In this lesson, you're going to learn how to become a system repair technician. You'll hunt down and destroy the biggest problems and failure points in your systems so that they become faster, more reliable and, most importantly, they don't get abandoned.

There are all kinds of ways to optimize a system and squeeze every last bit of speed and efficiency out of it. But an incredibly fast and efficient system isn't very useful if it's collecting dust on the shelf, is it?

If you haven't experienced this yet, you probably will. The biggest risk to your success with system and routine building is that you'll do all this work to create something great… and then never use it.

Or you'll use it for a little while, but then get frustrated and give up on it.

First, I want to let you know that this is okay. It happens to everyone, and it's part of the process. Sometimes you work really hard to build a system you think will be useful, and it ends up just not working.

That is especially true when you're new to all this and still learning the ropes.

What you need to do to get out of that funk is spend some time figuring out *why* a system isn't working. Why do systems fail?

There are several symptoms you can look for. And lots of ways to fix those symptoms. We'll get into those in a moment.

But first, let's get into what happens when a system crashes and burns spectacularly right off the bat. It has to do with motivation. More specifically, a lack of it.

One of the biggest problems we face when we build systems is directing our energy toward organizing parts of our life and work that we're just not very motivated in. We make the mistake of thinking: Hey, here's this part of my life that's not very motivating. Maybe I can automate it with a system so I don't have to think about it anymore.

That approach almost never works.

So, one thing you may need to consider if you find that a lot of your systems are failing and you're not using them is that you might be looking to use systems to remove the boring or uninteresting parts of your life and not focusing on systemizing the things you really enjoy and care about.

Systemizing the boring *can* work. I don't want to say that it *can't*. But the key ingredient to being successful with that approach still comes down to motivation. You have to have something that you *are* motivated by. And you have to be able to see how systemizing the boring stuff will allow you to pursue the motivating stuff. In the end, you're still moving *toward* your motivation.

So consider that. If you don't have the motivation in the first place, a system will not save you.

Let's assume you're moving the right direction, but you're still running into a lot of problems. Your systems just aren't working that well, even though you spent a lot of time and effort to create them.

Well, if you remember from earlier in the course, we talked about a concept called kaizen—the Japanese philosophy of continual improvement.

A system that works and serves you for the long term *requires* that you embrace that kaizen philosophy because no system will work perfectly on the first try. A system that works perfectly is almost always one that didn't work that well in the beginning, but got the care and attention it needed to become great.

If you have the mindset that you can just set up a new system in one try and everything will be great, you're going to be disappointed. But if you embrace the challenge of getting through the difficulties in the beginning, you can create really sturdy systems that serve you for a lifetime.

So let's talk about how to actually put that kaizen philosophy into practice. You're ready. You're committed to adjusting and nurturing your systems almost like they're your children until they perform the way you want them to.

It's going to help a lot to know *where* to look for problems. We need to know which pieces of our systems are most likely to break down so that we can identify issues and make adjustments quickly.

And a quick side note: The better you get at diagnosing and fixing problems in your systems, the more motivating system-building will be for you because progress and seeing your work pay off is really motivating.

Alright, so when I look at my own systems and help others troubleshoot theirs, there are seven places I look for problems. And I look for them in this order because they range from big, easy fixes down to very small and intricate ones.

The first question I ask when I see a system not working or being ignored is, “Can the end goal be controlled?”

We’ve talked about this before because it’s so important. If you—or whoever is operating the system—isn’t completely in control of the outcome you’re going to run into problems.

This ties directly back to the motivation issue, too. If you use all the steps in the system exactly as they’re intended to be used and you don’t get the outcome you’re hoping for, that can be really demotivating.

And if the problem is that the ultimate goal you’re looking for is not actually one you control, you are *never* going to be able to reliably fix the system.

Here’s an example from my own life. A few years ago, I decided I wanted to drive more traffic to my website. So I set up a goal. I wanted 100,000 people to visit my site every month.

And then I created a bunch of actions and workflows I thought would help me achieve that and put them together into a system I would operate. For months, I followed the plan perfectly. But, each month, I didn’t meet the goal. And, each month, I got a little less motivated to keep trying.

The problem was that I was operating the system perfectly, but I didn’t actually control the results. No matter how hard I tried, I could never *force* people to visit my site.

When I realized that, I re-tooled the outcomes of the system to focus more on the the things I *did* have direct control of—like how many articles I published or how much promotion I did. Of course, I still wanted all those people to come to the site—and I adjusted the system based on the results I saw—but now I had goals I could feel good about and keep myself motivated.

So, if you don’t control the end goal of your system, that system is going to quickly become demotivating.

If the goal *is* controllable, then the next thing I look for is to see if the goal is too vague. That’s also a common problem I still have to remind myself to avoid.

The end result of your system should be specific and repeatable. The reason we build systems are often for vague reasons like, “I want to feel more calm” or “I want to save time” or “I want to lose weight.”

Those are all fine motivations and you're in control of them, but they make lousy system goals because they aren't specific. There's no defined metric you can look at and say, "Yep, I definitely got what I wanted out of this."

So, instead of making a goal to "be more calm" it could be a goal to "meditate for 10 minutes every day" or any other number of actions you could string together that you believe would help you attain that goal. See the difference?

Instead of "saving time" you might say "I want to cut an hour from this process" and then adjust how you do your work in ways that you think will help you actually save an hour of time.

The next problem I look for is whether the system has even been used at all. If you built a system, but are struggling because you haven't implemented it, the most likely cause is that you're trying to make it too detailed and intricate. You over-engineered the system and burned yourself out before even getting started.

It's easy to make this mistake. One of the main reasons we create systems is because something in our life feels big, complex, and difficult to manage. When you give the problem that larger-than-life stature, it feels like you have to come up with something equally big and complex in order to fix it.

But that's rarely true. Just a little organization and planning can fix many big problems.

When you psych yourself out like that, you start getting bogged down in the details and creating solutions to problems you might not even need to.

Like we've discussed before, the best fix for this is to start small. When you embrace the kaizen philosophy, you can create something small and imperfect now and then improve it over time.

Basically, don't try to solve problems you don't actually know you have yet. As you start to operate your system, you might find that the things that need fixing are completely different from what you thought would need to be fixed when you started building it.

That's the kind of insight that comes from taking small, methodical steps instead of trying to fix everything at once—which, usually, is impossible.

If I don't find any big problems in those first obvious places, that means the system is being used (or at least *was* being used) but something inside of it is causing the problem.

The first thing I do in this case is look for choke points. And the first place I look for them is in the system steps themselves to see if any of them are too big.

So, I'll look at the whole flow of the system and try to figure out how big *most* of the steps are. A sort "average step size." And then I compare that average against each step to see if any of them are considerably larger.

This is a kind of subjective measure. I'm asking, does one step take a lot longer or require a lot more thought or work to complete than the rest of them?

If I find one that's really outsized, that's probably the culprit. That's where things are most likely to break down because the flow of the system gets interrupted. It's humming along and then it hits this big step and gets bogged down. That abrupt change can be frustrating, which causes you to give up and abandon the system altogether.

If you're looking at one of your own systems, this is easy enough to find just by walking yourself through each step and asking, "Where do I normally give up?" or "Where do I feel the most friction?"

This is really obvious when you think about it, right? But when you're in the middle of it and life is happening and you're busy and the kids still need baths before bed and your boss is having a bad day… you know, it can be hard to get yourself to sit down and think this stuff through.

But if you make the time for it, you'll see how easy this kind of problem is to spot if you actually look for it.

And the fix is simple enough, too. Make the step smaller. Split it up into multiple steps. A step in a system can almost always be divided further.

And even though you're doing the same amount of work, it feels like the system is flowing more smoothly, which is a lot more satisfying, so you stick with it.

If your steps are all about the same size and they aren't too big, then the next place to look is in the step *outcomes*.

We've talked about this before, and it's similar to the very first troubleshooting tip we went over. Just like the whole system needs to have an end goal that's within your control, the same is true for each step within the system.

If you have a step with an outcome that might change or be variable even if you do the same work each time, then you have a choke point because it makes it hard to tell when the work for that step is complete.

When I first started writing, there were just a few steps in my whole process. I would decide what I want to write, outline the article, write it, and publish it.

That was it!

And it worked okay, but I got stuck all the time and the quality of my work was really volatile because I didn't have any specific outcomes for the steps.

When was I done "writing"? At 500 words? 5,000 words? Some other metric? Would there be images? What kind of proofreading needed to be done?

It was all too vague. And that uncertainty made it hard to focus on writing.

So, eventually, I decided to break that step up into many different steps. And my first draft wasn't complete until I'd addressed every point in the outline.

Looking back, that seems pretty obvious, but it's easy to make those kinds of mistakes.

Once that problem is addressed, I look for problems in the step transitions. Now, this can look wildly different depending on how complex the system is and whether you're the only one using it or there's a whole team involved.

But some of the issues I'll look for here are too much time between steps, steps that are out of order, or unclear handoffs and transitions.

There isn't anything inherently wrong with a system that takes a long time to operate and has lots of time between steps but, when that's the case, there usually needs to be some sort of notification system to remind you or let the next person know when it's time to take action because, as time passes, it gets easier and easier to forget about it.

You might have the best of intentions to use your system, but you just forget to. In this case, calendar reminders or other tools that can creates cues to remind you to take action help.

Steps that are out of order are usually easy to spot because they throw everything into chaos. Sometimes, you notice right away that you can't do what the system says to do next because you haven't completed something that's further down. That's easy to spot and fix.

What's harder to see is when two or more steps can theoretically happen in any order, but the best order that makes everything easiest isn't immediately obvious. The system just doesn't work that well, and you get stressed out using it even though, technically, it works.

You have to be specific when you're looking for this kind of problem—which is why I recommend you follow the order we've been troubleshooting in so far.

And, in team systems, handoffs between steps can become a big problem. So, one person finishes a step and someone else takes over. That's a handoff. If it doesn't happen really smoothly, that can cause a system to stall.

Now, lots of times, the problem is actually further up the chain. It's something we've already addressed, like not having clear step outcomes. If that's the case, once the outcome becomes clear, it's easier to facilitate handoffs between team members because everyone understands when one piece of work is done and another begins.

Other times, it's just poor communication. One person finishes their step and the next person never realizes it's their turn to get started.

This is where better notification systems can help you—like moving your system into a project management platform where it happens automatically. You can also address this kind of problem by improving your chain of custody.

We talked about this in detail in a previous lesson about how to build team systems, so re-visit that lesson if you think you might be having trouble here.

The last question I ask on my troubleshooting list is simply, "Is the system too simple?"

We want to keep our systems as simple as possible, but we also have to balance that with reality. Sometimes, we need to add detail and complexity in areas where things just aren't working.

For instance, sometimes there's no way to guarantee the output of a step in your system. I have a promotion system for my writing where I pitch my articles to different outlets to try to get them to cover them or reprint them.

Since I can't force an outlet to cover my work, I can only take some actions that I think will get their attention. But that's not actually good enough. If "get attention" is the goal of one of my steps, that's going to cause a breakdown because sometimes you just don't get any attention at all. That's how it goes.

So, I need to add conditional logic. Maybe I'll wait a week and send a follow up message. If that doesn't work, then I move onto the next step in the system or head down a different path. There, now I have an unambiguous step that won't cause the system to stall because I finish the work but don't have a clear way to move forward.

These kinds of little tweaks can be made all over a system to speed it up, remove ambiguity or stress, and ensure the results you get are more reliable.

If you've gone through all the other troubleshooting steps, then this is where you'll want to spend your time. Looking for ways to add conditional logic to problem areas so that, whatever result you get at the end of a step, you know what to do next.

Alright, so here's what we learned in this lesson:

First, we learned the primary reason most systems break down is due to a lack of motivation. If you're not motivated to do the work in the first place, then building a system for it won't help much.

But then we learned that, when you do have the motivation but you're still struggling, there are 7 things you can look for to help you find and fix the trouble spots in your system. You'll want to make sure you have control over the end goal and that the goal is really specific and measurable.

Avoid making your system too complex from the very beginning. Don't try to solve problems you haven't actually experienced yet. Look for choke points. Make sure no single step of the system is too big, the outcomes for each step are really clear, and the transitions between steps are tight. Once you've gone over those things, you can start to make smaller adjustments to account for any variables that might make the system smoother to operate.

If you go down that list each time you run into a problem in one of your systems, I can guarantee you'll find the issue. And once you know the issue, fixing it is usually not that much work. Most of the work is in the diagnosis.

Go ahead and apply this troubleshooting plan to one of your own systems right now. Pick one that you've been struggling with and work through the steps until you find the problem. Then, make a plan to fix it.

And when you've finished that, I'll see you in the next lesson.