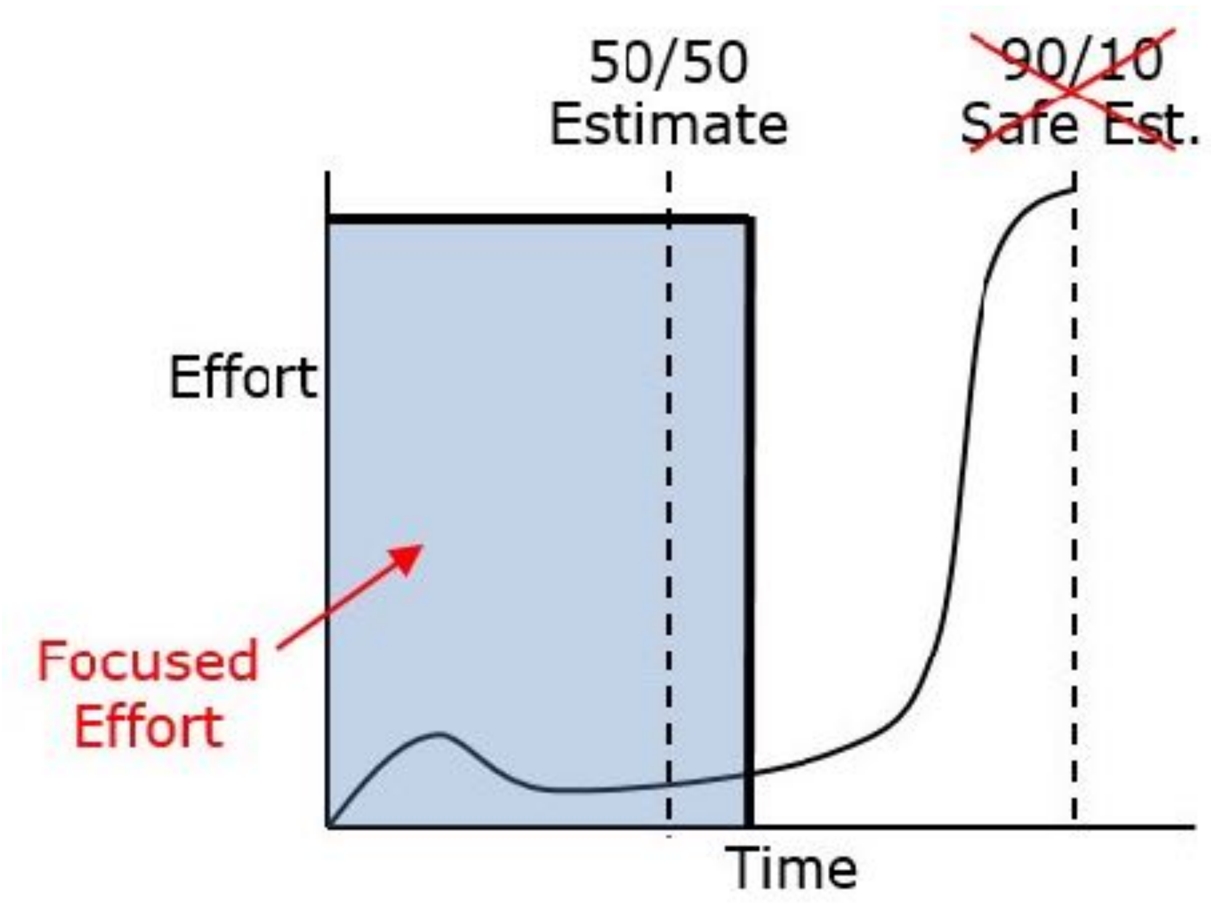


Figure 22: Focused effort

Instead of artificially buffering all our tasks, we estimate durations using a 50/50 guideline - 50% of the time the task will be completed on or before the estimated date and 50% of the time the task will take longer. These are sometimes referred to as focused durations.

We then focus (single-task) on the task until it is complete. We'll do the same amount of work, have some margin, work under less stress and we'll likely complete the project sooner.

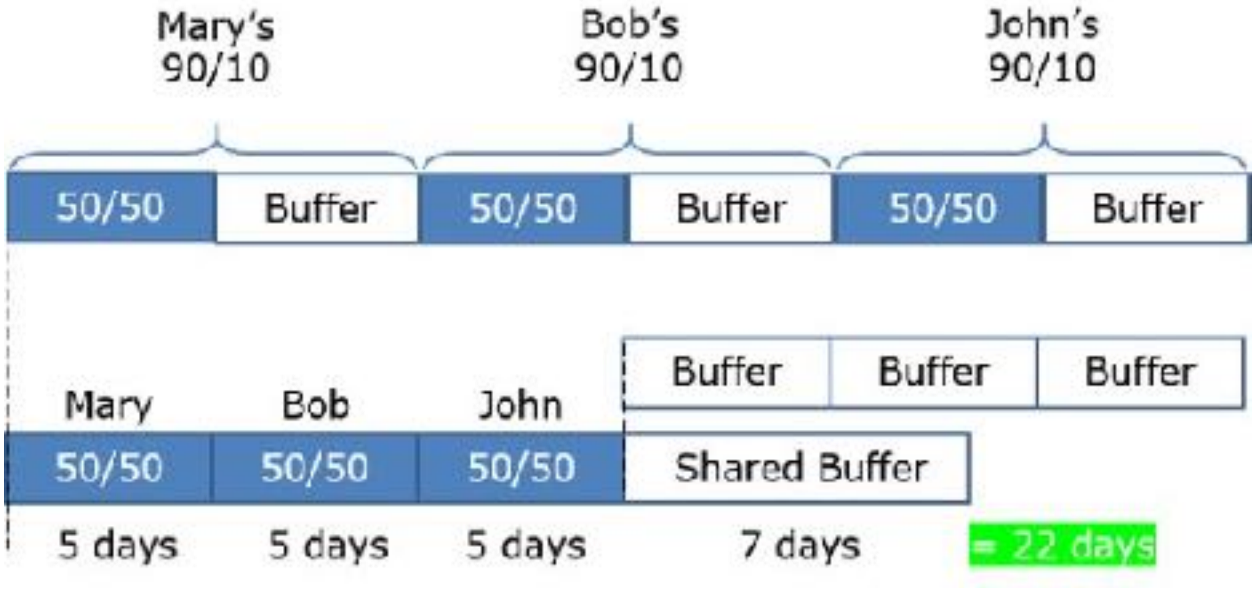
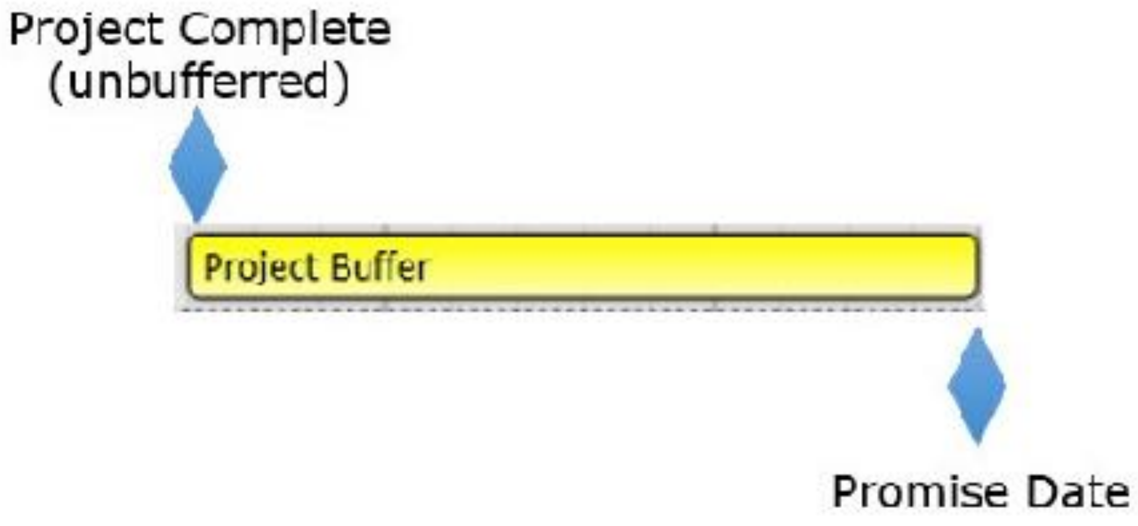


Figure 23: Bob's task duration 22 days

For example, if Mary, Bob and John think their tasks will take them 5 days each to complete, then their 50/50 estimates for each task is 5 days.

Figure 24: Promise date



To accommodate for the uncertainty in the work estimates and the resource's availability to do the work, the artificial, invisible task buffers are replaced with a shared team buffer at the end. The published completion date (promise date) is at the end of the shared team buffer.

The length or size of the shared team buffer is smaller than the sum of the task buffers, because we benefit from variability pooling, similar to statistical tolerance analysis. If you'd like to learn more about how to size buffers, click here.

Because we're single-tasking more and multitasking less, fewer tasks will go to full buffered length and some will get done early.

If we adopt this new way of executing our work, we will complete projects earlier and with greater confidence than the traditional individual task buffered approach.

Key Principle

Using visible, shared team buffers, instead of artificially inflating schedules by buffering every task, drives focus and encourages teamwork.

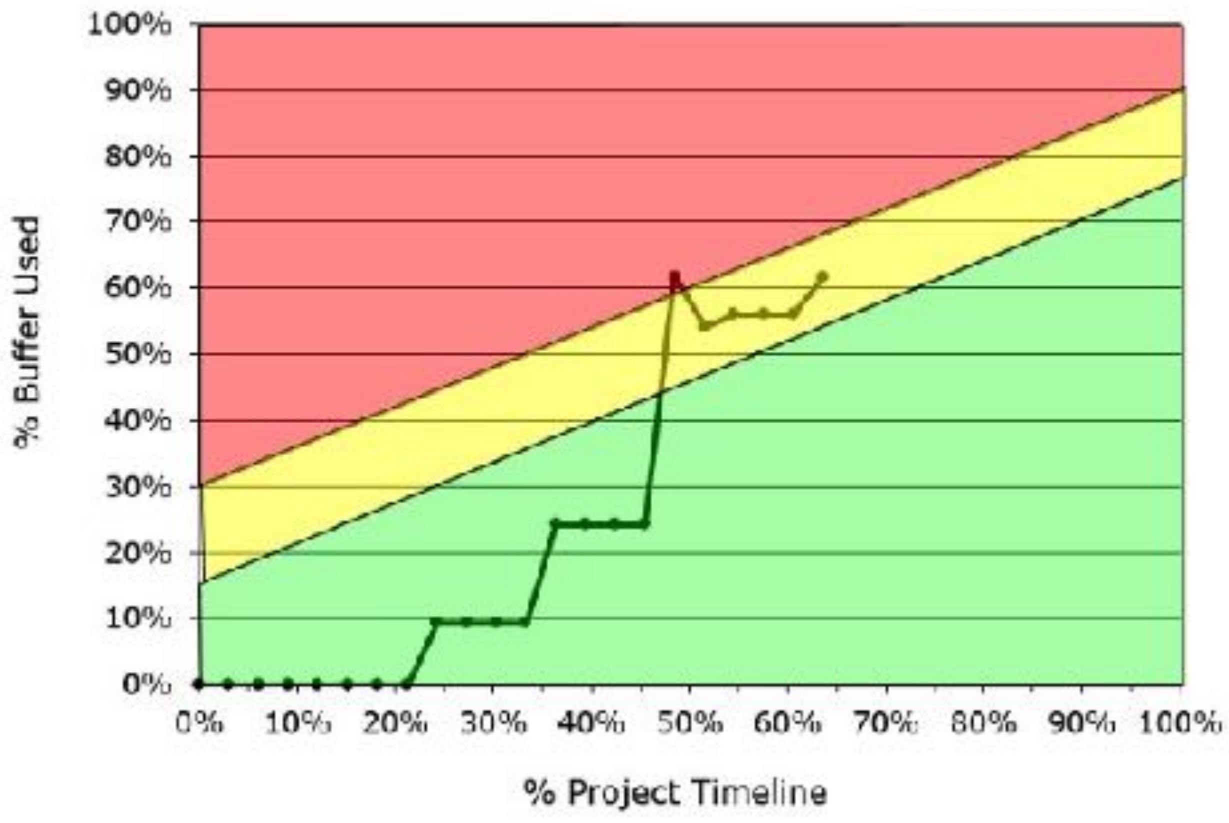
Monitor shared buffer consumption

The shared team buffer is like a cake (yum, did someone say cake?!).

Those involved in earlier phases of the project aren’t allowed to eat all of the cake, only their portion. They have to leave some cake for the downstream activities, such as verification testing, manufacturing, etc.

Buffer Charts (fever charts) are used to track the performance of the project by tracking the amount of buffer consumed, visibly.

Figure 25: Project Timeline



Typically buffer usage is charted weekly. If this week’s usage is in the green, the project is on track. If it’s in the yellow, the project is starting to use the buffer too fast and the team plans interventions. If it’s in the red, the team executes interventions. If you'd like to learn more about *Buffer Charts for Power Users*, click here.

In PLAYBOOK buffers are made visible using a task.

The amount of buffer used is seen by looking at how far the Project Complete milestone has moved across it. When the Project Complete milestone reaches the end of the buffer task, all of the buffer has been used.

You may be thinking, "What? If I remove my teams' commitments to Due Dates – won't the tasks take even longer?!"

There is a common concern among managers that removing commitments to due dates on tasks will cause engineers, for example, to keep working on their design much longer than they would with a due date. This concern is certainly valid, but there are good ways to prevent people from 'over-polishing the BB' without resorting to task commitments (and therefore individual task buffers). The answer is three-fold:

1. Visibility to the resource's work

2. A little peer pressure (via shared team buffers and standup meeting).

3. A good Definition of Done for those tasks where the end is not already clear

Decentralized planning

In project management timing is everything. Having the right information available to the entire project team at the right time is critical. In this way, project teams can make timely decisions that move projects forward, avoiding unnecessary delays.

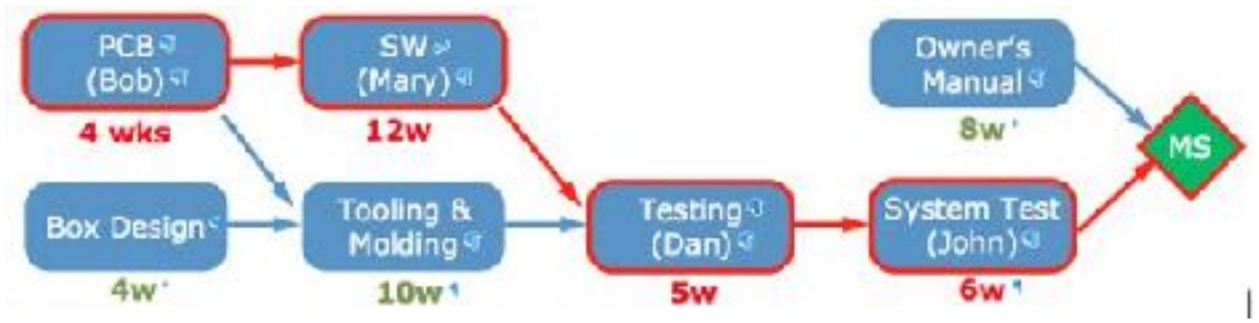
To sail from Los Angeles to Hong Kong in the least amount of time, we monitor our current position and course correct so as not to stray too far from the optimal path. The shorter we make the feedback loop, the less we deviate from the optimal path, the faster we arrive at our destination. Traditional project teams have weekly project status meetings (longer feedback loops) which result in greater deviation from the optimal path and longer project timelines. An entire week may have gone by before the team is aware of a delay that they could have mitigated many days earlier. How does a 12 month project become 2 months late? One day at a time.

Timely, fast information-sharing is what decentralized planning and management are all about!

What’s holding up information-sharing?

Traditional planning tools are file-based which restricts them to being edited by one person at a time, typically the project manager. In addition, project plans with more than a few hundred lines become too unwieldy for one person, the project manager, to maintain. To make the project plan more manageable the project manager combines activities into work packages several weeks long, often hiding potentially important information. In addition the plan is updated less often, weekly or bi-weekly.

Figure 26: Making queues visible



In the example above, how many weeks would pass before the project manager would learn the PCB is taking Bob longer than expected?

After the first week, the project manager would ask Bob, *"What percentage of the PCB is complete?" To which Bob would likely say,*

After the second week Bob would say, "About 50%."

And after the third week Bob would say,"About 75%."

However, after the fourth week Bob might say, "About 90%."

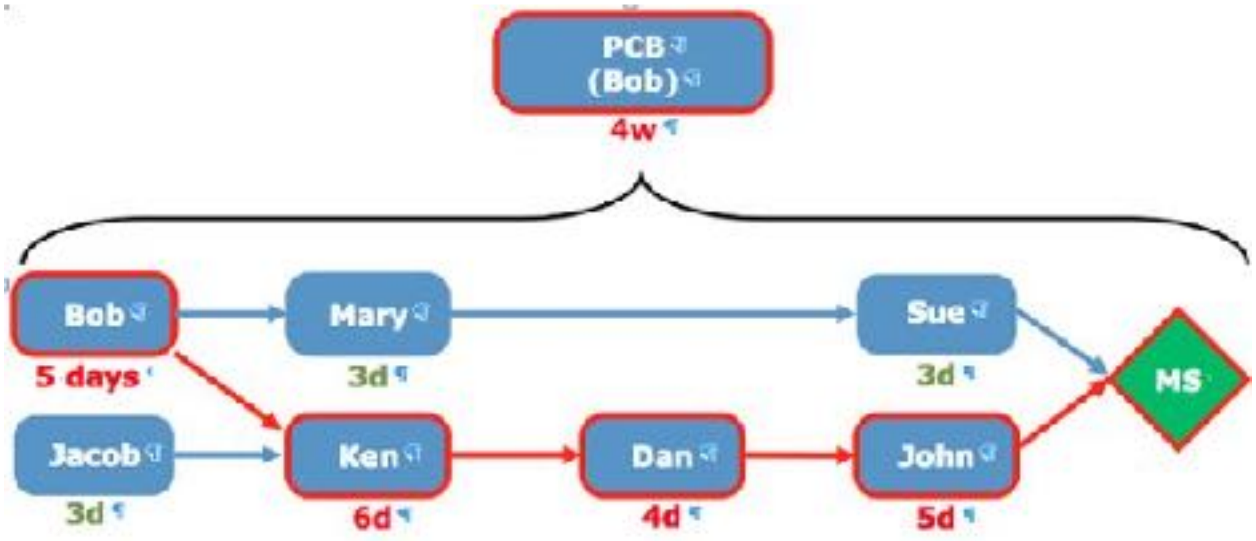
In a traditional environment such as this, feedback is delayed (weekly), leaving the project manager to be mostly reactive with fewer options available to him to prevent schedule delays. For example, if the project manager knew sooner the PCB was taking longer than Bob expected, he would have more options available to him to mitigate the potential delay and more time to plan and execute them.

Short and fast information feedback loops

What if project planning tools didn't have these limitations and could be edited by more than one person at a time? What if each project team had a project manager managing the big picture with an arsenal of mini-project managers, each managing their piece of the project?

Project plans could have a lot more detail and feedback would be much faster. The following flow chart shows an example of a process with a mix of resources and dependencies. Bob is the responsible engineer for the PCB. He develops the plan and updates his active tasks, as does Jacob and Mary and Ken in just a few minutes each and every day.

Figure 27: Short feedback loops



How much time would pass now before the project manager would learn the PCB is taking longer than expected?

Since Bob's first task is expected to be completed in the first 5 days, the project manager would know at the end of the first week that the PCB is taking longer than expected. If Bob's task was completed on time, the project manager would know in the next 6 days if Ken's task was taking longer than expected.

In a decentralized environment, feedback is much quicker, giving the project manager more opportunity to proactively prevent schedule delays. In addition, everyone's understanding of important details and the impact of change is clear, common and understood.

Won't the project team get bogged down by planning?

In a decentralized environment, the total amount of time each person spends updating the plan is about the same as the amount of time they would spend telling a project manager the status of their tasks. In addition, the project manager spends less time updating the plan and has the information on hand in real-time to proactively find ways to keep the project on track.

In summary decentralization and real-time dissemination of project information means:

- Daily operations are delegated to the team members with the project manager managing the big picture.
- Feedback is quicker and team members are more informed which allows the project team to make better decisions faster – they don't have to wait for information to go up and down the chain of command.
- Teams react quickly to situations where fast action can mean the difference between gaining and losing time.

Playbook enables decentralized project management

In Playbook the project manager creates a structure of the overall project plan with summary tasks and milestones. He then assigns Sub-Project Owners (SPO), typically core team members, to the summary tasks.

SPOs develop their detailed plans to near-term milestones. Since the people doing the work know best what needs to be done, this activity is typically performed with a cross-functional group. In our example, Bob might create an outline of the plan to develop the PCB, then review and revise it with Jacob, Mary, Sue, Dan and Ken.

The project manager oversees the overall plan (systems integration) and ensures the sub-plans come together accurately in a cohesive and consistent manner. Everyone updates the status of their tasks daily, in just a few minutes, which keeps the plan up-to-date and accurate. This provides the project manager with fast feedback so that they can make priority changes daily and gives them time to be proactive and find ways to save time on the schedule. Everyone is well informed, aligned and empowered, and projects get done sooner.