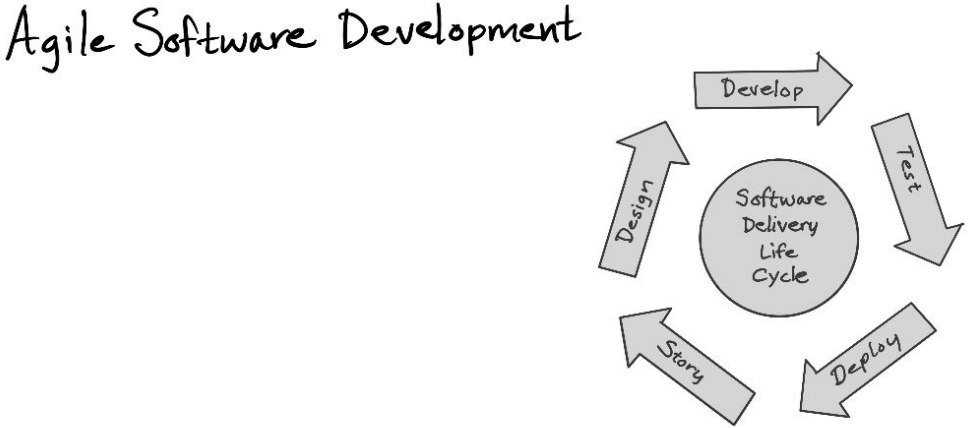
Improve Software Delivery

Need to integrate operations into the lifecycle

Agile Practices

Agile software development was a revolution when it came along. Working on stories through a design development test and deploy loop, and small iterations, allowed us to steer code in the direction of customer demands, rather than trying to predict the future.

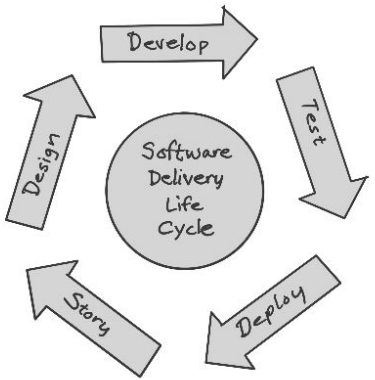


Personal Computing

It's easy to forget that agile software development grew up in the age of personal computing. We had small teams. We used an installer to deploy. Our customer was one person working on one computer.

Agile Software Development
grew up in the age of

Personal Computing



Small teams
"Deploy" was an installer

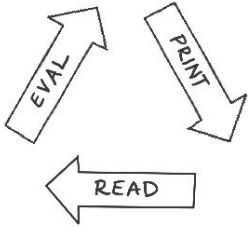
Eras of Computing

Situated in the larger history of computing, we can see agile is the second of three eras of computing characterized by their feedback loops. The read-eval-print loop allowed LISP, APL and Unix programmers to learn from what happened on the computer to figure out what to ask next. Small teams in the iterative delivery loop in agile, and armed with repeatable tests, allowed us to steer our code direction of customer interests. But we've entered a third era of computer where learning from incidents is the characteristic feedback loop. Teams of teams are serving customers with whole data centers.

Three Eras of Computing*

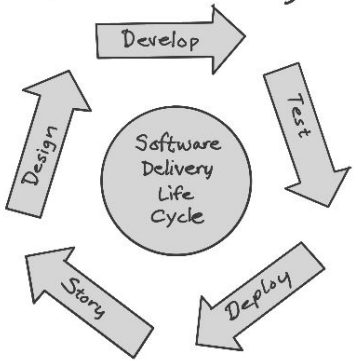
FEEDBACK loops capture the style of an era

1. read-eval-print loop



one programmer
LISP APL Unix

2. iterative delivery loop



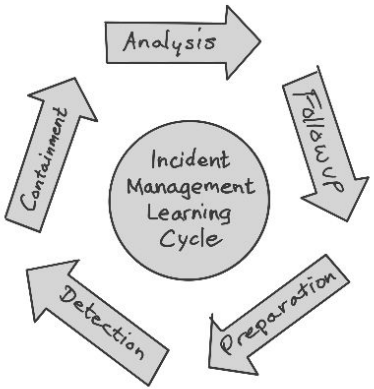
small teams
repeatable tests

3. learning from incidents
teams of teams
Serving customers with whole datacenters through the Internet

* credit to Ward Cunningham and Tim Tischler

Operations

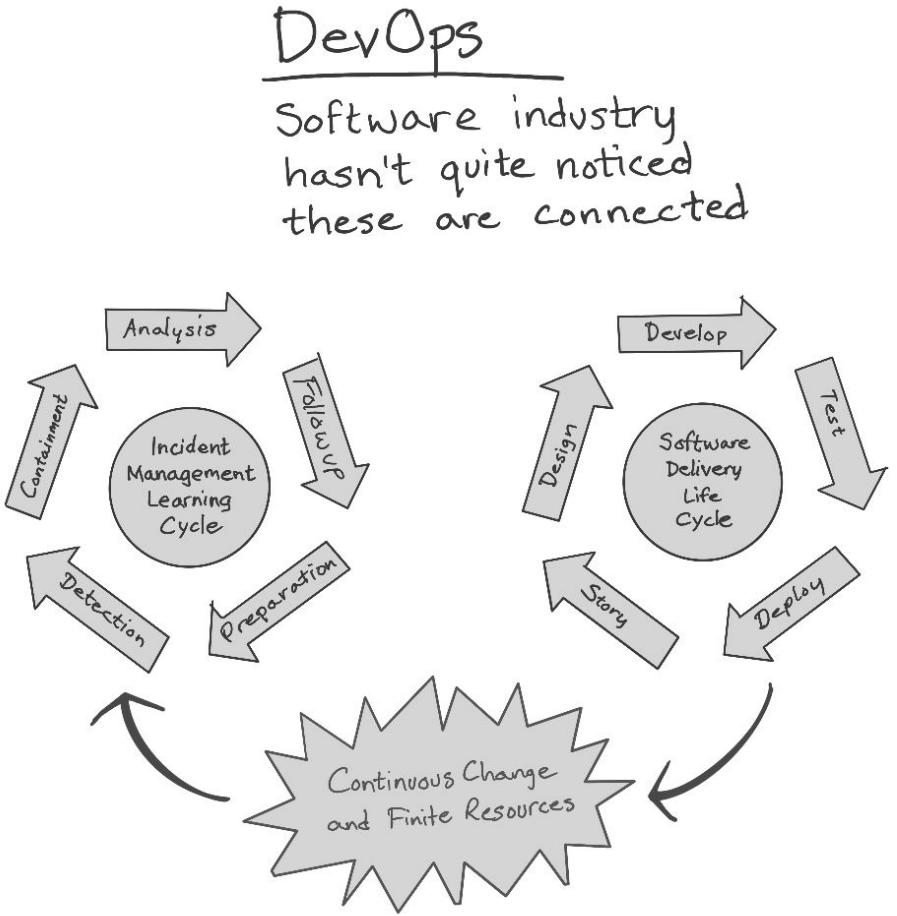
Let's talk about incidents. It turns out the operations side of the organization has had to manage incidents for decades. We have had some amount of preparation, detecting when things are going wrong, containing the damage, and then, once contained, doing analysis, and follow-up to improve the system for the next surprise.



Operations has always needed to manage Incidents

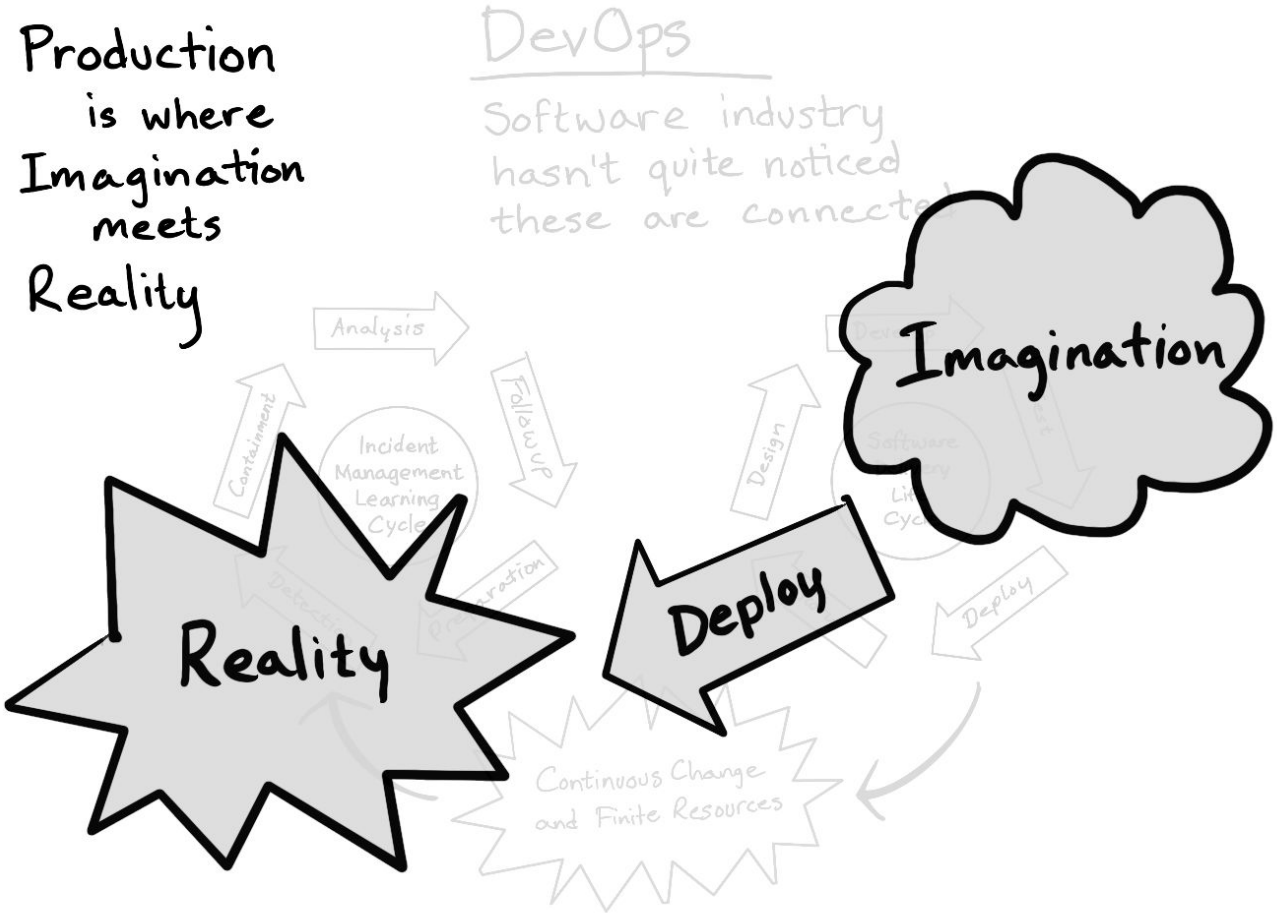
DevOps (and SRE)

The last 10 years, we've had these two cycles actually connected without sort of noticing or really understanding the implications.



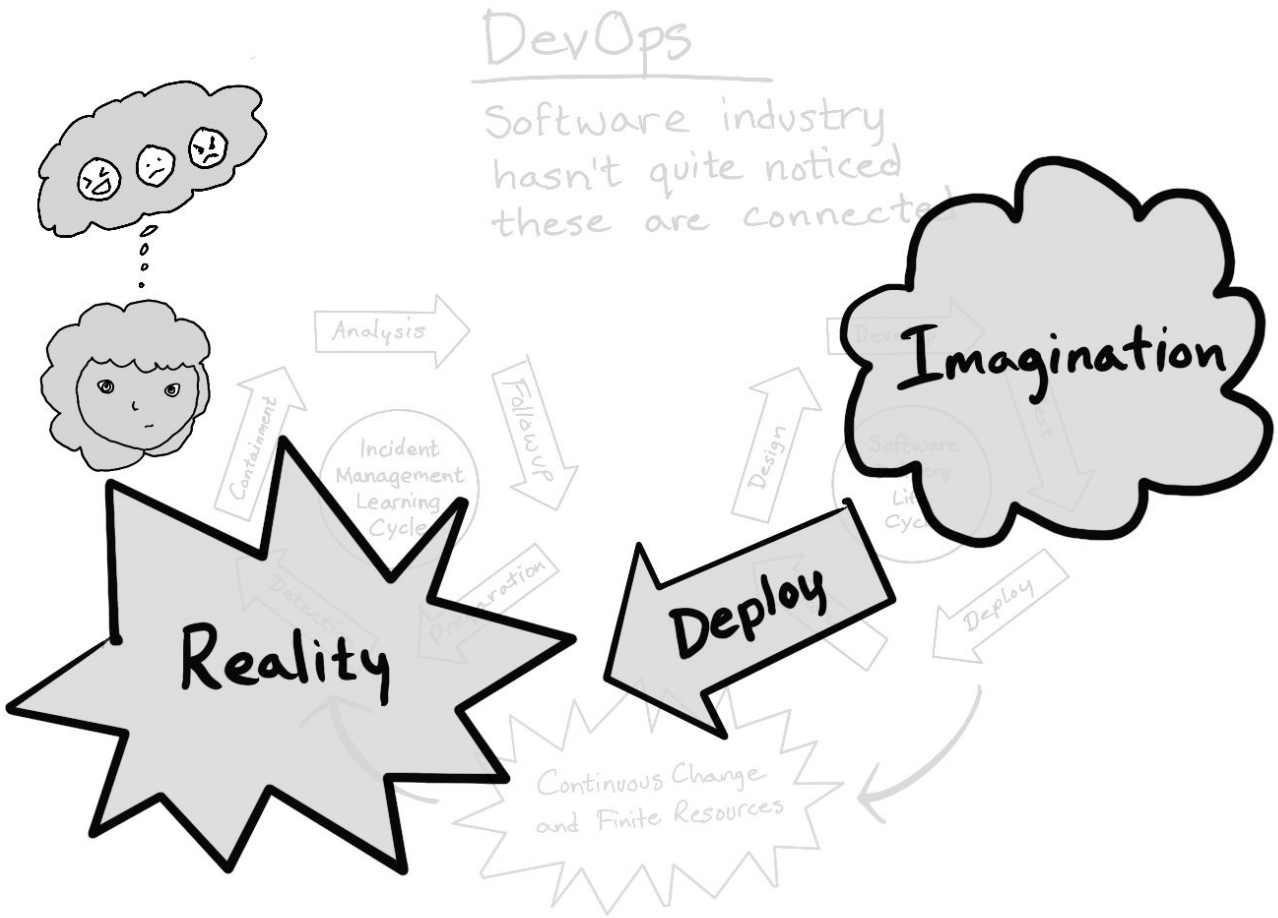
Production

Over on the right-hand side, where our software delivery life cycle lives, is our vision for the future: what we think we're going to be providing to customers. When we deploy that code, our imagination meets reality. Production is where all the sharp edges are and where we discover that clouds, in addition to being billowy and pretty, also have lightning, sometimes tornadoes and hurricanes.



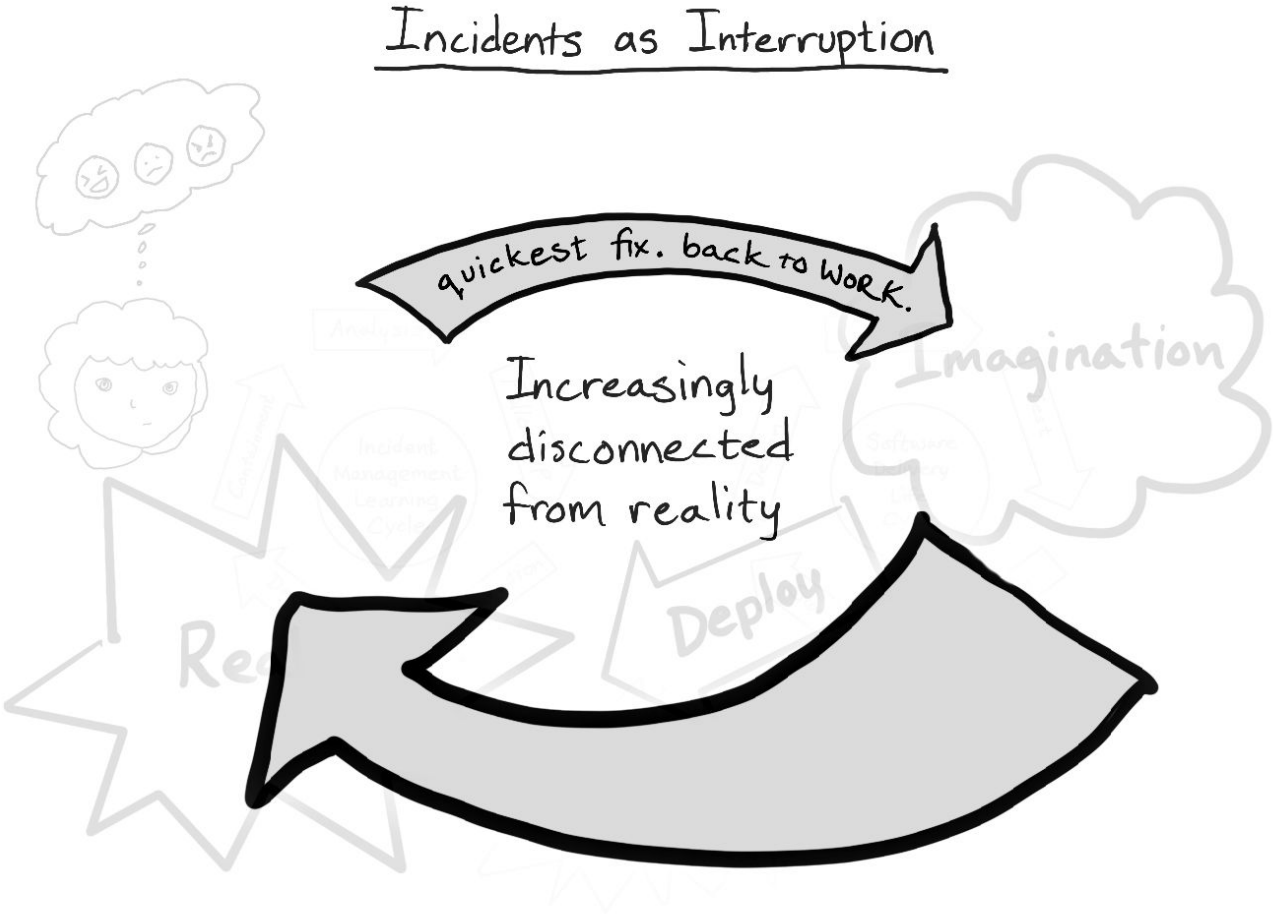
Customers

Also as important, our customers are over here in reality. Production is where our plan actually meets our customers and we either delight them or bore them or enrage them depending on what we've done with that system.



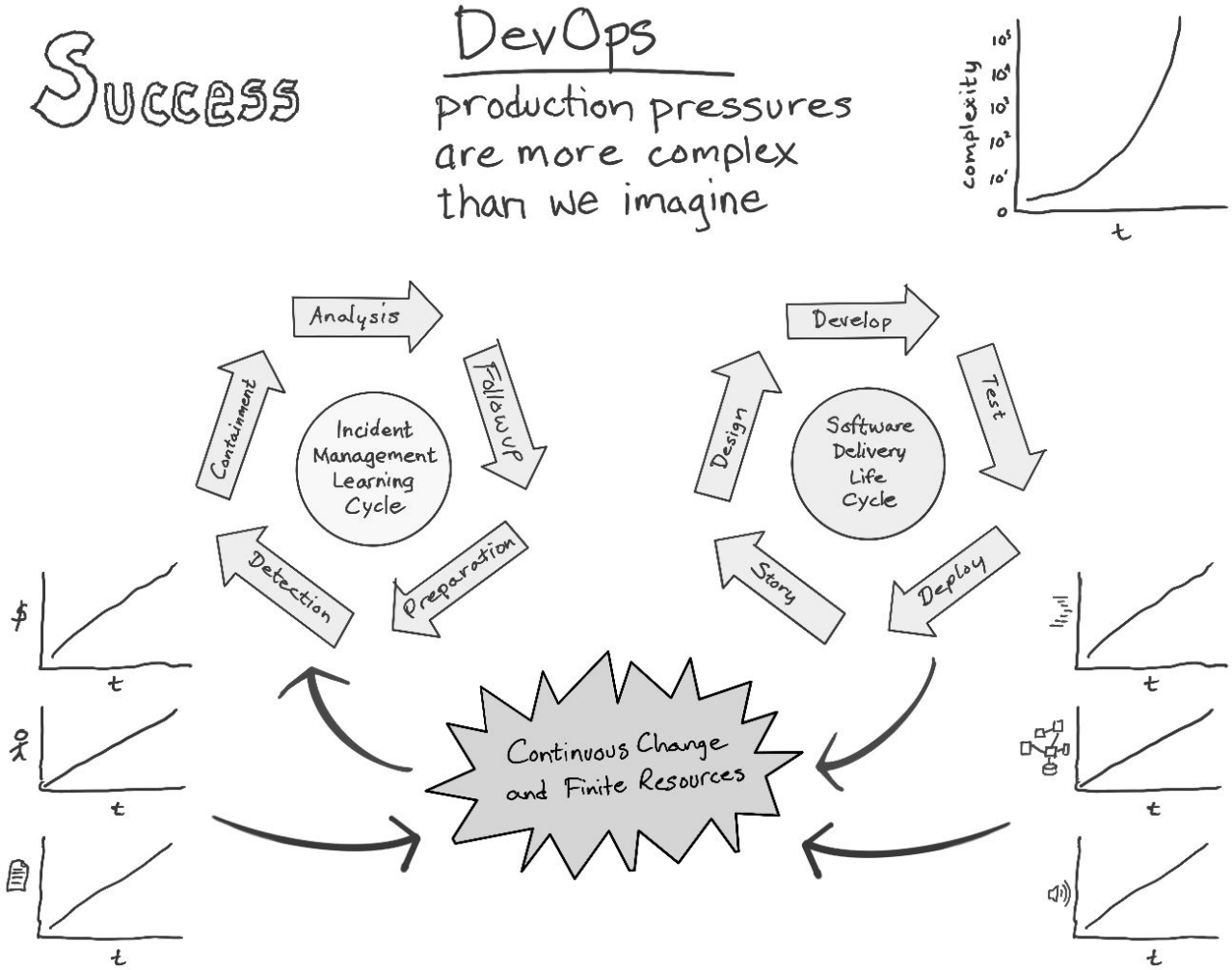
Interruption

Our standard playbook for managing incidents in this current world is to take the quickest fix we can find and get back to work. We look at the software delivery life cycle as the center of the universe, and the incidents are an interruption. We want to get back to work. But when you draw the arrows this way, it's pretty clear that we're getting disconnected from reality.



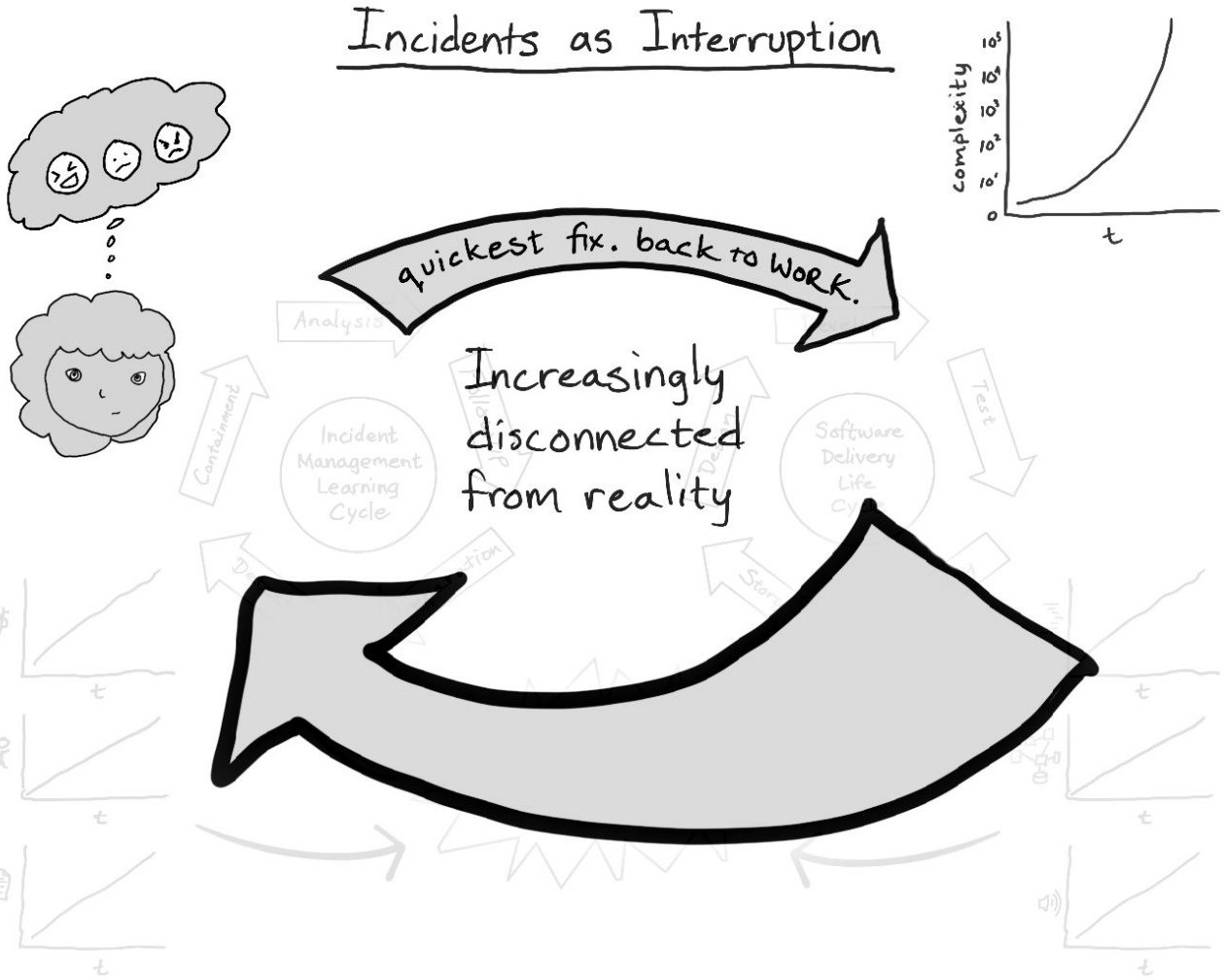
Complexity

And the reality is more complicated than we thought. So we're succeeding. We've got a growth in revenue. We're bringing in more customers. For that matter, we're hiring more employees to manage the demand. We've got much more data coming from those more customers. We keep writing more code. And we write the code into more components in a complex system. Some of those components are the ones that deploy the other components into the cluster. All of that's growing. And to all of that, we're adding different metrics, and events, and logging, and traces, alerting around all of those things. All of those things are impinging on this continuous change that is feeding the incident feedback loop. We've got all of those things, multiplying together, creating a combinatorial complexity. If you know a log-linear chart, you can see this is growing faster than exponential. The important thing, if you don't know a log-linear chart, is that those kinds of problems are actually out of reach of automation.



Not Listening

Let's reflect on this standard playbook for learning from incidents. We take the quickest fix to get back to work. We're not noticing the incredible growth in complexity and effectively not listening to the part of the system that's closest to our customers. This standard playbook for incident learning is basically steering the system at enraging customers rather than delighting them.



Investment

What we need instead is to invest in incidents so that we can continually recalibrate what we think we're building based on what's actually happening over in production. This calls for a much different kind of incident analysis than the standard playbook.