In this lesson, we'll build on top of what we learned in the previous lesson about creating simple systems for teams by transforming those systems into more complex ones that can achieve even better results.

First, why do we need complex systems? Why do we want to add complexity? We never want to complicate things just for the sake of complexity. But there are some relatively minor adjustments you can make to team systems that can dramatically improve them.

One of the benefits of a more complex system is that you can have much tighter control over the quality of the results of a team system. As we've learned, one of the biggest benefits to any system is getting more regularity and reliability in the end result.

Just by building any simple system or routine at all, you'll see those benefits. But when the quality of the end result is absolutely critical to you and your team, adding conditional logic at different stages of the system can help you find inconsistencies as you go and fix them before problems get compounded in future steps and stages.

Imagine you were manufacturing a product. You would want to make sure that every single unit you produced was exactly the same and that they were all the best possible quality they could be.

You would never want one customer to get something different from another.

Or if you were in a real estate firm and you wanted to ensure that all your customers got the exact same experience and that it was the best experience you're capable of providing.

The same could be true of a creative process where you want to make sure that, even though the end product might be different each time, everything you create meets some tight objective standards.

By adding the ability to measure the output of each step of a system by some standard, you can produce high quality results in a more reliable way.

The other major benefit of a complex team system is that it can operate much faster. When you have a complex system, you can break it up into sub-systems with their own steps.

Some steps have to happen in order, but others might be semi-independent. And when you have a team, you can have people operating different parts of the same system at the same time. That can dramatically speed up the time it takes to produce results.

But for all the benefits of adding complexity to a team system, there are also a few drawbacks you'll have to watch out for.

First, tracking progress can be difficult. When you have a system with many steps and sub-systems within it that have their own steps, there's just more to keep track of. And when multiple parts are moving simultaneously with multiple people working on them, knowing exactly where things stand can get even more complicated.

And communication can get a lot more complex. As you build more controls into a team system to try to improve quality, all the people who need to talk to each other in order to operate those controls and make sure the ball gets passed to the right people at the right time can get a little dizzying.

Basically, complex systems provide some very useful benefits, but they're also more prone to breakdown simply because they have more moving parts.

That's the trade off. But you can minimize those downsides with a little extra work up front when you're designing your systems and, of course, a commitment to steady improvement over time.

So, what's involved in that "little extra work" that's required to build faster and more reliable team systems?

Well, first, you'll need all the same pieces in place for a complex team system that you need for a simple one. So, buy-in from everyone involved before you build it is critical. Each step needs an owner so that someone's watching over every piece of the system. Deadlines are important so that slowdowns are caught in a timely manner. And really clear criteria to establish when each step is complete is needed to make sure there's no confusion about when handoffs are supposed to happen.

Once you have those in place, you'll need to focus on a few other important factors to get the most out of your system and prevent breakdowns.

First, the system controller—that's probably you, right?—will need to take a more active role, at least in the beginning.

When you're rolling out a complex system with conditional logic and multiple team members, it becomes more critical to have one person dedicated to watching over it, looking for problems, and making sure that everything is functioning the way it's supposed to.

This is because the more decision points you add to a system, the more opportunities there are for things to get stuck or handoffs between people to get missed, *and* the more people you have involved the less any one person will feel responsible to watch over everything. That's just a human psychology problem we all have to deal with. A shared responsibility makes everyone involved feel little less personally responsible.

So, whether it's you or someone else committed to the system that you trust, a complex system really needs a strong leader at the top of it to act as its shepard.

Sometimes, you might with a group of people to complete a single step. Like a marketing team instead of marketing person.

In these kinds of cases, you'll want one person on that team to be the controller for that step. Remember, in team systems, it's incredibly important for each step to have one and only one person who is ultimately responsible for it.

And to whatever extent you're able to, you're also going to want to encourage that team to create their own sub-steps. You're going to want them to kind of build their own system within the system. That way, each piece of the bigger step that the team controls can, once again, be assigned in a one-person-per-step sort of fashion.

That way, you continue to maintain that really strong sense of responsibility in each player.

Remember the ad agency we created a simple team system for in the last lesson? Let's see how we might improve their system by implementing this kind of structure.

Maybe we've done a great job recruiting clients and, in order to keep serving them all really fast, we need to speed up our ad creation process. And we decide there are two good opportunities for us to do that.

First, we realize we don't really need to wait for the copywriter to finish their work to get started on graphic design, so we start to run those two steps in parallel. They both happen at the same time.

We can't send the job off to the printer until both parts are done, but running both together speeds up the process by a full day.

Next, we notice that the copywriting goes fast, but the graphic design goes really slow. Our designer is overloaded. So, we sit down to figure out a solution and decide that we can break design up into two steps: illustration and layout. And to speed things up, we hire an illustrator to do nothing but pump out illustrations all day to allow our graphic designer to focus on layout.

Well, now we have two steps running simultaneously, and we split one step into sub-steps run by a 2-person team.

Since you're the account manager who's watching over everything, you'll want to pay careful attention to how the copywriting and design steps get completed on each job since they're both running together, but design still relies on copy in order to finish the job.

And you'll also want to make sure that the graphic designer doesn't totally offload their responsibility for illustration since that sub-step is being taken care of by someone else. The illustrator should be in charge of illustration, but the graphic designer should also take some ownership over it and make sure they're keeping an eye on all things design related.

That way, everyone still owns something, and any multi-step tasks also have someone driving them.

But, uh oh! Now we have another problem. We're creating tons of ad campaigns and business is going well, but our clients are kind of unhappy with the final results. They tell us we're not always getting it right after that 20-minute meeting we all do at the very beginning of the process.

Well, this is a problem that a complex system can solve, too. Remember: One of the biggest benefits of complex systems is that we can build some conditional logic into them to decide what to do next when the results aren't 100% under our control.

We decide that we want our clients to have a chance to make some changes before their jobs go to the printer. So, we start sending asking for approval after the design step is finished but before printing.

We don't want things to get too crazy, though, so we need to be specific about what things the client can and cannot ask for when they request changes. We decide that they can request changes to the copy and the general color scheme.

If the client is happy with our first design, we just send it straight to the printer. But if they aren't happy, we have them send us a form with the changes they want.

This creates what we call a system loop. A loop happens when we create a checkpoint in a system. In this case, we're asking the client to give us feedback. Is the design acceptable or is it not acceptable?

If it's not acceptable, we *loop* back in the system and perform the work again in order to make it acceptable.

On some jobs, the client just wants to changes a few words. In that case, we loop back to the copywriter to make those adjustments. And then the copywriter sends it to the designer again who makes sure the new words fit in the design before sending it back out to the client for approval again.

On other jobs, the client just wants to update the colors. In that case, we don't have to bother the copywriter. We can just run it by the designer and quickly make whatever changes were asked for. Easy.

Now we have a production system that's designed to produce really great results for our clients. And it's working! Everyone's happy.

But do you see how the amount of decision making increased? We introduced some new variables—some of them we can't control, like whether the client loves our design or not. So, you can see how it will serve you well to be really specific about what decisions are going to be made, when they're going to be made, who's going to make them, and what criteria you're going to use to make them.

If those pieces of the system aren't really clear, it can cause everyone involved to get confused, which can slow the whole system down and make things worse instead of better.

Okay, so now it feels like we're really winning. Every design we finish is getting 5 star reviews from our clients. The changes we made have really improved the quality of work we produce.

But, now we're starting to notice another problem.

We're busier than ever and creating the best designs we ever have, but we're actually making less money and the campaigns are taking forever to finish. What the heck is happening? We thought we'd perfected the system.

We sit down as a team to figure out what's going on, and the copywriter mentions that on some days, it feels like there's nothing to do. But on other days, they're spend the entire day doing nothing but making edits to the same campaign. Over and over.

After we probe a little further, we finally figure out what the problem is. Some of our clients are taking days or even weeks to approve our designs. And others are super picky and just won't approve anything. Every time we send the design over, they send it back asking for more changes.

What happened was that even though we set up deadlines for *ourselves*, we're now inviting our clients to be a part of the decision-making process inside our system, but we haven't given them any deadlines.

That doesn't work! They don't feel any sense of urgency to approve our designs, so their projects end up getting stuck in our queue waiting for them. Sometimes for weeks.

Complex team systems need more than just step-deadlines. They need decision deadlines, too. When a step is complete, we now have someone making a judgement about whether or not that step was done well enough.

We need that decision to happen quickly to keep the process from stalling. So, we decide that all our clients will get 48 hours to either approve our designs or request changes. And if we don't hear from them in 48 hours, we'll assume the design is approved and move forward.

This new decision-making deadline solves the problem we're having where campaigns keep getting stuck because of unresponsive clients.

But what about the other problem we noticed? The one where some of our clients are so picky they won't approve anything and keep requesting changes.

That's *really* slowing us down and making everyone frustrated. And it's because we failed to create what's called a "loop resolution."

Have you ever tried to go to a web page and you get an error message that says "we can't load this page because the url contains an infinite redirect loop?" It's kind of like that.

Or maybe you remember when you were a kid and you'd ask your mom if you could go to your friend's house. And your mom would say, "I don't know. Ask your dad." And when you asked your dad, he'd say, "Hmm, I don't know. Ask your mom."

And you're like, "I already asked everyone and I still don't have an answer!"

That's the problem we're having now, and we need to fix it.

In our newly improved system, we built a really useful loop that gave the client the ability to request changes which led to better results and happier clients. But it has a vulnerability where, in some cases, that loop can just go forever and ever and never finish.

To fix this, we do a little research and realize that most of our clients are perfectly happy after two rounds of revision. So we decide that's our new limit. You get to give input on the design twice, and then we move on.

This helps our indecisive clients become more decisive and it prevents the really picky people from monopolizing our time while still giving everyone a chance to give feedback.

And those final two changes are what really get things moving the right direction. Everyone's happy, and we're pumping out ad campaigns faster than ever.

So, here's what we learned in this lesson.

First, we learned about the benefits of complex team systems. You can control the quality of the output much better than you can with a simple system. And you can also organize the pieces of your systems to operate a lot faster.

But we also learned that the power of those benefits come with some drawbacks you'll have to work around. Tracking can get cumbersome in a complex team system, and communication can become really difficult if you don't plan out how to keep everyone in the loop.

Then, we built onto the example system from our last lesson to learn the most important aspects to plan for when you're creating a complex team system.

The first thing to do that helps everything else is to make sure that one person is ultimately responsible for looking after the whole system. You need someone who can zoom out and look at the big picture. And you also want to be sure that, when a team of people are collaborating on a bigger task, one person is in charge of overseeing that shared work.

And since one of the biggest benefits to a complex system is its ability to handle conditional logic and decision making, you'll want to be sure that the criteria to make those decisions is clear and there's a deadline for when to make them so that progress doesn't get stuck.

And, finally, we learned how powerful loops can be for improving the quality of what your system produces, but also that you need to plan for how to resolve the loops in a team system so that they don't accidentally loop forever and bring everything to a grinding halt.

Now you know the most critical steps to take to build these kinds of complex team systems in your own work and life.

Go ahead and spend some time today planning out how you might upgrade some of the systems you have with your own coworkers or even your family to get better results even faster.

Good luck, and I'll see you in the next lesson.