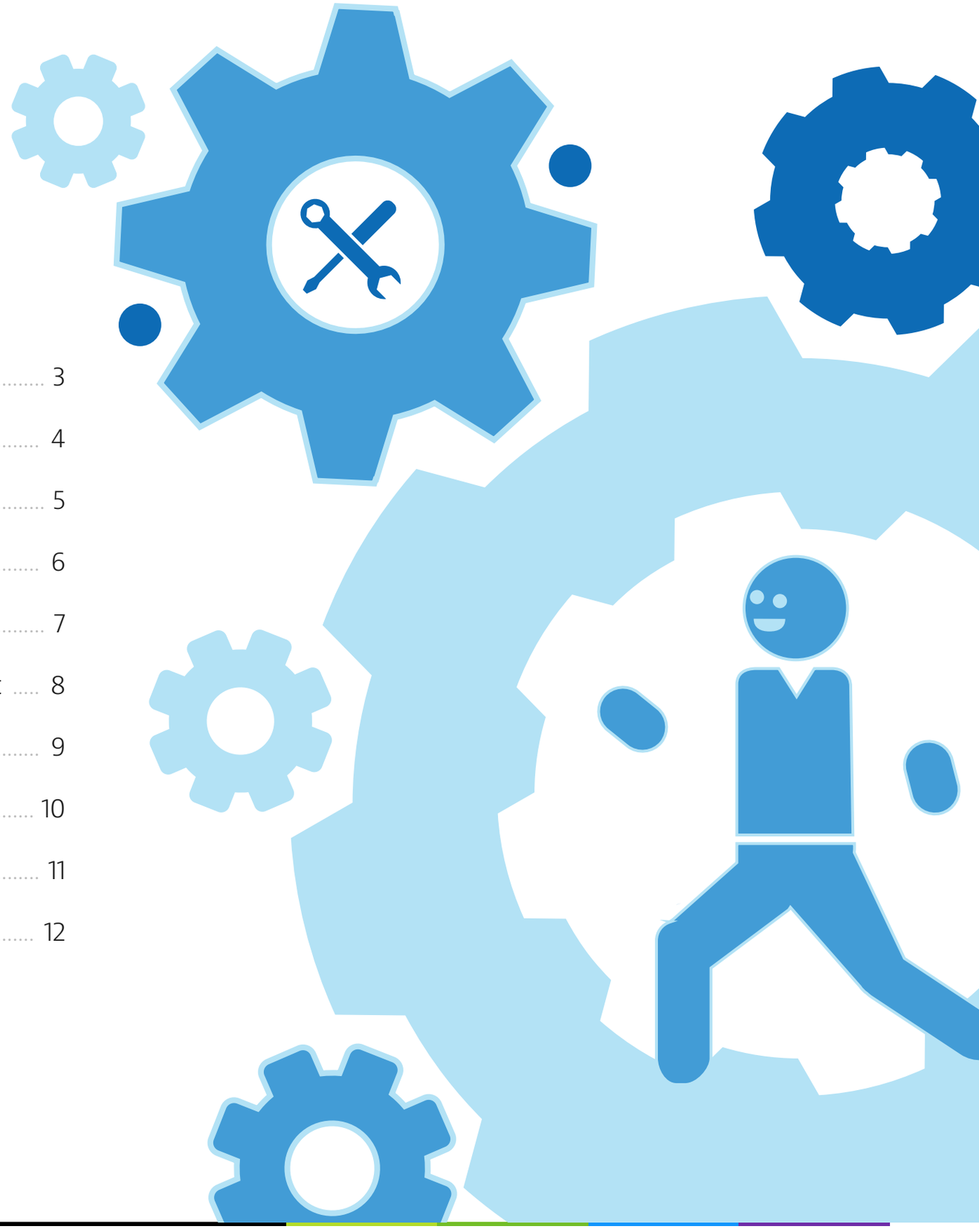
dynatrace

6 tips to increase DevOps collaboration and improve performance

Recommendations for all: Business, Engineers, Test, and Ops… and even your ego

Table of contents

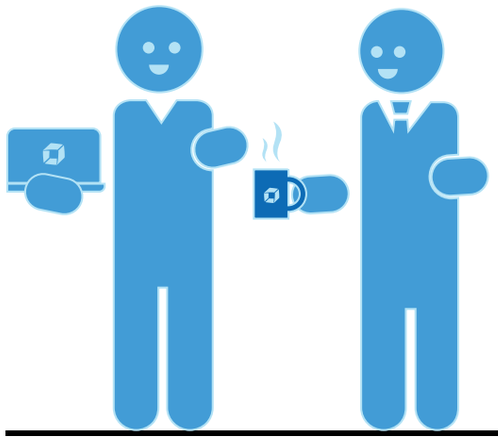


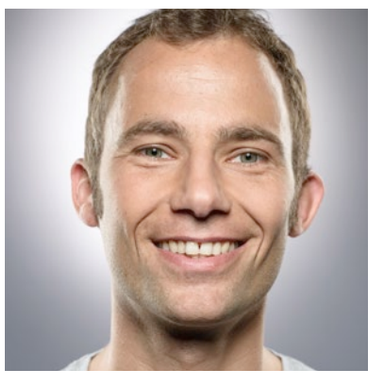
Our authors:

The DevOps experts

When apps fail, whose fault is it? In today's DevOps world, every stakeholder along the application delivery pipeline is accountable for performance.

Our authors have accumulated many years of experience improving performance. In this eBook, they will share important tips to increase collaboration and improve performance for each role on your DevOps team: the Business, Engineers, Test, and Ops, and last but not least, your own ego.





Andi Grabner

Performance Advocate at Dynatrace

Blog: blog.dynatrace.com
Twitter: @grabnerandi

Andreas Grabner has 15+ years' experience as an architect and developer in the Java and .NET space and is an advocate for high-performing applications. He is a regular contributor to large performance communities and a frequent speaker at technology conferences, regularly publishes articles on blog.dynatrace.com.

Mark Tomlinson

Performance Scientist and CEO of West Evergreen Consulting

Online: perfbytes.com
Twitter: @mtomlins

Mark's career began in 1992 with a comprehensive, two-year test for a life critical transportation system. This was a project that captured his interest for software testing, quality assurance, and test automation. After extended work at Microsoft and Hewlett-Packard, he amassed broad experiences with real-world scenario testing of large, complex systems, and he's regarded as a leading expert in software testing automation with a specific emphasis on performance. Mark now offers coaching, training, and consulting to help customers adopt modern performance testing and engineering strategies, practices, and behaviors for better-performing technology systems.

Setting stage:

What went wrong?

Picture this: it was the holiday season and, one of our eCommerce clients was preparing to launch. They did proper load testing and everything looked pretty good, or so they thought.

They did have a small problem with a sales report. It required 1600 database statements to run that report. But they went ahead anyway with the Go Live. It was just a report and the more critical user scenario was the product order transaction after all. That scenario tested out fine.

When the week of Cyber Monday came along the picture changed dramatically. For the user to run the same report, they had to wait six minutes with 5200 database calls now being made to the Oracle database. Unsurprisingly, this totally crashed the system.

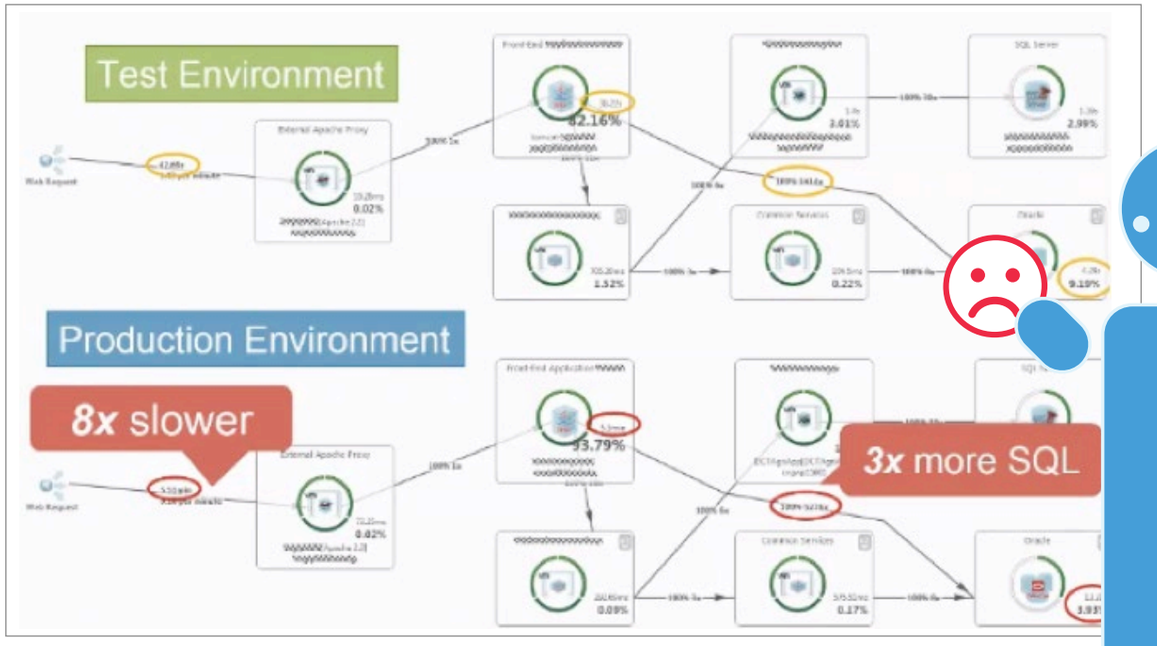
You can see the report transaction flow mapped in Dynatrace in the image on this page. Test environment flow at the top is o.k., but things changed drastically in the production flow below at the bottom.

How could this happen? They did some pretty significant testing in the test environment and production environment, but something was totally different.

In this case, there were two problems. The first one is the number of database statements. This was a straightforward problem of marketing and IT not collaborating. They simply tested with the dataset from last year instead of against an increased forecast for the current year taking into account a bigger product catalog and increased sales.

In the second issue, in the test environment, most of the time was spent in IO on the web server. While this is completely normal, in the production environment the picture was different. The #1 API was Hibernate. What happened was that on the eve of Thanksgiving, somebody decided, "Let's upgrade the libraries in production to the latest version of Hibernate, and let's also swap the XML parts of the views right now. There's something that I read on Twitter that is much more efficient." Ouch.

These last minute changes are against everything we should do, but under stress, these things happen. The developer upgraded to a Hibernate version with a well-known performance bug causing the system to crash on the most important week of the year.



Tip #1 for your ego:

Individuals don’t win the game

In Andi’s example the developer, Mary, made the decision to update Hibernate. A bad decision right before the big day based upon bad information.

I don’t know about you, but in times of information overload, especially during a stressful time like the holiday peak load, I can easily understand Mary being overwhelmed. She didn’t have the right information nor did she know who to talk to before making the change. So how did she make the decision? By information she found on Twitter.

In the DevOps world, how do you make sure Mary has the right information at the right time to make the best decision possible? That’s the first tip: Individuals don’t win the game.

We use the phrase often – check your ego at the door. It’s the idea that we all need to be a bit selfless. Mary probably got herself into this situation because she didn't feel comfortable talking with the development team or with the database development team or by talking to somebody in the marketing department.

In these situations, everyone’s in panic mode. We're all concerned for ourselves so that we can get through holidays without getting fired for a major crash. Leading up into these panic modes, it's really important to have built relationships. You need to check your ego at the door during these times. Individuals don’t win the game. The whole team does.

To make this happen you have to collaborate, especially in performance because much of the information is spread across teams. It is sure sign that something is being done wrong if the team is under micromanagement. When there's a bigger-picture concept, like performance and application end-to-end infrastructure, you can’t chop it into too many small pieces. You'll no longer be able to make decisions with the greater good in mind.

You need to check your ego at the door during these times. Individuals don’t win the game. The whole team does.

Tip #2 for your Team:

Good DevOps teams are accountable

A DevOps team, a good DevOps team, wants to be accountable. Accountability is an essential part of making the whole performance picture work.

It used to be that when things went wrong there was a lot of hunting for who was to blame. “Well, it wasn’t me,” says the development department. The database department says, “It wasn’t me. It was them.” And eventually, somebody points the finger at the network guys. It always ends up with them.

This was all caused by centralized decision making and hierarchical transference of power, which is another way of saying bad things roll downhill. DevOps is a complete rebellion against that. Everyone is partially accountable, and no one is 100% accountable. We all have shared accountability and have to work together.

The flow in the decision making process is another thing that is happening in DevOps. It's the idea that you’re only accountable for this information at this point in the process. You provide this inspection. You do the analysis. You make these projections, and the assessment. Then, you pass it on to the next step in the flow. Maybe it's manual or maybe you use some tools that help things move along.

This is the modern-day approach with automated, continuous releases. The people in automation work together to remove roadblocks and to remove the silos from the old way we used to build software.

This is what I like about the new DevOps paradigm. It changes the status quo to be more about taking more ownership and accountability for the output of your work – you're just one part of the greater process.

In the earlier example, Mary's case with the last-minute Hibernate upgrade, she was disconnected from the flow of information and decision-making. If she knew something from test results, maybe she would have never pushed the Hibernate update. The flow was broken, and the accountability was broken in that chain. It was not a DevOps way of working.

It used to be that when things went wrong there was a lot of hunting for who was to blame.

Old Paradigms:	**DevOps Teams:**
· Hunting for who to blame	· Shared successes and failures
· Competing silos and departments	· Everyone is partially accountable
· Centralized decision making	· A semi automated flow of "decisions"
· Hierarchical transference of power	· People and automation working together

Tip #3 for Business:

Build for the performance needs of your users

Business analysts or product owners, are often measured by the number of features they push out on a release.

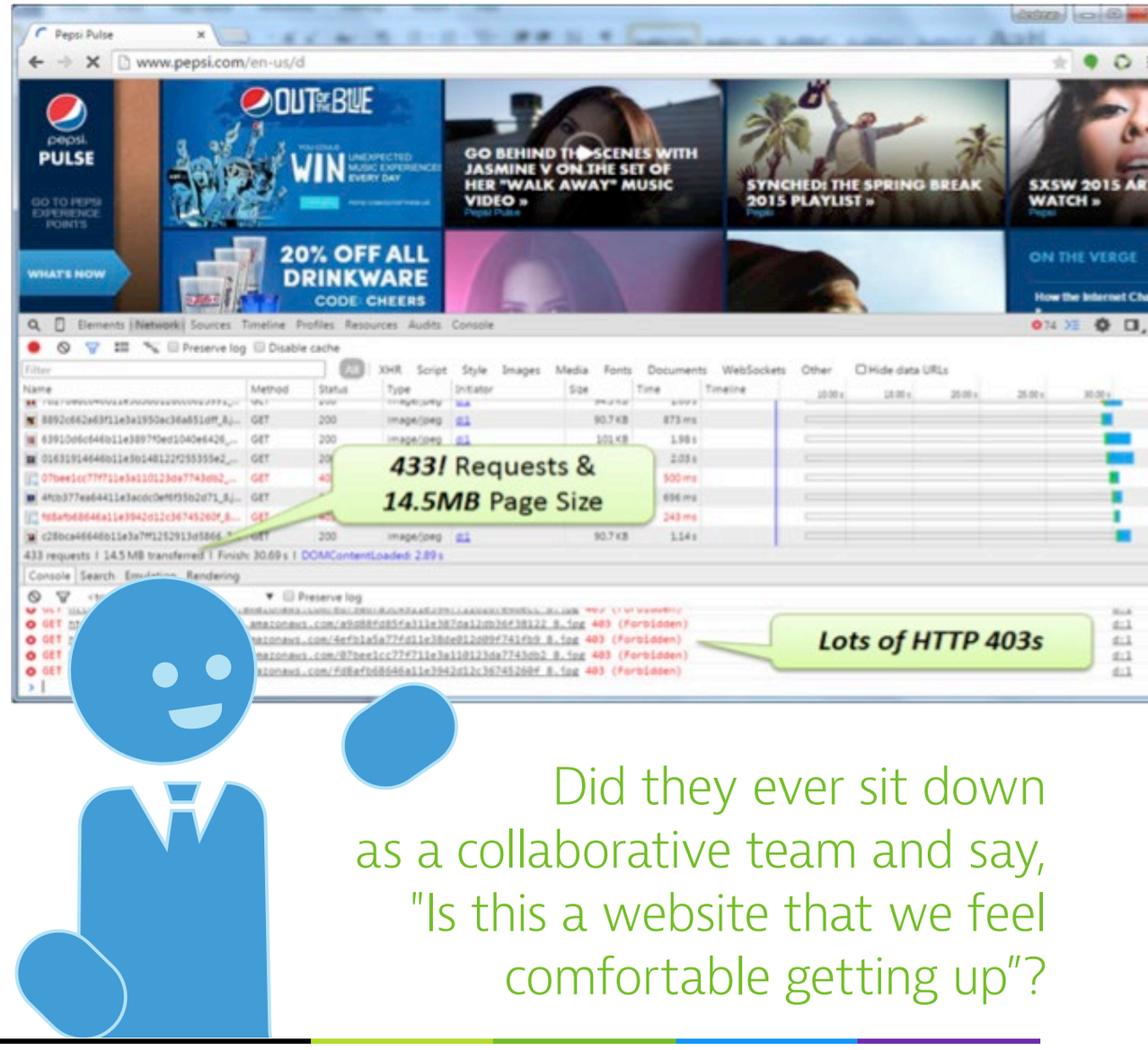
What they should really be measured against is how much value they deliver to the end user. Unfortunately, the status quo is still feature driven, because features make money. When things go wrong, bad inputs get into the delivery pipeline. It's click paths that are too complex or too many features in one spot.

What I also see a lot is designers coming up with great ideas, but not understanding what's technically possible to implement. The mock-ups are overloaded and too complex and can never be implemented within the available UI frameworks. That's why designers and developers need to work together to figure out what the constraints are and what the real options are.

This is still the status quo – this feature pushing when we are trying to bring out cool new feature ideas without ever really sitting down with real end users to figure out what they really need. So the tip is: keep it short, think lean, clean and responsive. Involve your end users and engineers in the process. Focus on the 20% that makes sense for 80% of your users, and not the other way around.

Here is an example at Pepsi.com (in the image). The design is beautiful and the marketing idea is great but from an implementation perspective, I often wonder, did they ever talk. Did they ever sit down as a collaborative team and say, "This is a website that we feel comfortable getting up"?

There are 433 requests to download the page, the page is 14.5 megabytes, there are way too many individual images, and a lot of HTTP 403s due to missing events, resources, and a misconfigured CDN. If it were a good DevOps team, the Ops guys would be empowered to go back and tell the business the site can't go up because it isn't working or better yet, all of these issues would have been caught earlier in the development process.



Did they ever sit down as a collaborative team and say, "Is this a website that we feel comfortable getting up"?

Tip #4 for Engineers:

"Built-in" quality to avoid built up technical debt

The current performance thinking of engineers, the developer-architect group, is to implement as many story points as possible because that's how they get paid.

Another problem is that these engineers sometimes put bad input into the pipeline. Too often I see prototypes making it all the way into production, because then nobody takes the time to say, "Now, let's do it right."

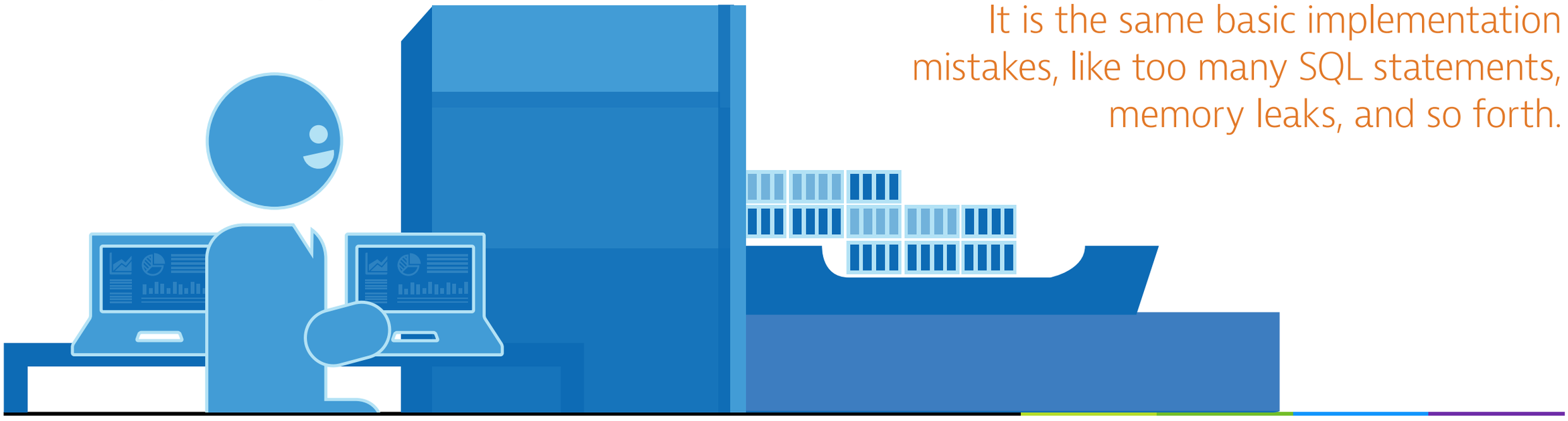
I see some strange things when I analyze performance problems. It is the same basic implementation mistakes, like too many SQL statements, memory leaks, and so forth.

I think it happens because we're just rushing through features and story points. We do a prototype and the product gets put right away into production without anyone thinking about the bigger picture later on. I think this is something we need to change as well.

There is this movement toward micro-services to compartmentalize development so each feature can be isolated and development is fast and clean. I've worked with a company that tried to do this: they re-architected their product to micro-services. The problem was that it wasn't done well. When we analyzed the site performance, we found way too many web service calls, calls to the database, and connections. Then we saw the database pumping out all kinds of exceptions.

They needed to put this kind of performance analysis into their agile work during their stand ups and code reviews, and during the architectural reviews. Not after the fact, when their users were affected.

The bottom line? Build in quality from the start so that you are not building up technical debt later on and spending all your time on firefighting and working support cases.



It is the same basic implementation mistakes, like too many SQL statements, memory leaks, and so forth.

Tip #5 for Testing:

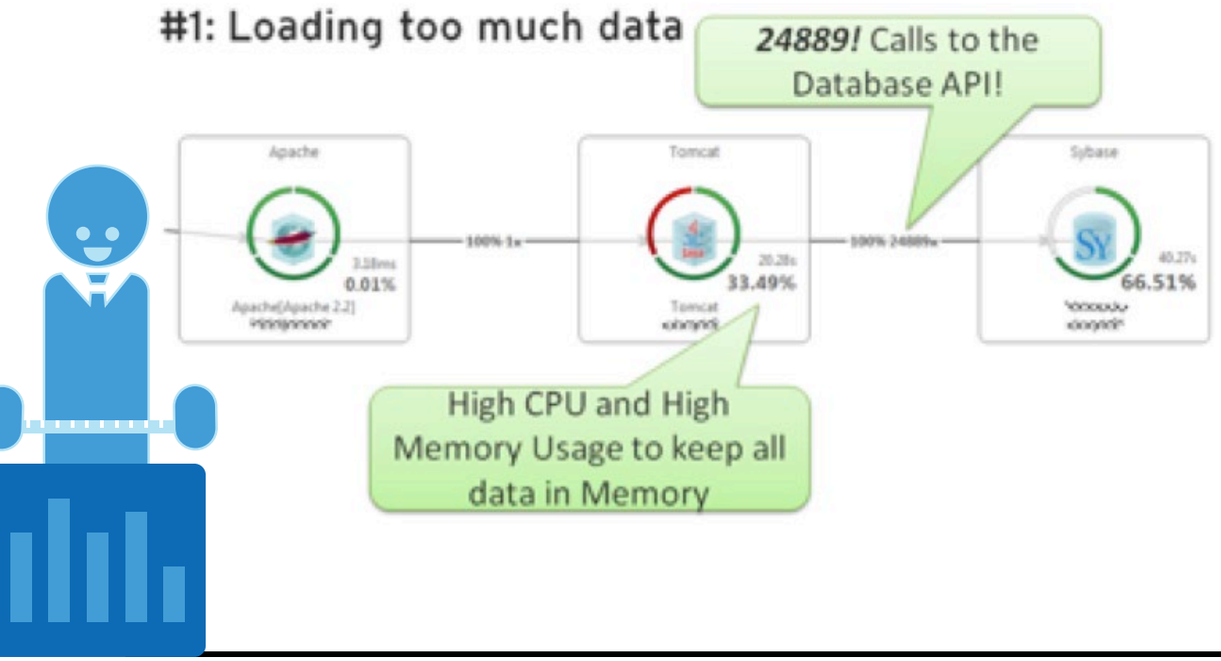
Level-up by adding development tools to your work

I'm a software tester by heart. That's where I started.

And it's pretty simple. Software testers should not be measured against the number of tests executed or the number of bugs reported. It's the wrong metric. If you want to push software faster through the pipeline, automation is key.

A lot of testers are very good with testing tools, but often the reports don't make any sense to engineers. As an engineer, if I see a LoadRunner report, what do I do with it? It doesn't make sense to me.

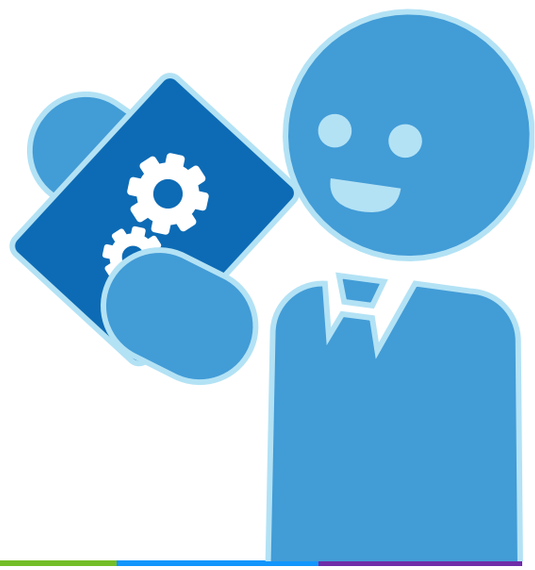
Testers need to sit down and collaborate with Engineering. *What metrics do we really need in our testing? How do we do testing? And, how can we become better than just being measured by the number of tests executed?*



Testers need to level-up. One way to do this is to not just look at the functional problems you find, but also look behind the scenes. Become familiar with Tomcat, JBoss, Spring, or ASP.NET. Understand what a connection pool is and whether it is good if we are making 24,800 database calls for a simple Login transaction. Testers should get used to looking behind the scenes and instead of just saying functional green or red, should be able to say, "Hey, from an architectural perspective we think something is going wrong here."

So there is no excuse anymore because the tools are there. All the APM vendors, including Dynatrace, have free trials that you can use. We even have the personal license now, so as a tester you can sign up for the free trial. And, after 30 days, it converts to a personal license and you can use it as long as you want on your local machine. No more excuses.

The tip for Test then: we all need to level up. We need to add development tools to our work, to better communicate the issues instead of just creating bug reports. Sharing expertise is key.



Tip #6 for Ops:

Provide feedback loops

I see a lot of applications that make it to production that are configured incorrectly.

For example, Tomcat is deployed into production with a default connection limited to a database of 10 connections for the pool and then there are thousands of people coming in. I also see database queries being executed that are not at all optimized, even though, of course, it worked well on the developer machine, on the small demo database. In real life, things are different.

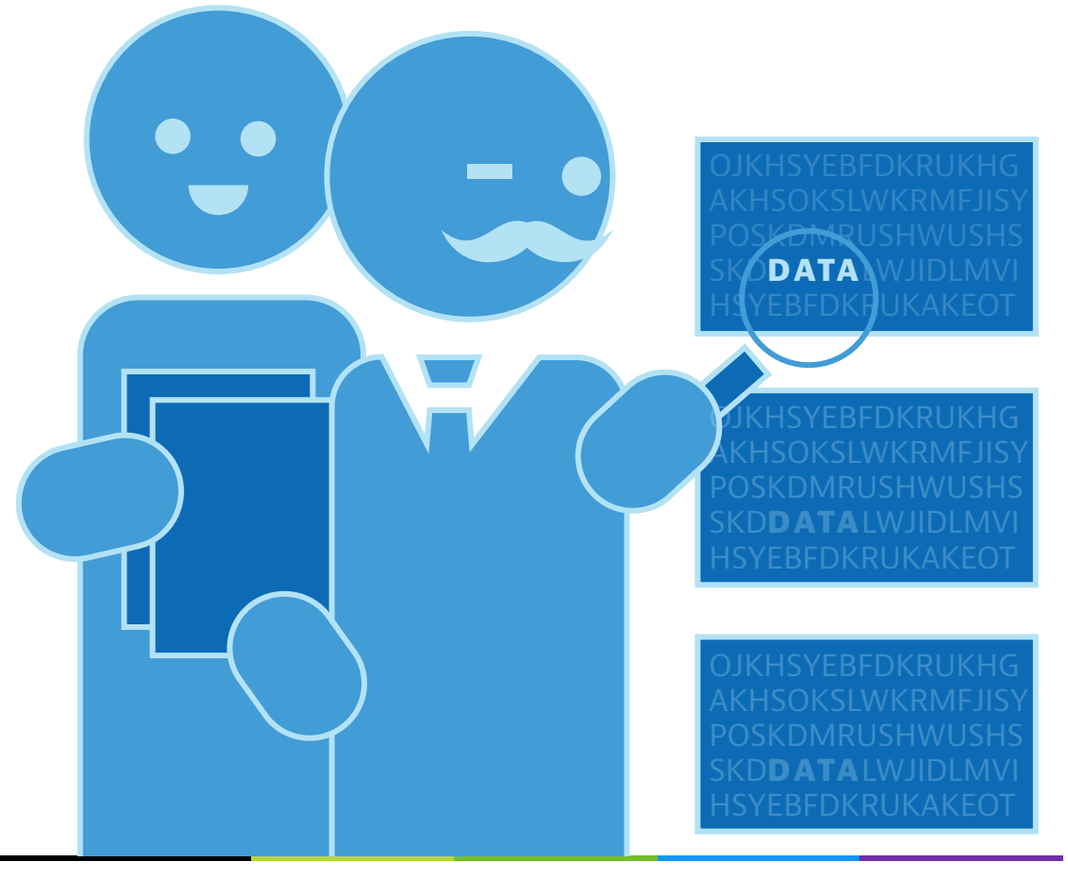
If system and database admins are just looking at up time and keeping resources within acceptable limits, it is not enough. They also need to look at the bigger picture. They need to reach out to their engineers and say, "Hey, let's sit down, look at the data you want to query from the database, and the most efficient way to do it before it gets to production."

Then there is operations. I mentioned earlier the example of Pepsi and browser caching not done right. That all made it into production, and nobody thought about how to correct it. They needed to set the browser cache settings so that every request was not delivered again to the data center, properly configure the CDN, and so on.

This is when operations needs to think differently and collaborate with the engineering team so that right from the beginning, all these best practices are already applied early on and tested. The Ops team must be accountable for providing that feedback to architects and developers on what the real world looks like in production. They are part of feedback loop.

So the tip for ops is: provide guidance back to development and test. Give them the insight into what it's like in the real world, in production. Ops has a great role and responsibility to level-up each member of the DevOps team with their expertise.

"Hey, let's sit down, look at the data you want to query from the database, and discuss the most efficient way to do it before it gets to production."



Summary:

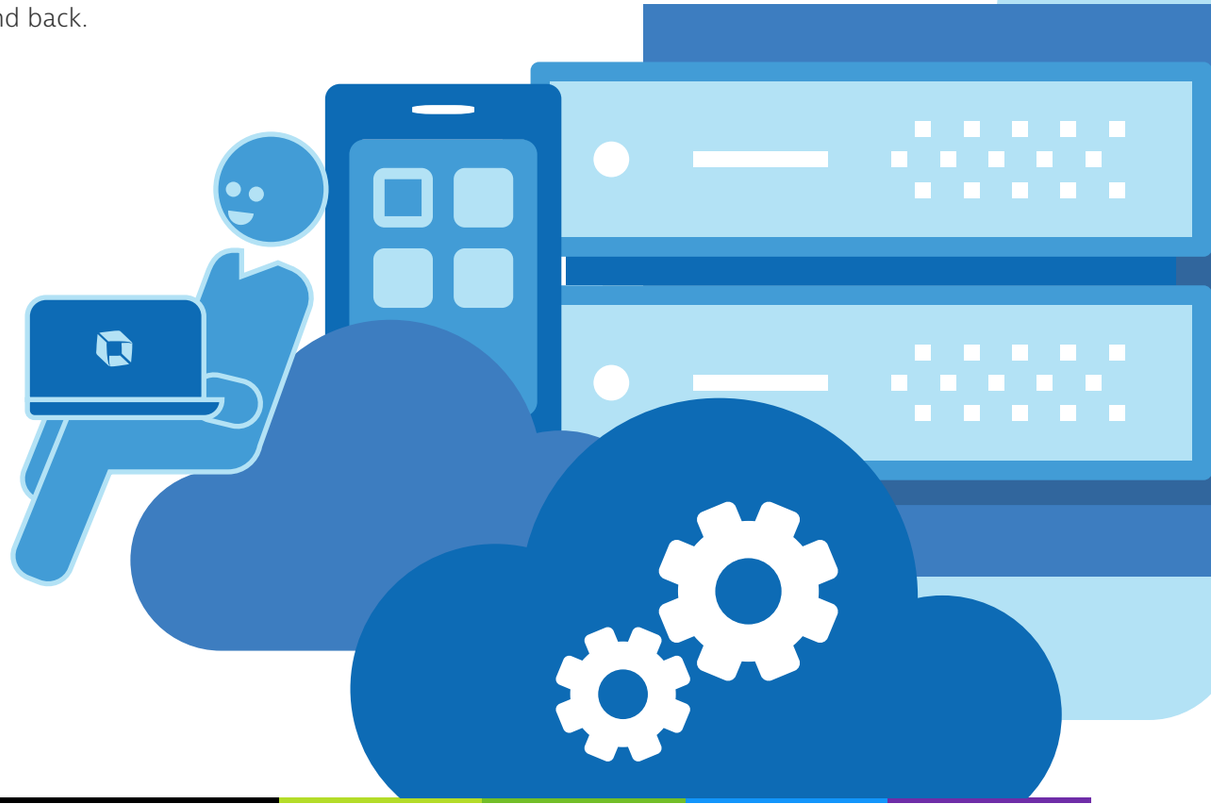
How to avoid the dysfunction junction

In the DevOps world, these are the kinds of behaviors that will put you on a quick road out the door.

I've seen really high-quality engineers and testers end up not staying very long at a company because they committed some of these crimes.

There are four of them.

1. **Anti-collaborators or refusal to cooperate –** This means anything from, "I know I have some expertise, but I'm not willing to provide it," to "I could roll up my sleeves and get my hands dirty, but that's not my job." That will go all the way to the top of IT management if not business management, that somebody's not being a team player. Be careful that there may be some unspoken or unwritten cultural rules around collaboration, and this is no place to be an anti-collaborator.
2. **Self-excusal or ambiguous accountability –** If you don't know what your value is to a team, you should point the finger at yourself first. If you know that you're one of the best people in the company to provide information about database performance, you need to give it even though you are working in the marketing department and not IT.
3. **Exclusive participation –** Don't go off secretively and do some project and don't tell anyone and all of a sudden you're getting all the kudos. Politically, that's not very smart. Your colleagues may not want to invite you in the next project. This approach destroys collaboration. Instead publish all the documentation, give a brown bag on the project, do it in the open. Invite everyone. Don't be exclusive.
4. **Deficient communications –** Don't leave out critical performance information. If you are not a good verbal communicator, do it via email. If you need to do it in the company Wiki, do it. Critical performance information about the marketing campaign needs to go from the business to operations and development and testing teams and back.



Additional resources for building a DevOps performance pipeline

The biggest system failures have on thing in common: dysfunction. DevOps practices can help bring the team together to leave egos at the door, work collaboratively and use performance metrics to insure a quality product hand-off at each stage of the delivery pipeline.

To be successful, you must re-define what it means to be accountable in your team, contribute to the information flow between roles, step up your testing practices by providing more information sooner, and quantify the value of your performance improvements.

We hope the tips in this eBook have given you some ideas on how to improve. Below are some additional resources to help you on your DevOps journey.

DevOps tools we love

Here is a collection of tools that foster collaboration amongst Product Management, Development, IT Operations, and Technical Support teams to help you to build more quality into your products, and support you in establishing better feedback loops.

- **Configuration Management —** Ansible, Chef, and Puppet
- **Test Automation —** LoadRunner and Selenium
- **Change Controls —** JIRA
- **Development —** Eclipse
- **Source Control —** Git/GitHub
- **Build Automation —** Ant and Maven
- **Virtual Machines —** Vagrant, Packer, VeeWee, Docker, and Cloud Foundry

Webinars and a Podcast

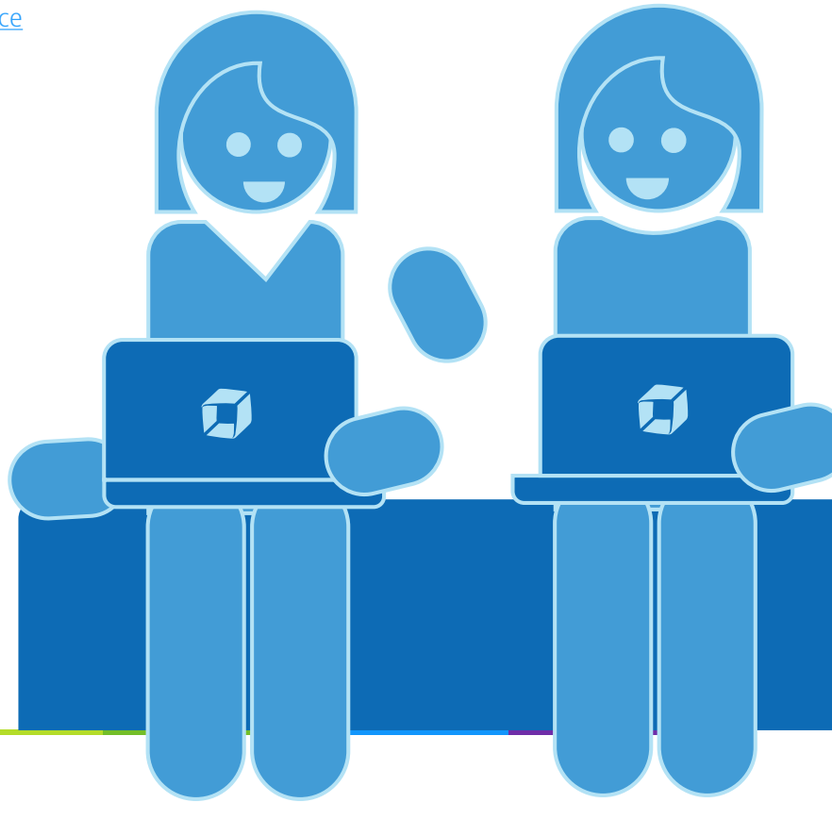
- 7 Key Metrics to Release Better Software Faster
- 6 Ways DevOps helped PrepSportswear move from Monolith to MicroServices
- Dynatrace's Pure Performance Podcast

The blogroll

- DevOps reactions — Enjoy some DevOps humor!
- IT Revolution's DevOps
- Software Quality Metrics for Your Continuous Delivery from Dynatrace

Recommended reading

- *Continuous Delivery* by Jez Humble and David Farley
- *Release It* by Mike Nygard
- *The Other Side of Innovation* by Vijay Govindarajan



Learn more at dynatrace.com

Dynatrace is the innovator behind the industry's premier Digital Performance Platform, making real-time information about digital performance visible and actionable for everyone across business and IT. We help customers of all sizes see their applications and digital channels through the lens of their end users. Over 8,000 organizations use these insights to master complexity, gain operational agility and grow revenue by delivering amazing customer experiences.

10.25.16 885_SixTipsToIncreaseDevOps_jg