Workiva

TestOps in a DevOps World

Simon Howlett

Good morning..

My name is Simon Howlett, Product Development Director at Workiva, and welcome to ‘TestOps in a DevOps world’ where I’ll be talking a little about integrating your automated testing workflow and resources alongside a devOps team and and a continuous delivery infrastructure.

A little on Workiva..

- Founded in 2008
- Provides a cloud-based SaaS platform for collaborative reporting across multiple industries
- Around 1000 employees in 14 locations, with the main HQ in Ames IA
- 65% of Fortune 500 using Wdesk currently

A little about the presenter..

- Director of Product Development for Workiva Quality Assurance organization
- Based in Portland Oregon but originally from Preston, England
- Previously worked on projects for Barclays Bank, National Grid, Aegon, FujiFilm amongst others

Workiva

Workiva

- Workiva provides a collaborative, cloud based solution for distributing complex documents and data, especially those required by regulatory boards such as the SEC
- All of our offerings are cloud based, from a development and testing perspective that offers some interesting challenges, ranging from security, data protection, to performance and browser consistency..

Myself,

I run what gets called the test engineering, or TestOps, team and as such I get to oversee testing process and infrastructure for most of our product teams, 15 engineers keeping our test platform running all day every day, also ably assisted by a 70 strong QA Department

- I am based right here in sunny Portland, originally from the even rain soaked north west of England.

‘Need for Speed’

Agile mindset promotes frequent software releases to customers;

- Less arbitrary process
- Less manual checking
- Less time to market

- Quicker Feedback from customers
- Faster release cycles
- Increased fostering of innovation

Workiva

- Need for speed – something we as company hold dear, as do many others.
    - Essentially it encapsulates the fast feedback cycle all product owners love,
        - get the minimum viable product in front of stakeholders, and customers, through continuous delivery.
        - This mindset can be an interesting challenge for a QA Analyst
        - Automated Testing is essential, and must be reliable and trusted
            - Understanding of what needs testing, how, and when is paramount, the usual ‘test everything, all the time’ mindset can be a tricky one when you could in theory be releasing updates multiple times every day.

            - To do this properly you need the support of a number of people…

Supported by;

- **Product teams**
  - accountable for their own releases & decision making
  - encouraged to iterate, fail fast..
- **DevOps**
  - 'Frictionless Development' practitioners
  - helping product teams adhere to 'Need For Speed' mentality
  - providers of deployment, hosting & monitoring infrastructure

workiva

Product teams.

- Small teams, owning their own release schedule.. Which adds an Accountability – they decide what gets released when, they don't have to wait for a large, scheduled release

- DevOps – Many organizations are turning to devOps teams to manage development and continuous delivery platforms,

- Frictionless Development – easy workflows to get products out – no bottlenecks, spending as little configuration of environments as possible (turning to things like virtualization & Docker…)

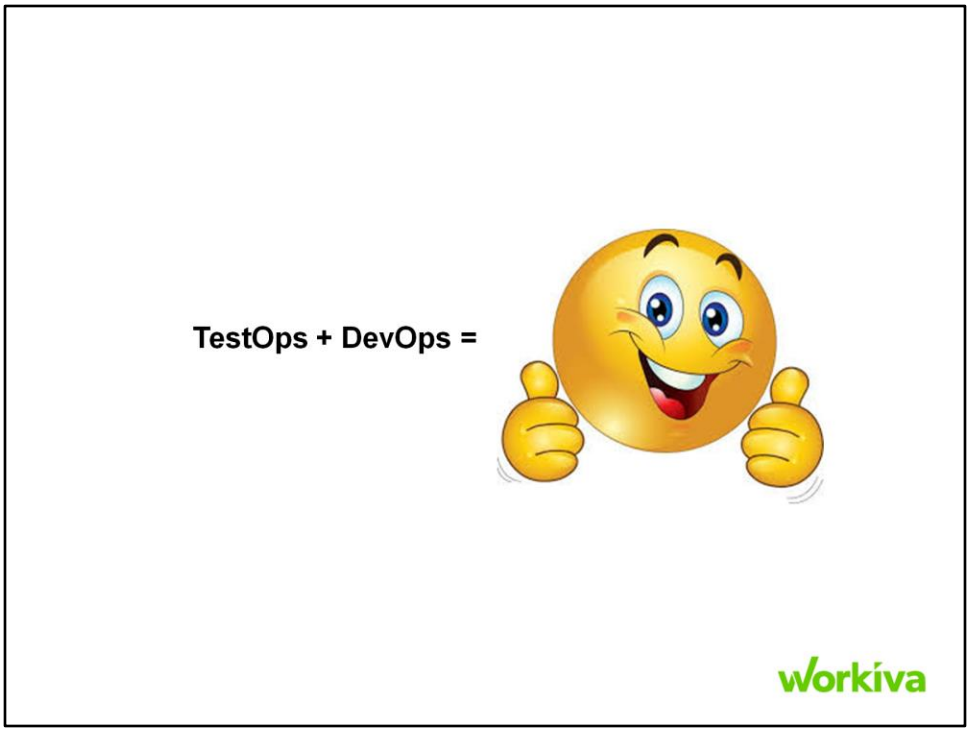
- DevOps teams can tend to oversee any/all of the environments needed from development to deployment of products, but not always the deep mystical chasms that hold QA environments

- **'TestOps'**
  - Responsible for enabling all aspects of testing
    - Testing Environments & Infrastructure availability
    - Integration with DevOps infrastructure
    - Test Coverage & Reporting
- **Release Management**
  - Audit tracking
  - Expectation Setting/Evaluation
  - Release consistency

workiva

- A TestOps Team own the provision of any and all testing resources,
  - Owning Testing Environments, Automation frameworks and everything that goes with that, actually very similar to DevOps and their provision of environments, which is really why its important the two teams are in sync, and don't appear as separate workflows in the delivery lifecycle.
  - 'Integration' with Devops here here meaning interacting with their CI items such as your build system, code repositories, and even up to to the push to production, this changes QA from a separate process at
  - Reliability is critical as getting buy in for owning quality in a product team as whole, rather than just the QA person, is a cultural shift and any hint of flakyness in your chosen system is usually not beneficial
  - Coverage & Reporting, produce easy to read, standardized reporting that can be provided to auditors to support releases
  - the end of development to a part of the same workflow development is using, which is crucial to a quick CI system
  - The simple way to think of it.. devOps looks after all our pre-production and development environments, and the support mechanism enabling items to be ready for promotion to production, testOps owns the testing resources WITHIN that same system

- Release Management
    - A trusty release management team is a necessity in this conversation also, they are the point of contact questions on what went out when internally and from auditors, ‘how did you test this release, how were these issues treated’ – this information all comes out of devOps and testOps platforms so it needs to be easy to follow and automated where possible.



So, (and apologies, but you will be seeing a lot of this guy)

The theory of fitting DevOps and TestOps teams together into an continuous delivery workflow seems pretty sensible? Hopefully it should be something easy to implement?

Well, tread carefully… in reality moving to such a workflow poses a few challenges, at least for traditional QA as it requires..

Infrastructure changes
Expectations on who owns what in the testing part of the software lifecycle, and changes to the parts of the process QA usually delves into
Often leading to a pretty large cultural changes on the part of most teams..

If you would indulge me a little, lets take a look at how we at Workiva ended up at this point…

Early 2010…

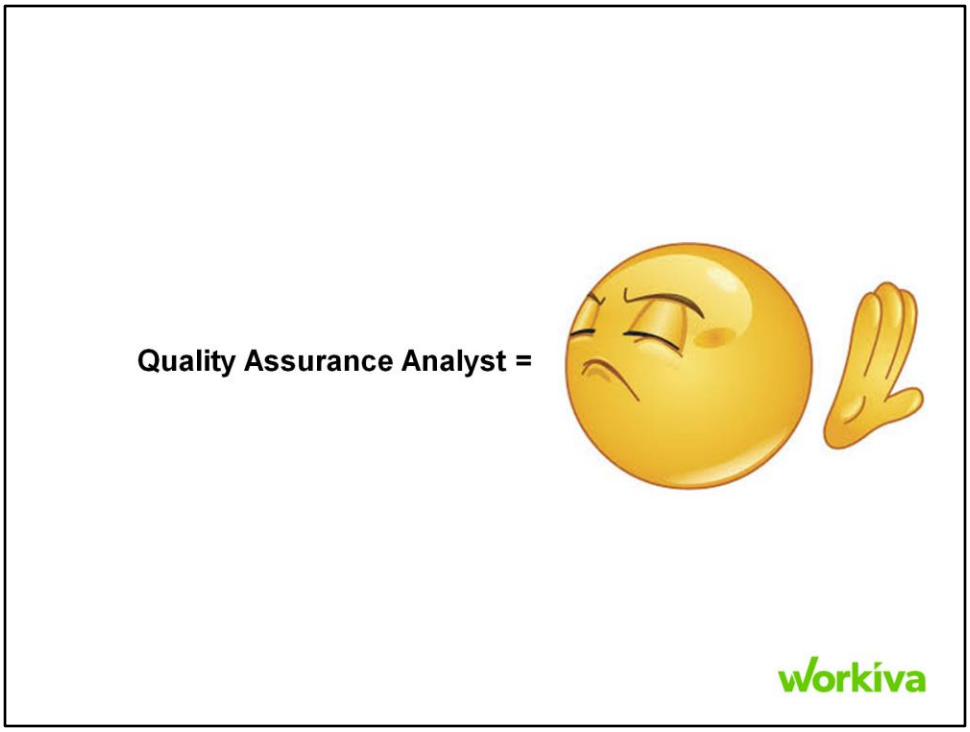
- 1 product
- 2 week release cycle
- 1200 new items per release
- 5-8 QA Analysts
- 800 Manual Test cases
- 4/5 day regression testing cycle per release
- regular hotfixes

workiva

In the start up days, starting with a few testers, we naturally evolved from ad-hoc manual testing of beta products, to a large set of manual test cases on a reasonably settled product feature set.

These tests took around a week for 8 of us to complete and we managed to release on a fairly rigid two week cycle, with a huge amount of changes in each release, though usually with a hotfix or two each day to fix bugs we'd not spotted. Within some of these tests were a lot of things that really shouldn't have been done by the human eye, things like large document comparisons

Luckily for us we has a base of pretty talented QA people who were also on very good terms with their development counterparts, so we managed to uphold a pretty decent level of quality, considering.



As you can imagine, a QA Analyst wanting heads down time to test, constantly led to some pretty interesting conversations, and the general state of affairs was ‘I’m busy’…

So we needed to do something about that, and quickly.. With those 800 or so handwritten test cases, so we knew the grind that we had to do something about…

Dawn of test engineering...

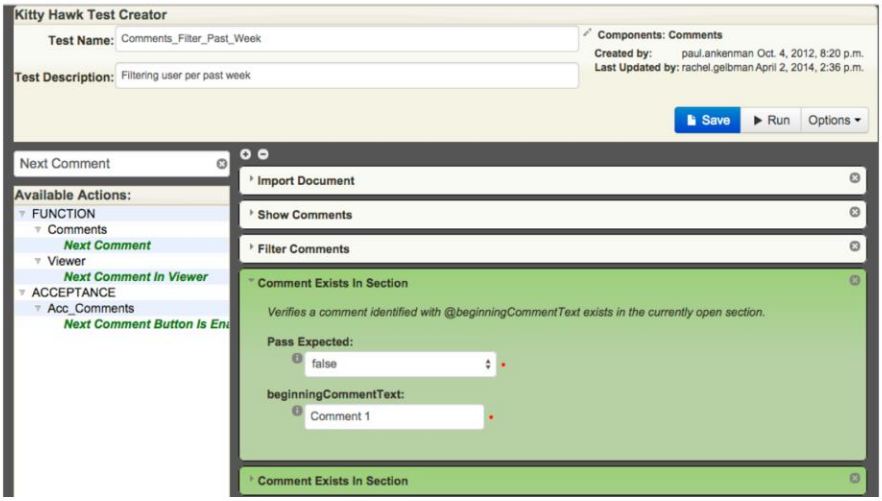
- An Automation / Development Team within QA
- Needed a way for QA Analysts to quickly create their own tests: we built 'KittyHawk'
  - An IDE built on top of TestComplete code
  - Drag and drop interface
  - Comparison tools to check document, XML and image content,
- Added a high tech server stack to run these tests – 'Skynet'

workiva

Luckily we had a few folk on the QA team that were interested in attempting a proof of concept for automating our testing regime at that time. We carved off those guys into an initial 'test engineering' team.

The initial goal was to automate the work our manual testers do,, The challenge was an interesting one, not least because our main consumer application was built on adobe's flex platform, after a couple of attempts with open source utilities, we settled on SmartBear's 'Test Complete' platform

We then set about finding a simple way to replicate our test cases, and also provide testers with a simple way to add new ones



Workiva

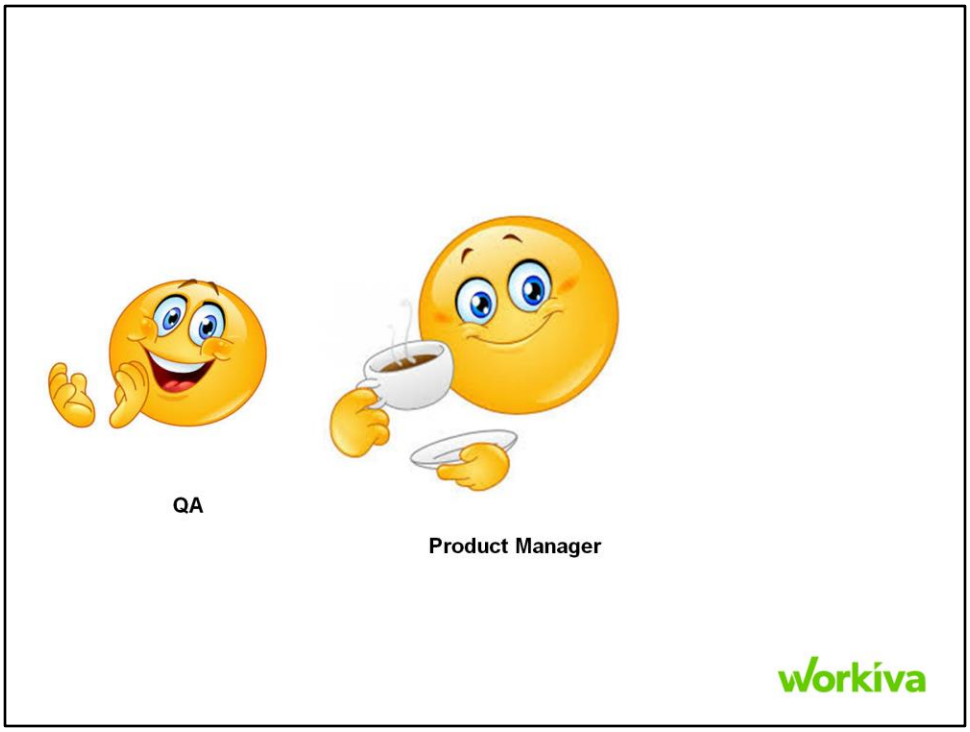
About 3 months in we proved out we could do what we set out to do, we managed to replicate some of our test cases, and build a UI within which testers could add new tests. Here you see a shot of the UI for a drag and drop style that IDE we built on top of test complete, so that our QA folk (and anyone else who wanted to) could make test cases pretty simply.

At this point we then replicated what we could of our manual cases, so our daily builds now had some sort of test coverage – those 800 tests ran on our initial 15 machine setup and took most of the night to run. Step one was accomplished (albeit incredibly slowly), we all patted ourselves on the back, and set about expanding the system, whilst QA set about replicating ALL of their test cases, existing bug scenarios etc.

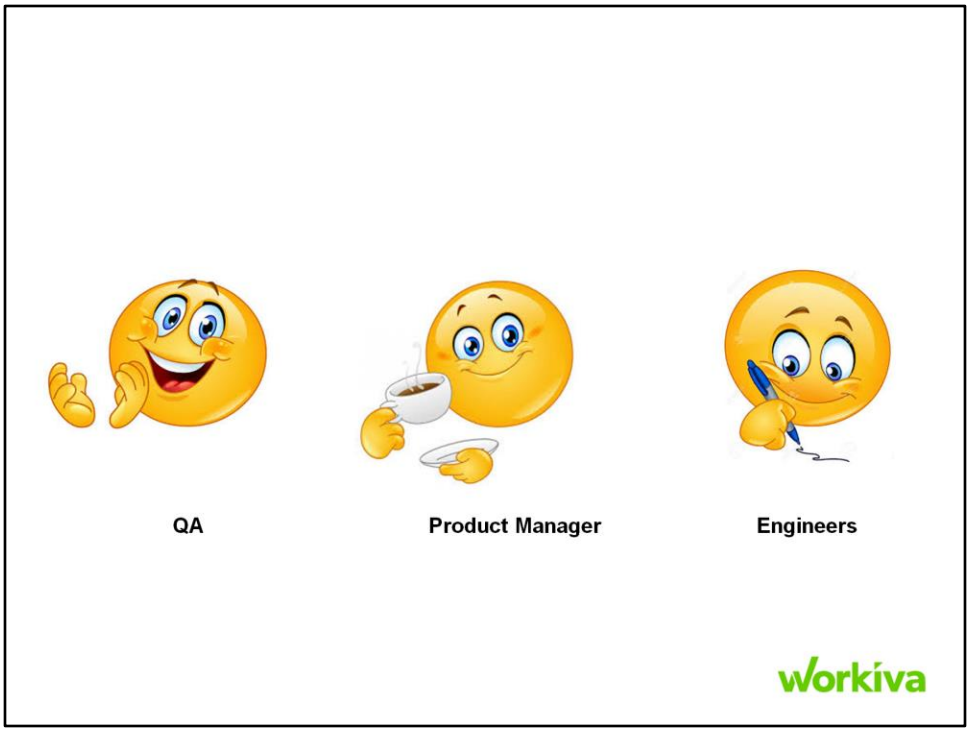
QA rightly applauded, they were getting some of their lives back, and we had something tangible to support each of our releases in terms of auditable test results.

We also added some extra features at this time..

- Screen Recordings
- Results emails/notifications
- Document Comparison tools (MS Office, PDF, XML) to confirm we were translating client documents properly – very important, given what we do as a company, nobody wants an em-dash instead of a hyphen, or a pirate instead of a check box…



And somewhat importantly, my product manager was vindicated in his support of our quest, we got to push harder on what we wanted to do with this new found resource, and we got to pitch our success to the rest of the company..

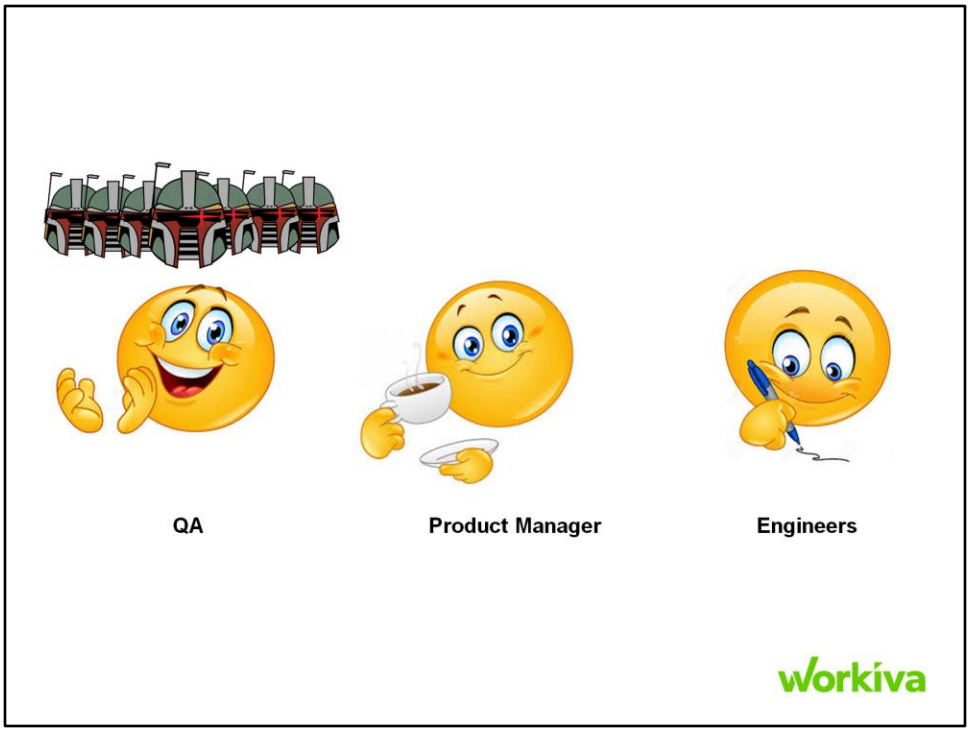


Our Engineers were also thrilled, they got to concentrate on coding, and their QA folk had been freed up to dig deeper

Over time..

Our bug count in production decreased
Our time to production started to go down
Our level of test coverage increased..

So…



With this initial success, we got to increase our investment and reliance on this framework, that came to be known as ‘Skynet’, a nod to James Cameron, and also a regular threat by one of our engineers that he would replace most of us with a small shell script at some point over the course of his career, (he’s not far off, currently..)

We increased the machine resources we had for testing and came up with some fun algorithms to speed up test machine assignment, sped up our tests significantly so we got the length of a full test run down from ‘overnight, at best’ to around 3 hours. This enabled us to do a whole lot more testing and get a lot more efficient at it..

Success?

- Number of test scenarios grew to 7000+ tests
- increased infrastructure to run 1,000,000 test scenarios per month
- built out a full command, control, reporting platform
  - Flex tests
  - Selenium tests
  - Mobile tests
  - API / Integration test reporting
  - Log Viewers
- testing around 1000 updates per month
- consistently reducing costs..

workiva

So, were we successful? (and this is where you guys might spot some red flags.. But I'll come back to those..)

We've settled at around 7000 tests, theoretically giving us a large amount of functional test coverage.

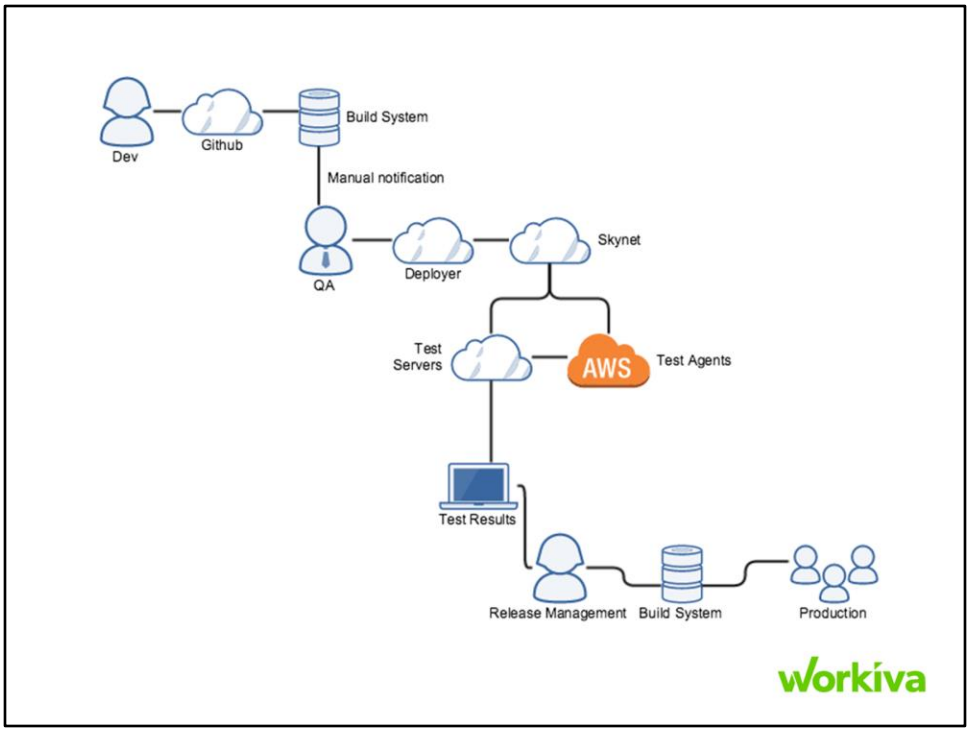
Right now, we run an average of 1 million individual test scenarios, split amongst around 1000 builds per month. These range from hourly smoke tests and release candidates, to development builds and prototypes.

And then the change in release schedule, we went from a two weekly release schedule a couple of years ago to a daily release model, now each night each night our tests run against the release candidate and each morning QA review any failures, providing release management with a go/no go answer on that build.

For the last year or so we've seen costs reduce also by about 5%-10% per month due to efficiency gains from making tests more efficient or moving service based tests into their own frameworks, saving time and making those tests much less flaky.

But at this point, this was still a system in isolation, people have to intervene at various stages in the workflow to get items through the next stage, and there is very

little integration with other teams especially not anything resembling devOps…



So here is our 'classic' workflow..

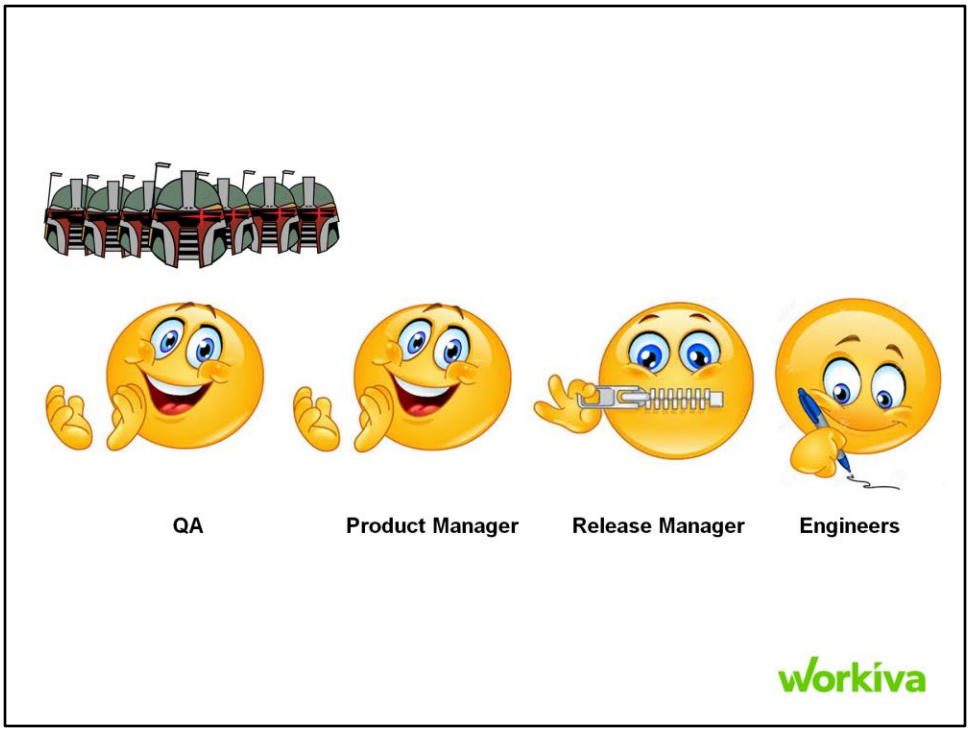
- One of product team devs commits some code
- The build system we have, 'smithy', kicks off the build, any unit tests, and spits out a build artifact if everything completes ok.
- The QA person on the team (or in some cases the dev), deploys that build to Skynet, selects what tests they want to run and Skynet hands out the tasks to starts testing.
- Skynet puts the test results together, sends out some emails and some chat room notifications, along with annotating Jira tickets and GitHub PR's with the results
- At this point Release management is notified this is ready to go (usually by the QA person commenting on the Pull request)
- If all goes well, Release management press the button, and our daily release goes out.

So in theory…

QA has a robot army doing it's bidding The product manager is overjoyed because everything ticks over in a nice hand over system
And engineers can keep writing code whilst QA handle all their testing for them..

Anyone think this this is the reality?

It’s more like this…



Here is what it possibly ended up like..

QA and Product management is pretty happy, they create their own test coverage and as such know where the faults can lie in their applications

Release management look a bit concerned though, and for one this is a broken workflow..

- People have to tell each other to do the next part.. That hand over from build to test, that's a human interaction, one on which we ran the numbers, and there was an average delay of a day between getting items built in dev to getting them into testing in Skynet, Release management actually spend a lot of time chasing these handovers and the detail from them, and that should just be automatic
- The promotion to release management, relies on someone reviewing test results, particularly in the case of master runs, and with teams operating in different time zones that can delay releases by a number of hours, chasing people to look into that last remaining failing test
- Also at this point, all of the systems needed to get software out into the customers hands are owned by different people, IT own deployment, QA owns testing, another group owns the build system, so at any point any break in the process halt everyone until we can find out where the problem is, and also, and this may seem trivial, but its not, everything looks and feels like something disconnected from the

whole, one language for one thing, another for the next tool, different reporting styles, inconsistent and often confusing – something TestOps will look to resolve in working with DevOps

- So there are some obvious challenges..

- Not least that in the main none of this interacts with our new DevOps systems which teams are beginning to use to get their new products out…

Upcoming Challenges

- New Products...
  - Splitting up a a monolithic codebase
    - Microservices Platform Model
    - Independent release model
    - Increased release frequency - many times a day
- Crucially, a new platform for deployment and production services, built and owned by DevOps..

workiva

At the same time as reviewing all this, as a company we decided on a pretty seismic shift in the way we release software.. And this was firmly in the DevOps realm of ownership, something as a QA organization was going to have a massive impact on us.

everything we released up until around 1 year ago, was conveniently contained within one or two build packages, and really we utilized a couple of specific languages and server platforms for everything. This was to be split up into a decentralized micro services model (allowing us to scale more easily)

Teams will then be able to release on their own, whenever they need, outside of the release schedule and without reliance on release management to fit their updates into a timeslot.

This is really the 'need for speed' mantra in all its glory, product teams accountable for everything they do, an interesting challenge for anyone with a traditional QA mindset…

To support this change the devOps team was to build new development environments and deployment infrastructure to support this plan, this gave us a chance to take stock of what we had done previously, and figure out what we could fix by integrating tighter with this new devOps ideal..

Existing Challenges.

- Functional testing = Good!
- Functional testing = Bad!
  - Test code separated from product code

- Flaky tests!
- Testing to high up in the stack
- Hold ups in the process
  - a full test run on a release takes 3 hours
  - delays between build and test - up to one full day!
- false wall created 'my build is done, now you test it'

workiva

Here is what we found with Skynet in its first version, and things we need to fix..

Functional testing has proven great for what we initially wanted it for, it catches bugs, every day, it frees people up to do deeper dives in their products, and it gives us feedback pretty quickly (within reason..)

But, some challenges become obvious over time..

- When people needed test coverage for new features, it isnt simple to do it inline with those features being developed, and the test code itself lives in another repo, which needs to be kept in sync, so that gives us a headache of branching and merging tests to support new features, not simply adding them in the same pull requests as the features themselves – this we could fix by not separating test code repositories from product code.. A first switch to using the CI model to solve one of out issues.

We have also noticed we are victims of our own success..

- the industry wide problem of flaky tests, our master runs usually have around a .1% failure rate, anywhere from 1-30 failed tests, aside from the bugs themselves, that means
  - Lack of confidence in tests – people tire of reviewing the same thing and

become complacent about it, often the first thing to be blamed is the test, not the product

one way to help fix this, is again, containing tests with the code they examine, so hopefully the team in charge adjusts their mindset and has more solid tests (or moves them to a less fragile place in the stack)

- We tend to default to functional testing. Here is one easy example…

we had around 350 or so tests that cover when a document is translated from the format used in our text editor to PDF, checking did it match an expected example.

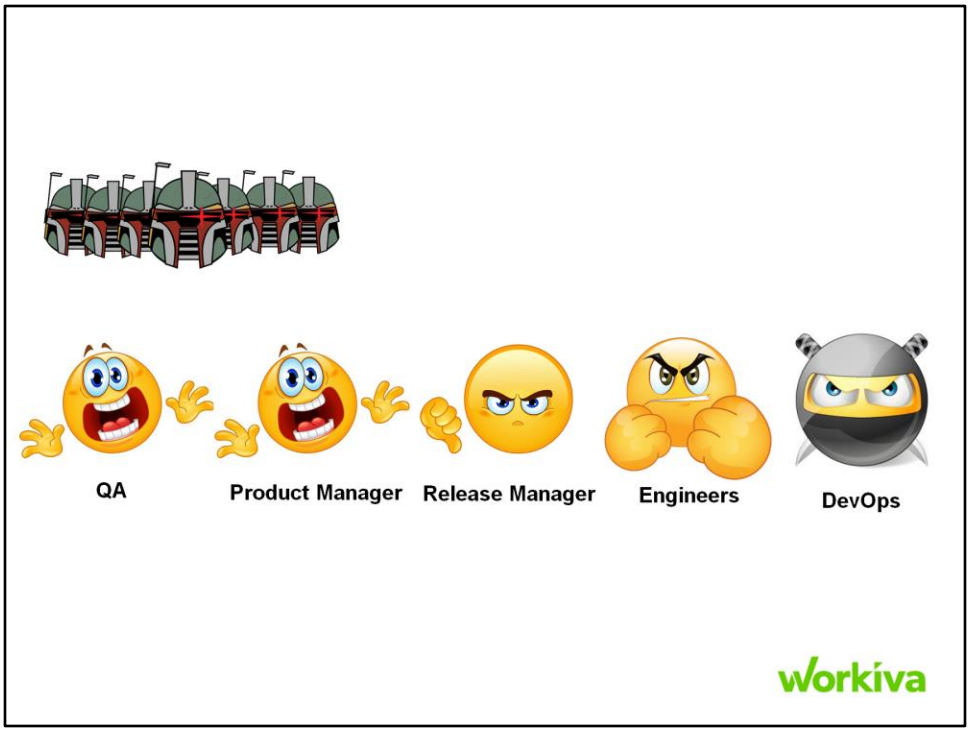
- each of these tests..

logs into our application
imports and edits a document
saves that file
does the translation
compares the output

These tests took around 4 minutes each, when really, the last two steps are all the test needs to do. We made it too easy for users to do this via a functional test, and created massed of duplication, in this case, to fix this (and a model we now promote) is we built a test harness to take just the document translation service and test that service in isolation, shaving time, costs and increasing reliability in tests – these tests now run in isolation as part of our new system (more about that later..).

- Those delays we spoke out..
    - 3 hours for a full test run is a much improved timeframe, but its still way too slow.
    - Too many human interventions in the process that caused delay, not least that one day between build and test..

- culturally, it was not helping the 'QA does Quality' mindset go away, we want this to be replaced by 'Quality Is everyone's responsibility', as it kept that wall of, 'here's your build, go test it..' – in another workflow entirely from where the feature has been built



The happy scene amongst our guys ends up a little like this..

QA is freaking out, for numerous reasons, The product manager, wonders how this is all going to work, especially given the mild mannered QA folk freaking out is never a good sign..

Release Management, well, their natural paranoia says thumbs down to this, how are we every going to have ‘safe’ releases

The engineers go into battle mode, as they assume people freaking out means people don’t trust them, all the while they are pretty excited in the main about owning their own destiny (though not 100% sold on the ‘quality is everyone’s responsibility’ thing..’)

And then the devOps ninjas, who have some nice new shinny tools to build to help empower our product teams to do more.. Faster..

DevOps..

- 'Frictionless development'
  - removal of technical bottlenecks in getting code to customers
  - flexible enough to add in technologies as needed
- Support multiple platforms for build and deployment
  - simple build-test-feedback workflow
  - ownership of a new deployment architecture
  - configurable by teams to get the results they need

Workiva

So lets look again at DevOps teams

Their remit is mostly to ensure everything each product teams needs to get their products out to market is available, build, deployment, monitoring etc.

When product teams choose their own path, if a new language (such as Dart for example) was what the team wanted to work in, then DevOps infrastructure needs to be able to support it, and as we will see, so do testOps..

At this point, what you want is everything starts to seemingly merge into one simple system, the more human intervention we can take out, the more frictionless the lifecycle

Obviously what you don't want in this part, is flaky, slow functional tests that take hours to run sticking up a slick build process, its imperative development teams (and by extension QA and TestOps) test efficiently, testing at a lower level where possible, and minimizing the level of functional tests where possible.

So, if 'need for speed' and 'frictionless development' is the desire, how does that QA testing process, and in essence, TestOps, fit in, given the challenges I've mentioned previously…

TestOps..

- Testing infrastructure focused team
  - Tight Integration with development workflow
  - Integration with DevOps infrastructure
  - Test Frameworks
  - Test Runners
  - Reporting
  - Quality Coaches

workiva

The plan here is engaging a testOps team, fully focused on testing infrastructure and resources,

- The key part of this conversation is Integration with development workflow, and DevOps CI process..
    - TestOps needs to be able to spin up any number of environments for testing, and in essence DevOps should already have some of the tools to do this, for example spinning up docker containers, you would expect a docker image to be available for testers that mirrors production, and also any custom images for testing, this should be tooling from devOps that test ops can leverage
    - Ownership of test frameworks and test runners, TestOps needs to have acute knowledge of what available test frameworks are out there for their teams, be it the various flavors of webdriver, intern.io, or anything else that comes along, and also how that can fit that into our release workflow for teams, and how we best run and report on those tests in a unified manner that looks and feels like the rest of the development process, not an after thought.
    - Reporting, given the types of data companies use have, a lot of clients expect various auditable standards, so TestOps makes their

results available to whatever team needs them, and also pointers to build results etc from devOps, so the full lifecycle can be audited if needed, once again, it needs to look and feel like the same process, the simpler and more uniform the better

- Also coaching, what is the most efficient way to test your service? TestOps should be able to coach and assist on anything in their purview, including how best to test (after all, they have probably learned the hard way how not to test

DevOps - TestOps integration

- 'One Workflow'
  - automate all the things..
  - test all the things
  - seamless integration of tooling
  - rapid commit-testing-feedback loop
- Team-centric - not 'tester'
  - team created specification for test specification
  - moving quality to part of the development and build process

Workiva

So, the meat of the conversation,

- We have devOps teams owning build and deployment architecture,
- We have TestOps teams ready to change the way we test and provide the infrastructure to do it.

- The ideal is to put everything in one workflow, keeping everything in the same CI/CD process if possible, remove any separation and gates in the process, testing should simply be an extension of the development process, not an add on
- Test all the things, but in the right place – if something doesn't need to stand up the whole stack, don't, if something is better suited as an integration test, find a way to do it there. The more reliable tests the better, if functional tests are not reliable, then they are not providing value
- Long running tests where possible should be avoided
- No separation of product and test code, so nothing slows down the feedback loop, no waiting on TestOps to make fixtures, commit the new test at the same time as the product change, getting pass/fail back from testing quicker, means quicker code in the hands of customers

- For us, we also had a few other concerns..

- Remove the focus on 'tester' tooling. It should be simple for anyone interacting

with the code to get quick feedback.

- Test Specifications – the basis of what TestOps is concerned about. Have teams define what they want to test, when, and what they need, and TestOps automatically provides it (more about that in a second..)
- The holy grail – make quality everyone's concern, by providing tooling that can practically move it into part of the development process at whatever part of the test stack is appropriate, meaning the tooling used to test, is the same as used to make the feature, so the two can be done at the same time.

DevOps - TestOps integration

- Workflow integration
  - Build System driven
    - (REST API)
  - Easily configurable Test Specification
    - Skynet.yaml

workiva

Practically, what do these things mean..

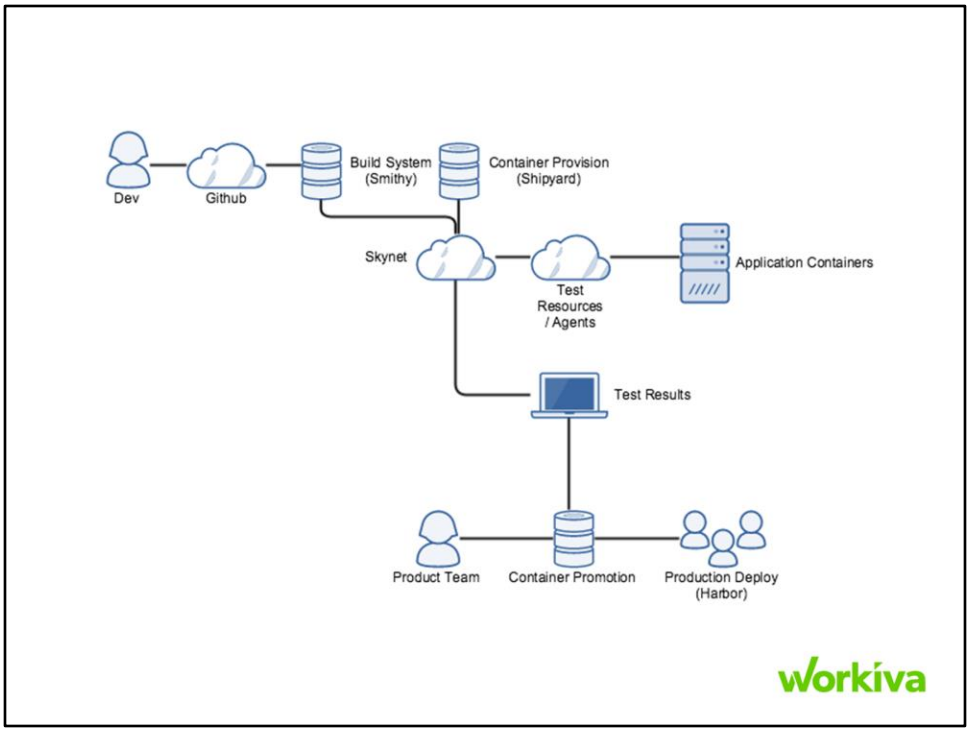
To be successful at turning a number of disparate systems into one smooth workflow, that makes testing seamless to the product team, integration between tooling and infrastructure is the starting point,

- In our case we start at the build system itself, when a commit comes in from a product team and kicks off a build, this is where TestOps want to be ready to kick off whatever testing is defined by the team. To achieve this, we requested access to a status API within our build system that gives us all the builds in the queue and their current status – from Queued, through to success, when a build changes status to 'success' we know a build is complete, and at this point TestOps needs a way to look at what is required for testing.

- The API here is probably the most important link between DevOps and TestOps, We built a 'watcher' application that looks at that API constantly, and notes new builds, their status and where the code lives. Once the Watcher gets notification on what state that build is in, it looks into the repository for that build/branch and gets the testing specification.

- It has to be easily configurable, in our case we use a YAML testing specification that translates what the team wants from its testing, into a machine readable

format that tells Skynet exactly what needs setting up once a build completes



Here is the updated workflow we aim at for continuous delivery using devOps tooling, as well as TestOps resources. You'll notice the addition here of docker containers, something as a company we are moving to for provision of some of our product base, but in reality, it is whatever test bed used, or production hosting mechanism you use.

The idea is, a dev on a product team commits some code, our build system by way of the API lets us know the build is in progress, Skynet peers into that build's repository, looks for a testing specification, and confirms what is needed for testing that product. Once the build has a 'success' value, whatever testing needed commences. When it is done, we send out the test results, comment on the build PR, and if everything is good, that container is deemed as ready for promotion to production.

You'll notice some changes from the earlier diagram, less human interaction, that wait for deployment is removed, as is the manual step of letting release management know everything is in place, all of that is now automated. In theory, if there is no failure, no drop in code coverage, once the dev commits the code, the rest of the process to get through to production is in one place.

To make this work, we had to think about what Skynet would look like in this new world, with fitting in with DevOps CI systems as the main driver

'Skynet 2.0'

- Product/Testing integration
  - Tests live with code
  - Written in the same language as the code
  - Test Platform / Language agnostic
  - Consistent Reporting (Xunit, Rich Test Format...)

workiva

Here we needed to shift our focus to ensuring testOps plans fit within the workflow the development organization was chasing, not a parallel project, but fully integrated with product teams day to day workflow.

- Where possible now we want the tests to live with the product they test's code, and where possible, written in the same language. This makes management easier and again makes testing available just as part of the development workflow, no context switching jumping to other repo's and tools
- we need to be adaptable enough to support any new feature that comes along, regardless of language, in essence what we need to now is – what do you need us to setup, and where do your tests live
- We need to pick up any cross over we can, utilizing the same (or variants of) tools devOps uses to spin up environments, be it Ec2 servers, Docker Containers, or both..
- Again, we get pretty heavily audited, and we also want people to be able to review test results quickly, so we aim to be consistent with our results formats, generally we use xunit based reports)

- All of these items start to move the ownership of Quality into the product team's hands, and work in conjunction with any new tooling and workflow they were deciding on themselves, using devOps infrastructure to bring it to fruition

'Skynet 2.0'

- **Public Specification**
- **Test Specification**
  - each product has a testing specification file - skynet.yaml
  - during the build process:
    - skynet watches the builds api, checks available builds
    - Looks for a presence of skynet.yaml
    - Checks the rules within, and starts tests accordingly

workiva

Prior to starting work on building any of this, we circulated a public specification, first with our engineering architects and devOps, then our engineering department as a whole. This gave us the chance to have a conversation on how teams felt about testing, and really how they envisaged it working, given a new build system, hosting and deploy mechanism.

The testing specification really is the key here, I've mentioned how we watch DevOps API's to know when builds are ready, but we really get most of our information on what we need to support from the testing specifications. we use a simple YAML format file – 'Skynet.yaml' in the root of a product's repository. Another relationship with DevOps here – a yaml file in the same location configures their build properties, so teams are already used to configuring items in this way.

'Skynet 2.0'

- **Product/Testing integration – Skynet.yaml**
  - Testing starts as soon as the build completes
  - Tests can be started on specific events:
    - Each Commit
    - Pull requests
    - Specific file changes
    - Branch Names
  - Additional script configuration indicates
    - Consumer/Contributor tests
    - Resources & Data
    - Environment Variables
  - GitHub PR integration
  - Testing mechanism – skynet.cli

workiva

How we put this together is (I hope) pretty simple, and really allows the teams to get really specific with what they need..

- Within this file we allow teams to specify when, how, and what they want to test, in that they can specify what change or changes should kick of tests, be it every commit, every pull request creation, when specific files are changed, amongst others.
- Specify where the tests are, any test runner commands needed to be executed
- Crucially we can also daisy chain tests together, if one product or service affects another, we can also specify where those tests are and kick those, so we can ensure we don't break anyone dependent on our changes as a consumer of a module or service, for example
- We also include any service templates, essentially any servers, docker images etc. that are needed to be stood up for these tests.
- We built a command line test tool to prove these settings out, enabling teams to test these out prior to committing them to their product repos, so they can ensure they are getting what the expect from testOps at build time.

skynet.yaml

```yaml
requires:
  Workiva/repo: docker
  Workiva/repo2: docker

run:
  on-promotion: true
  on-pull-request: true
  when-branch-name-is: .+integration
  when-modified-file-name-is:
        - auth
        - .+.js
script:
  - test.sh

also-run:
  Workiva/wBinderManager:
    when-requirement-modified-is: w-binder-manager
  Workiva/api-testing:
    on-pull-request: true
  Workiva/xbrlTests:
    when-modified-file-name-is: ^xbrl/.*

test-reports:
  - test-reports

size: medium
timeout: moderate
```

Workiva

This example skynet.yaml file instructs Skynet to;

- Run tests when a pull request is made from a branch with ‘integration’ in its title (or any of the other specified criteria)
- Use the specified Docker containers run the tests, using any commands noted within test.sh
- If the conditions in the also-run are met, also run the consumers of this products tests to check nothing gets broken by these changes when those specific changes are made
- Set a timeout value for test completion, if exceeded this will cause tests to fail

SKYNET Results Editor Management Metrics Status

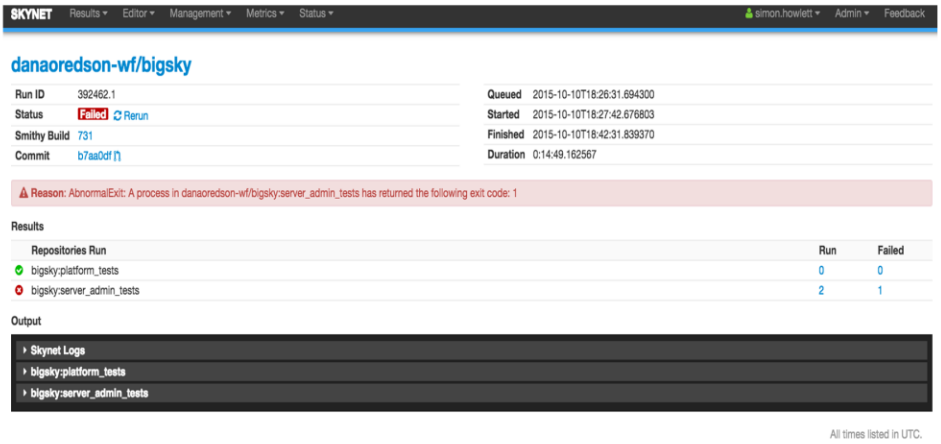
Latest Results

Run ID	Repository	Commit	Queued
2398.1	markerickson-wf/bigsky	6bb4941	2015-10-11T01:28:42.820100
603.1	johnlomen-wf/bigsky	81e6e1c	2015-10-11T00:30:22.434490
485.1	chrislococo-wf/bigsky	cde2243	2015-10-10T20:51:43.380070
731.1	danaoredson-wf/bigsky	b7aa0df	2015-10-10T18:26:31.694300
18594.1	Workiva/bigsky	b7aa0df	2015-10-10T18:26:12.073770
6264.1	Workiva/wTable	f6be57a	2015-10-10T17:14:34.626920
18593.1	Workiva/bigsky	bf08140	2015-10-10T10:06:24.334210
18592.1	Workiva/bigsky	d7d80a5	2015-10-10T09:32:04.129700
18590.1	Workiva/bigsky	1fc7a85	2015-10-10T09:28:31.447480
18589.1	Workiva/bigsky	a2eba4a	2015-10-10T09:26:09.049760
18587.1	Workiva/bigsky	c0e051e	2015-10-10T08:47:45.506420
18586.1	Workiva/bigsky	ebc38e7	2015-10-10T08:46:43.628030
18588.1	Workiva/bigsky	c2fece5	2015-10-10T08:46:12.945970
18584.2	Workiva/bigsky	bf08140	2015-10-10T03:48:18.384090
1066.1	andrewalbers-wf/bigsky	cbebc63	2015-10-10T02:15:40.914280
1054.1	Workiva/skaardb	c7f58c4	2015-10-10T00:24:23.282390

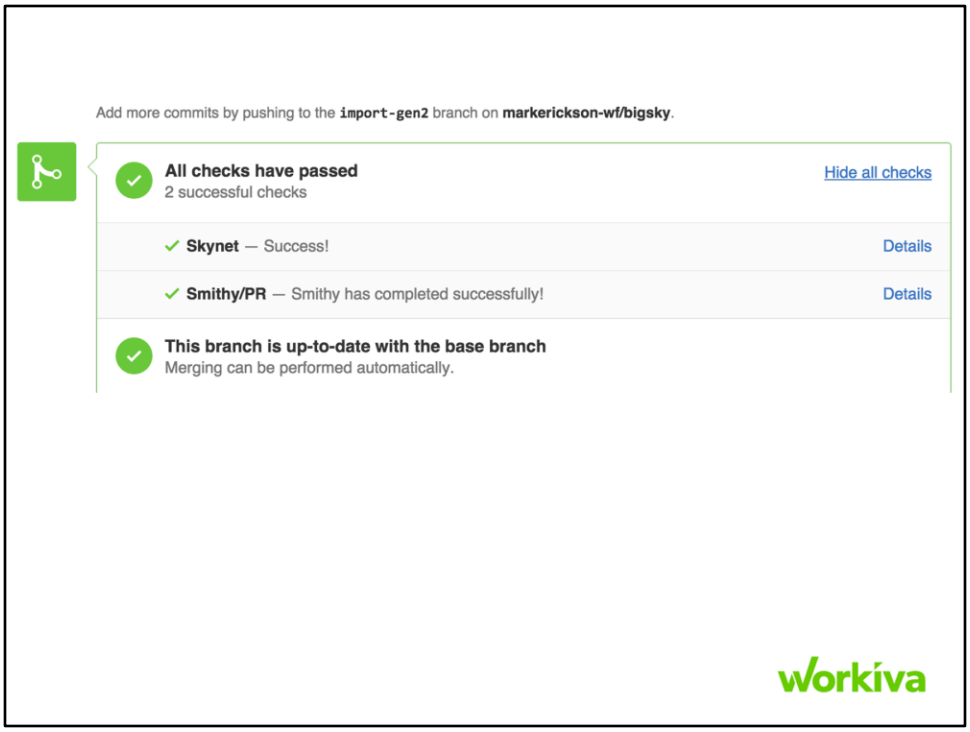
workiva

This is how we report the results, intentionally modeled on the way devOps build system reports results also..

Each build, listed by time of completion, pass/fail with a link to the repo, and the specific commit the tests ran on

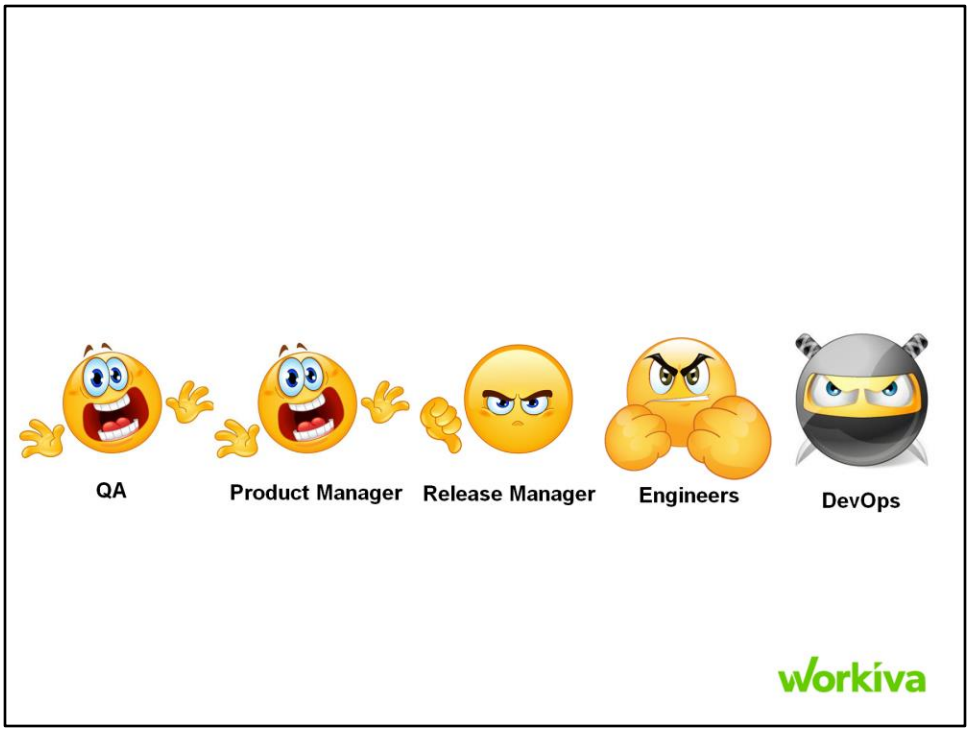


Again, with the same appearance as DevOps build results, here you can see the output of a set of integration tests for one of our platforms, they have one failed test they need to resolve. In this case it's a team that has elected to run phantom.js tests to test a component, and one of them is failing on a timeout.

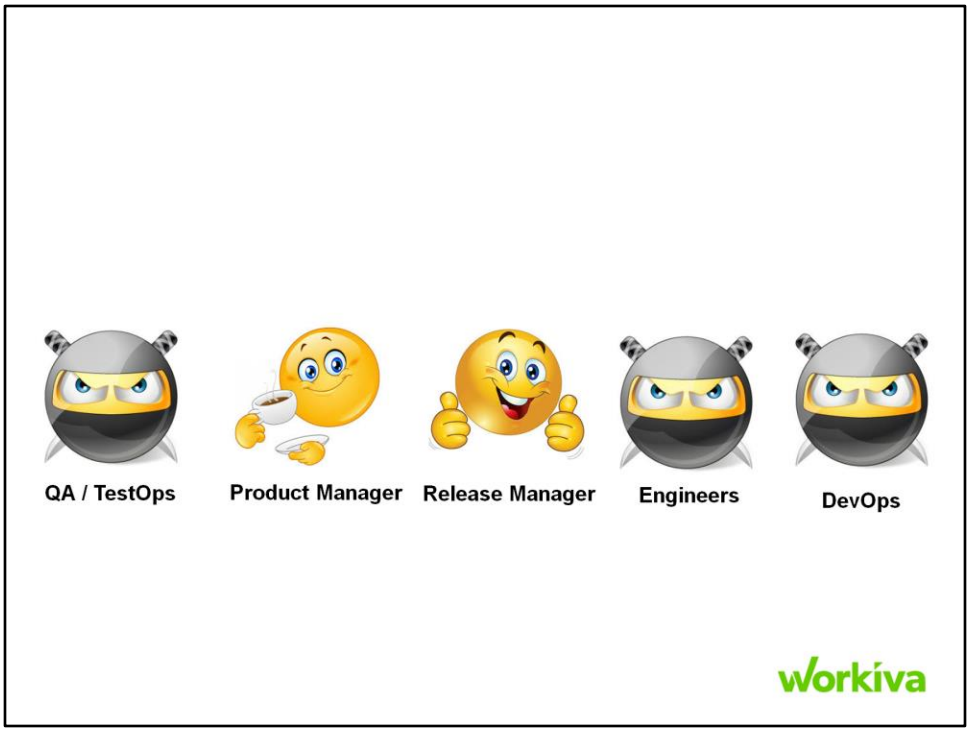


Had this passed, the github PR would have seen a comment from Skynet, much like this one. with a failing test it would see a failure report and a link back to the test results.

All of this, as you hopefully can see, makes the development lifecycle and CI process seem a little more together than the traditional get something build and throw it at QA to test mentality, automating pretty much every step of the process from commit onward, through testing to reporting, and making sure it seems like one smooth system helps everyone work together.



Remember these guys.. At this point, its more like this..



We think this work really helps everyone,

For the QA person, we took the separation of development and testing and used these changes to empower them to be quality coaches, and not a perceived bottleneck at the end of the process, with testing becoming seamless to the whole team, they can get more involved in the whole development process, and also help encourage that accountability for quality and the user in the whole process. They also get to write some code, which most of them seem to enjoy.

The Product Manager is happy as their team can configure their own testing requirements, they don't have to wait on anyone else or queue for test resources, so everything speeds up for them.

The Release manager has better insight into the state of testing, by the time word gets to release management that team x is going to release something, everything should already be in place.

Engineers, fully bought into this process, and also see less delay getting their code out of the door (and less bugs…)

DevOps, well, they are happy because we are getting extra work from their tooling, ensuring that what goes through their platform gets treated correctly.

Success?

- No delay between build – test
- No queuing
- Quicker items to production
- Less instance hours (quicker test runs, less $$)
- Reduced costs
- Cultural change?

workiva

So, the benefits we have seen,

- Teams now want to own their testing. We have 15 teams on Skynet 2.0 now, we spin up a variety of platforms and test frameworks, so far no project has been the same as another, so we've learned pretty quickly how to bend our framework to support anything new products need
- Those teams see the build/test/feedback process as one process, even though it spans 3 teams, and 3 levels of infrastructure.
- That 1 day average delay from a build completing to it getting into testing, that's gone.
- Those teams that would previously had to queue for Skynet test resources, when they move to 2.0 – no more waiting, once their builds are complete testing is off and running
- We've reduced the average time a Pull request into production is open, from 1 week, to 1 business day.
- Moving testing from functional tests within the old framework where possible to integration tests is reducing the amount of instance hours required per test run.

- One last thing on this, part of the reason for doing things this way is working toward an environment where quality is just a given, and everyone is accountable to themselves and the customer in terms of the standards they keep up for their products. We have been very intentional about making tools for everyone, that fit

in with the CI tools devOps provide for the organization, so that we end up with one mindset on what the expectations for teams in terms of quality are, and in that process we remove the 'them and us' separation of development and QA, keeping everything in one workflow helps that immeasurably.

The idea extends to allowing for more innovation, less bugs to deal with equals more time to be creative and come up with killer new features. Hopefully by sharing how we ensure that devOps and testOps work together, you can do the same.

Questions?

workiva