DevSecOps as an Agile Force Multiplier

Nate Berent-Spillson

VP, Engineering

Nexient

About Nate

Nate has over 25 years' experience as a full-stack architect, developer and delivery leader

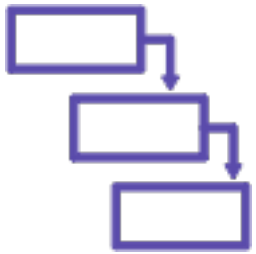
He is a frequent author and speaker on Agile, DevOps and enterprise technology development

BEST BUY · GAS SOUTH · Ford · AutoZone · THOMSON REUTERS

DEVOPS digest · TechBeacon · InformationWeek · InfoWorld

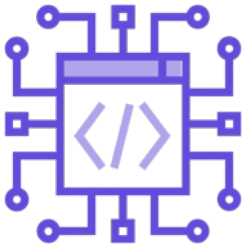
O'REILLY · THE ENTERPRISERS PROJECT · SD Times · AB AGILE & BEYOND · TechRepublic

The rise of agile process



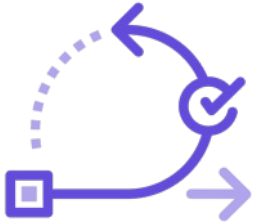
Big upfront design

- simply doesn’t work
- only suitable when the cost of change is really high



For software

- cost of change too great
- feedback too long
- batch size is too big



Agile initiatives

- helped us move to smaller bites on smaller timescales

Agile process is better

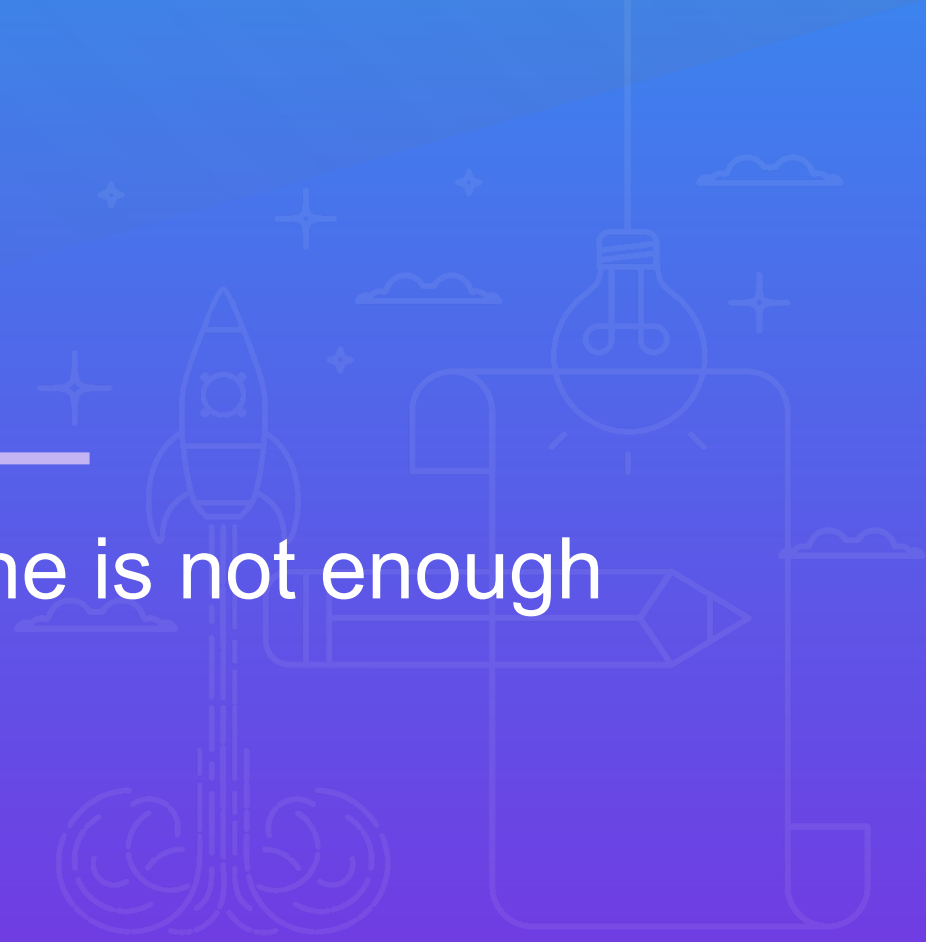
(Yea, waterfall is that bad)

tighter business collaboration

smaller units of development

better feedback

Agile process alone is not enough



The goal is business value

Agile process alone is not enough to:

- Features and products into the hands of customers faster
- Be able to respond quickly to changing market conditions
- Deliver business value
  - efficiently
  - safely
  - securely
  - consistently

Nexient

DevSecOps unlocks feature flow

Every software feature is ultimately intended to make money or save money

But this value can't be released until that feature is in production.

Until then, your investment is locked up

DevSecOps exists to eliminate blockers that delay this time-to-value

Nexient

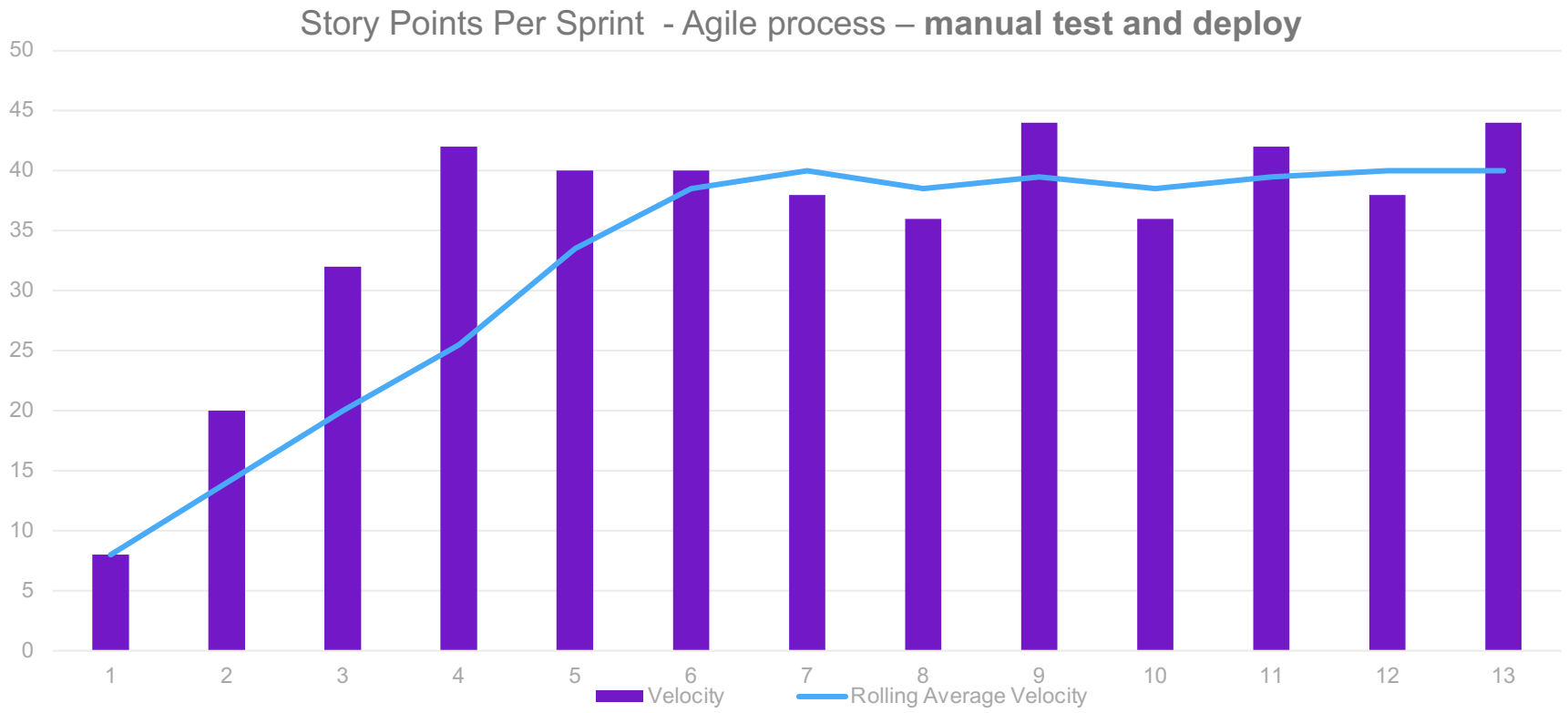
Agile in a typical enterprise

DROP IT INTO DEV AND LEAVE THE REST UNTOUCHED

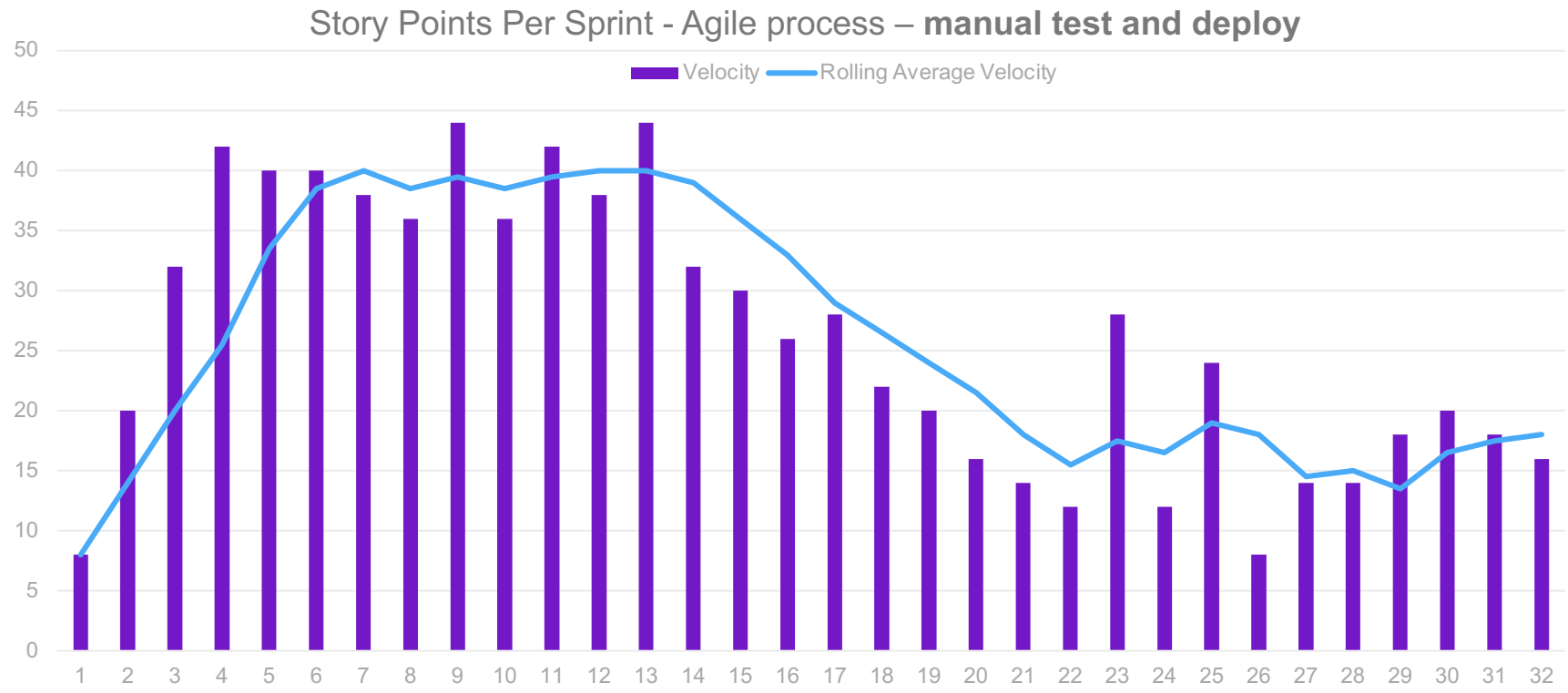
- Existing teams ‘siloed’ norms
- QA remains separate and manual
- Infrastructure and Operations remain the same
- Shared services & “ticket purgatory”
- PMO requires the same ‘metrics’ as waterfall
- Finance - “big-bang”, massive waterfall style initiatives

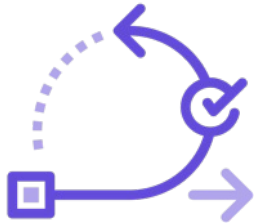
Great at first

Getting things “done”



Gradual velocity decline



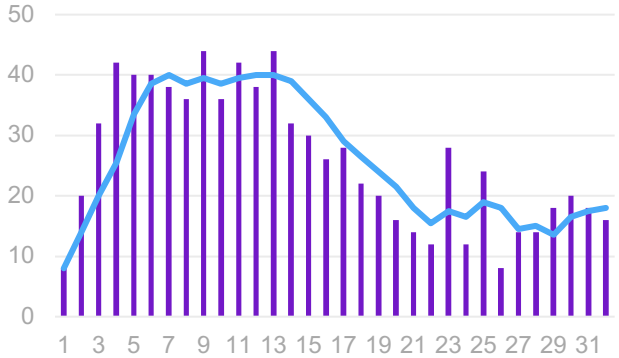


BUT...

We're doing all the agile things!

Friction in our process

MANUAL PROCESS QUICKLY BECOME BOTTLENECKS

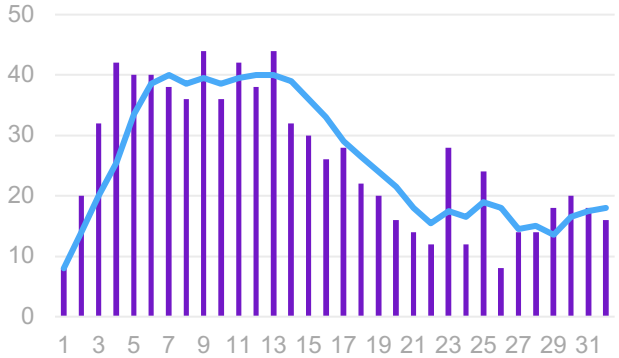


1. Gradual decrease from manual testing as size of regression suite grows

Nexient

Friction in our process

MANUAL PROCESS QUICKLY BECOME BOTTLENECKS

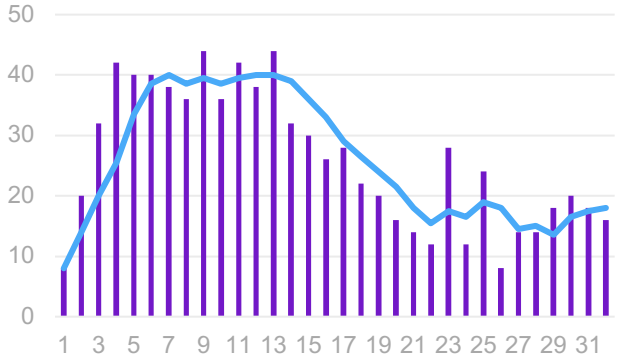


1. Gradual decrease from manual testing as size of regression suite grows
2. Mini-waterfalls – QA under utilized at beginning of sprint, immense pressure at the end of sprint

Nexient

Friction in our process

MANUAL PROCESS QUICKLY BECOME BOTTLENECKS

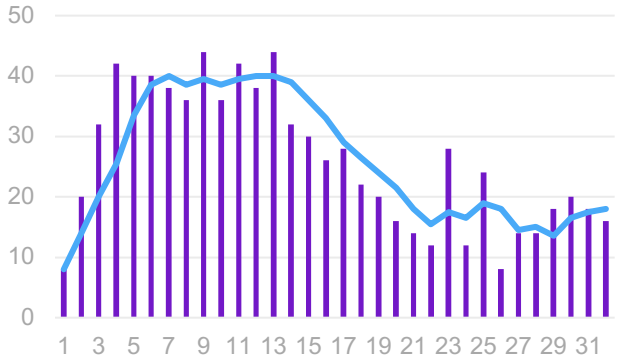


1. Gradual decrease from manual testing as size of regression suite grows
2. Mini-waterfalls – QA under utilized at beginning of sprint, immense pressure at the end of sprint
3. Manual build and deploy requires constant context switching

Nexient

Friction in our process

MANUAL PROCESS QUICKLY BECOME BOTTLENECKS



1. Gradual decrease from manual testing as size of regression suite grows
2. Mini-waterfalls – QA under utilized at beginning of sprint, immense pressure at the end of sprint
3. Manual build and deploy requires constant context switching
4. Piling up work for deployment
    1. Merge requests and deploy to QA
    2. Ops can’t keep up with deploys to PROD

Nexient

What are we missing?

Let's go way back for a minute to 2001

Agile manifesto

BACK TO BASICS

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Businesspeople and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile manifesto

BACK TO BASICS

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.**
4. Businesspeople and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. **Working software is the primary measure of progress.**
8. **Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.**
9. **Continuous attention to technical excellence and good design enhances agility.**
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Nexient

Agile manifesto

BACK TO BASICS

7. Working software is the primary measure of progress.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
8 Agile processes promote sustainable development. […] maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.

Nexient

Let's re-state those tenets

Primary measure of progress

delivery of working software

at a constant pace

designed and built with technical excellence

Agile and DevOps are really just Lean

Systems thinking

- Software value stream
- Eliminate waste
- Reduce unplanned work

Feedback amplification

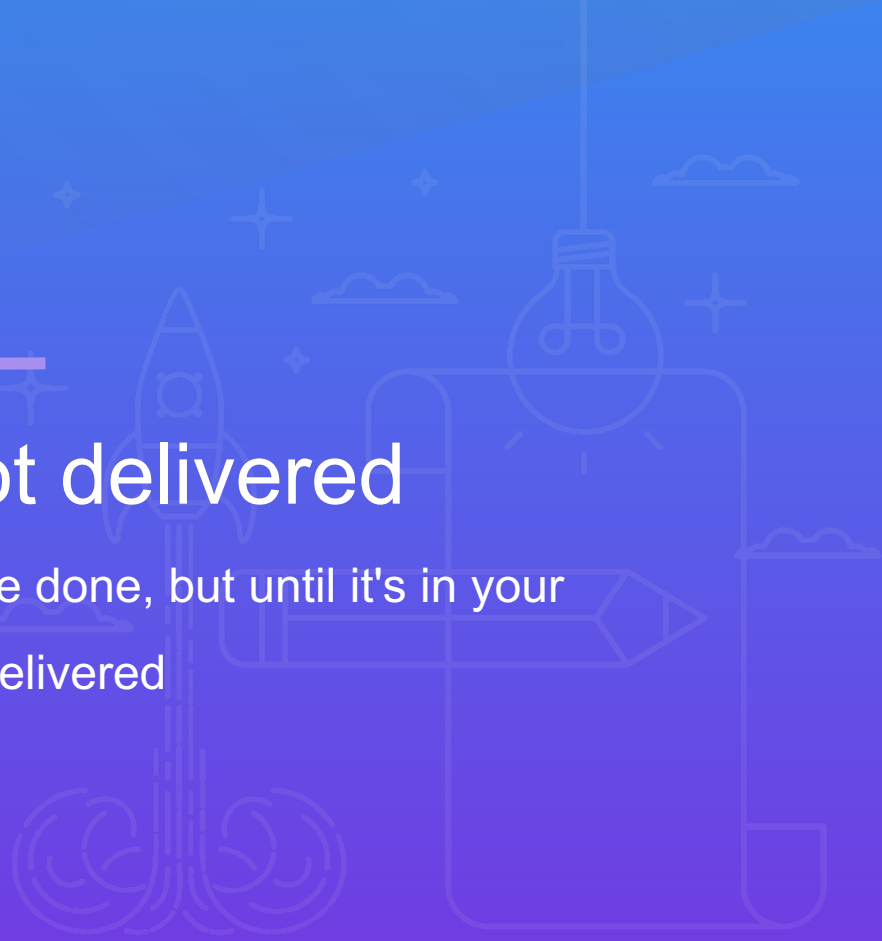
- Tight feedback
- Short timescales
- Technical **and** business

Continuous improvement

- Good can always be better
- Constant learning
- Optimize at constraint

"Done", but not delivered

A pizza that's been baked might be done, but until it's in your hands it's not delivered



Done vs delivered

IS IT DONE, DONE, DONE?!?

- Delivered is released to production

Nexient

Done vs delivered

IS IT DONE, DONE, DONE?!?

- Delivered is released to production
- Add released to production to our Definition of Done
  - really tough for most organizations
  - would get management attention ;)

Nexient

Done vs delivered

IS IT DONE, DONE, DONE?!?

- Delivered is released to production
- Add released to production to our Definition of Done
  - really tough for most organizations
  - would get management attention ;)
- Middle ground – deployed to stage, deploy**-able** to prod

Done vs delivered

IS IT DONE, DONE, DONE?!?

- Delivered is released to production
- Add released to production to our Definition of Done
  - really tough for most organizations
  - would get management attention ;)
- Middle ground – deployed to stage, deploy**-able** to prod
- Work more into your Definition of Done over time

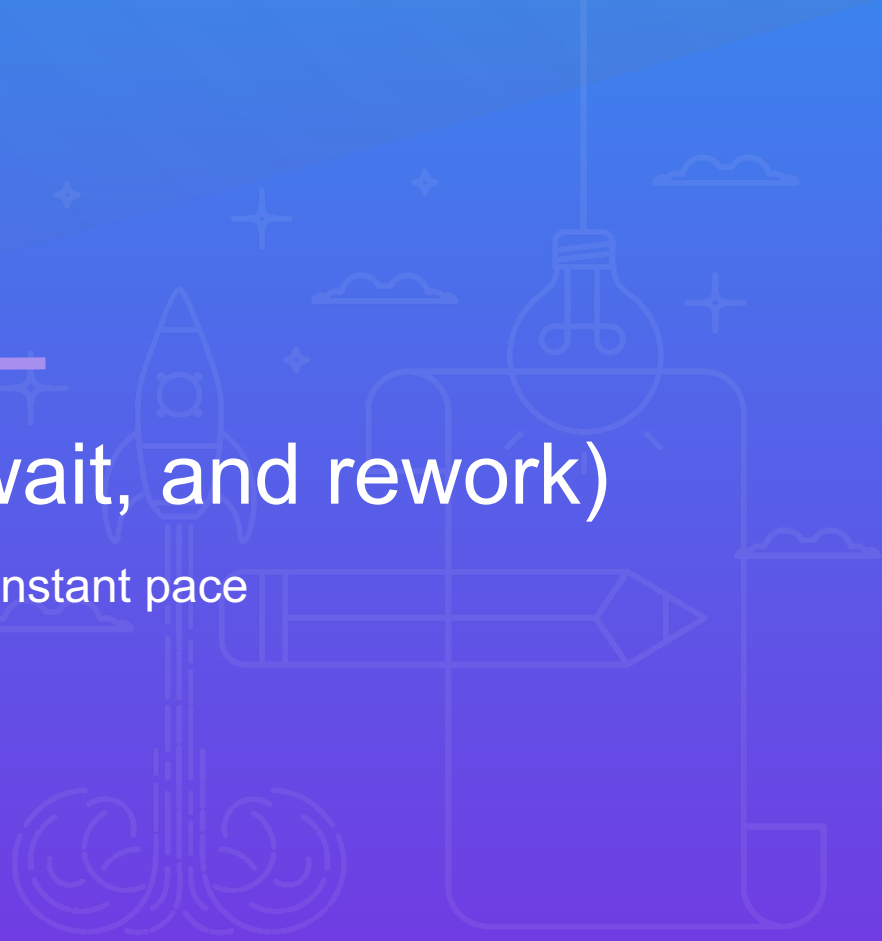
Done vs delivered

IS IT DONE, DONE, DONE?!?

- Delivered is deployed to production
- Add deployed to production to our Definition of Done
  - really tough for most organizations
  - would get management attention ;)
- Middle ground – deployed to stage, deploy**-able** to prod
- Work more into your Definition of Done over time
- Decouple deployment and release with feature flags

Identify the **waste** (wait, and rework)

The drag on our constant pace



Manual process drag

WHERE DO WE HAVE INCREASING FRICTION

- **Work In Progress (WIP)**
  - **Silent killer**
  - **Waiting to be tested or deployed**
- **Manual Testing**
  - increasing feedback loop
  - manual regression takes longer
  - constant rework to test cases
- **Operations**
  - hand-off for upper environments
  - using manual process
  - dev teams can't get access to anything (even logs) in prod
- **Manual** == wait AND rework

Anti-patterns creep in

PRESSURE CAN DRIVE UP REWORK

- **Manual testing and mini-waterfalls**
  - pressure to accept stories at end of sprint but create a defect
  - stop testing within the sprint and test a sprint after
  - Devs busy at beginning of sprint but QA waiting for stuff to test
- **Defects steal velocity**
  - don’t point defects
  - they incur overhead

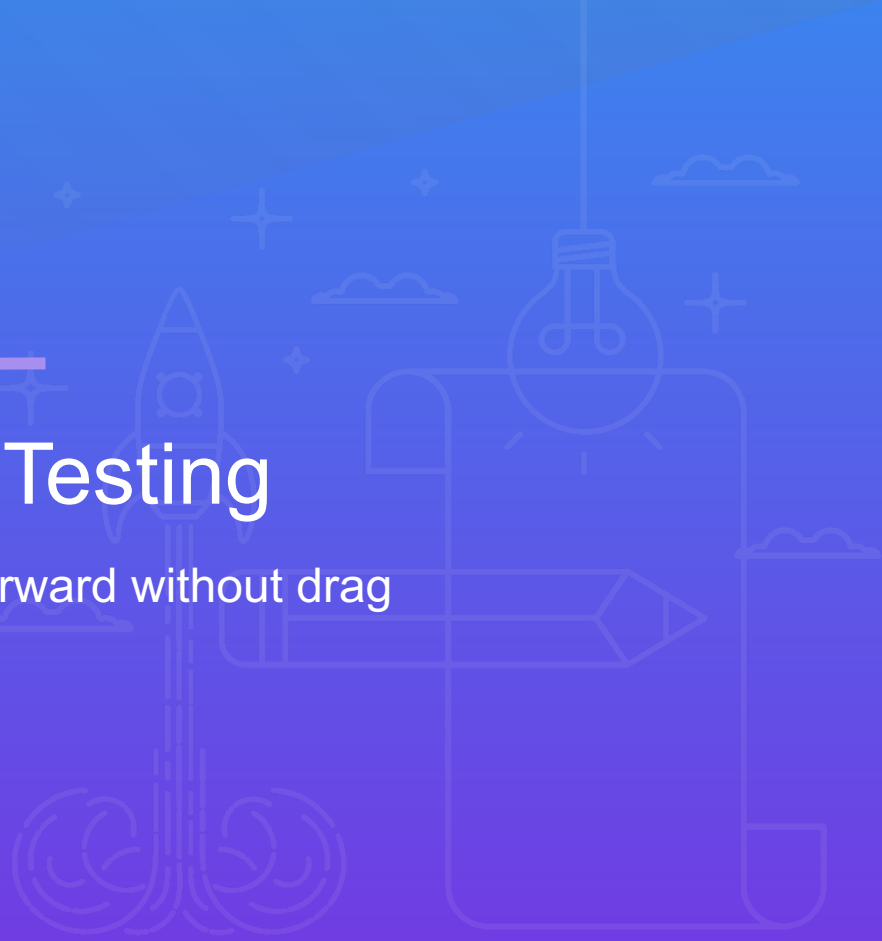
Prevent rework and limit WIP

DON'T PUT OFF WORK FOR LATER

- Strict WIP limits
- Test within the same sprint
- Only accept defect free stories
- Create failing test **first** for every defect found

Automated Testing

The **key** to progressing forward without drag



Automated testing

LEVEL SET

- **Automated tests** –test that could be run automatically (in a CI process)
- **Test Driven Development (TDD)** – write the test before the code
- **MANUAL** testing is not **TDD** even if you write the **manual** test first!
- You will **NEVER** go back and write the tests later
- **TDD** and automated testing does not slow you down
- Automating **mouse clicks and keypresses** is still slow and brittle

Automated testing

WRITE BETTER CODE

- Testability helps you write **BETTER** code from the start
- SOLID principles are more easily tested
- Self documenting
- Tightens up your feedback loops
- Speeds up dev testing
- Prevents regressions

Automated testing

IT'S NOT EASY AT FIRST

- **Tough** for management to commit – treat it like a feature
- **Tough** for developers and QA to adopt – learning curve

- **Absolute MUST** – proven benefits
- It's the **safety net** – stop 'hoping' nothing broke
- **Manual testing drag is self-inflicted!**

Manual build and deployment

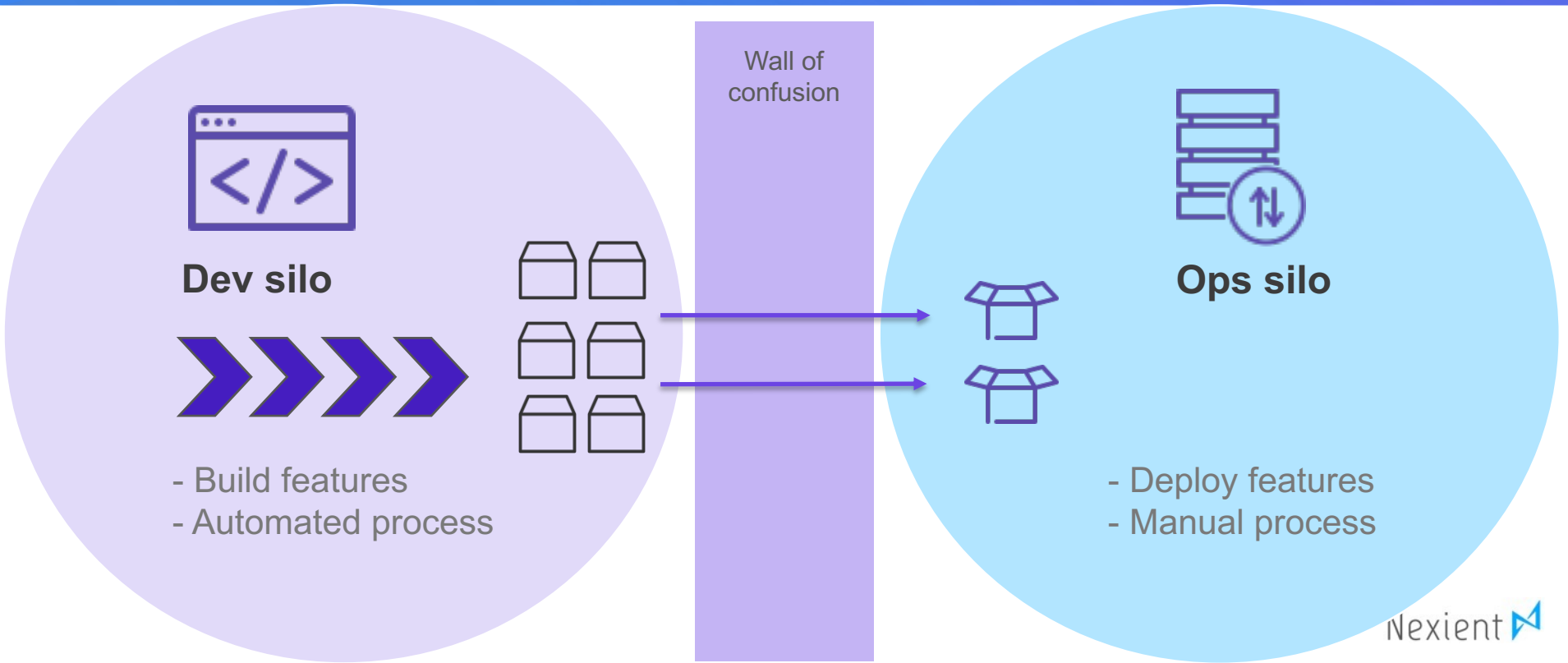
Operations process bottleneck

AGILE DEV DELIVERS FEATURES TOO QUICKLY

- **Manual** operators, performing **manual** steps, defined in a **manual**
- Higher risk of human error, less auditable, less secure
- Requires accurate, complete and up to date documentation
- Long **wait** times between individual teams / steps
- **Manual** review, change, and approval processes

Dev and Ops siloes

THE WALL OF CONFUSION



Nexient

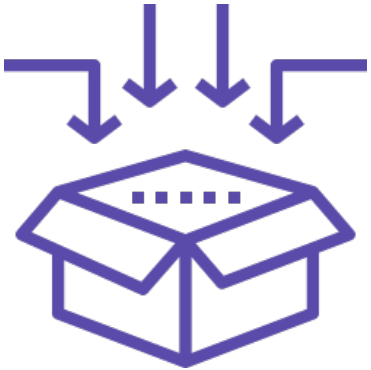
Conflicting metrics

RE-ALIGN MEASURES FOR DEV AND OPS

- **Development**
  - rewarded for throughput
  - push more up against the “wall of confusion”
- **Operations**
  - rewarded for stability
  - prevent change to a stable environment
- **Process mismatch**
  - Flow is interrupted with manual hand-offs and change of process
  - Use metrics that measure outcomes globally

DevSecOps

TEAR-DOWN THE WALL OF CONFUSION



- Enable safe, frictionless flow throughout
- Ops focuses on providing a stable platform, and pipelines to deploy to it
- Dev focuses on building stable, secure, highly automated features
- Security is baked in throughout the process
- Establish and maintain flow

Nexient

Dev and Ops siloes

THE WALL OF CONFUSION

Build and deploy features using fully automated process

Nexient

Lean metrics

MEASURE FOR OUTCOMES

- **Business Outcomes**
  - Design the metrics into the features
  - Automated, objective measures
  - Adjust current metrics accordingly
- **Feature lead time**
  - from first line of code
  - to deployed in production
- **Deployment frequency**
  - Smaller
  - More frequent
  - Less risk and variability
- **Mean time to recover**
  - how quickly can we restore service
  - fix forward, rollback, or switch-over

Two enterprises

ONLY DIFFERENCE IS ONE TORE DOWN THE WALL

Separate Dev and Ops

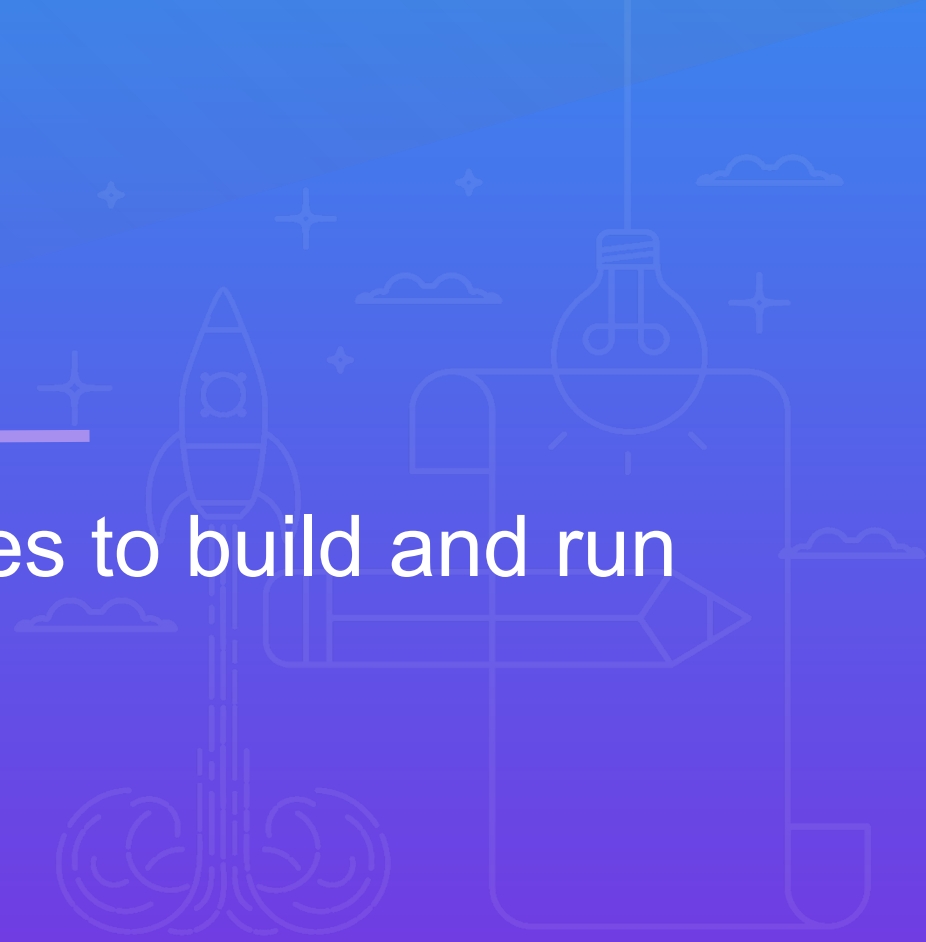
- Feature lead time
  - 6 weeks – **12 months**
- Deployment frequency
  - 4 – 6 weeks
- Mean time to recover
  - 2 – 8 hours to roll back
  - 2 – 4 weeks patch

Unified DevOps

- Feature lead time
  - **2 – 4 weeks**
- Deployment frequency
  - 1 – 2 weeks to stage
  - 2 – 6 weeks to prod
- Mean time to recover
  - **60 seconds (blue/green)**

Nexient

DevSecOps pipelines to build and run



Continuous Integration (CI)

SOFTWARE THAT BUILDS SOFTWARE

- **Replace** manual merge, build, deploy, test with software
- Continuously **integrate** source code with every check-in
- **Verify** working correctly with automated tests
- **Scan** for **security** vulnerabilities automatically
- Make sure it all **works together**
- **Bundle into deploy-anywhere business feature (“deployable artifact”)**

Continuous Delivery (CD)

SOFTWARE TO DEPLOY THE SOFTWARE

Software that can automatically

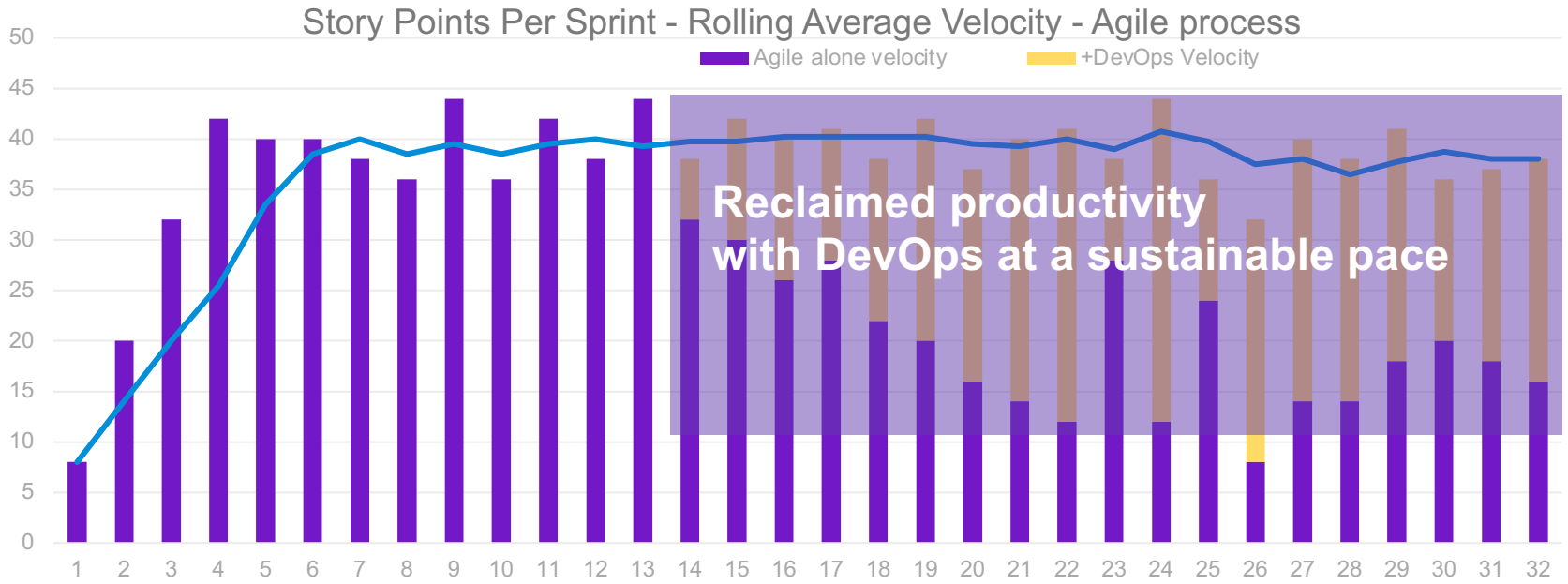
- Take the deployable artifact (from previous step)
- Combine with configuration, access keys, and secure passwords (”secrets”)
- Place them in an environment (Dev, QA/Test, Stage, Prod)
- Deployed business feature
- Test and verify the security of the exposed surface area

Nexient

Achieve a sustainable pace

Combining for maximum benefit

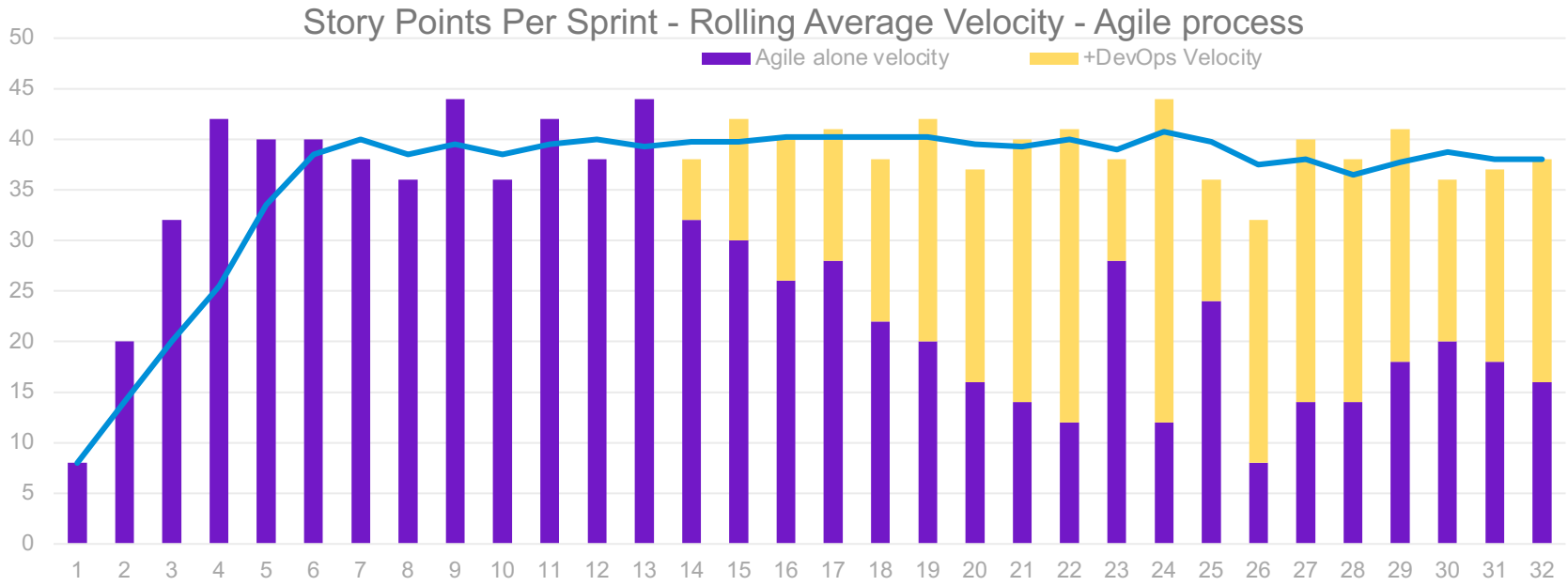
AGILE + AUTOMATED TESTING + DEVOPS



Nexient

Combining for maximum benefit

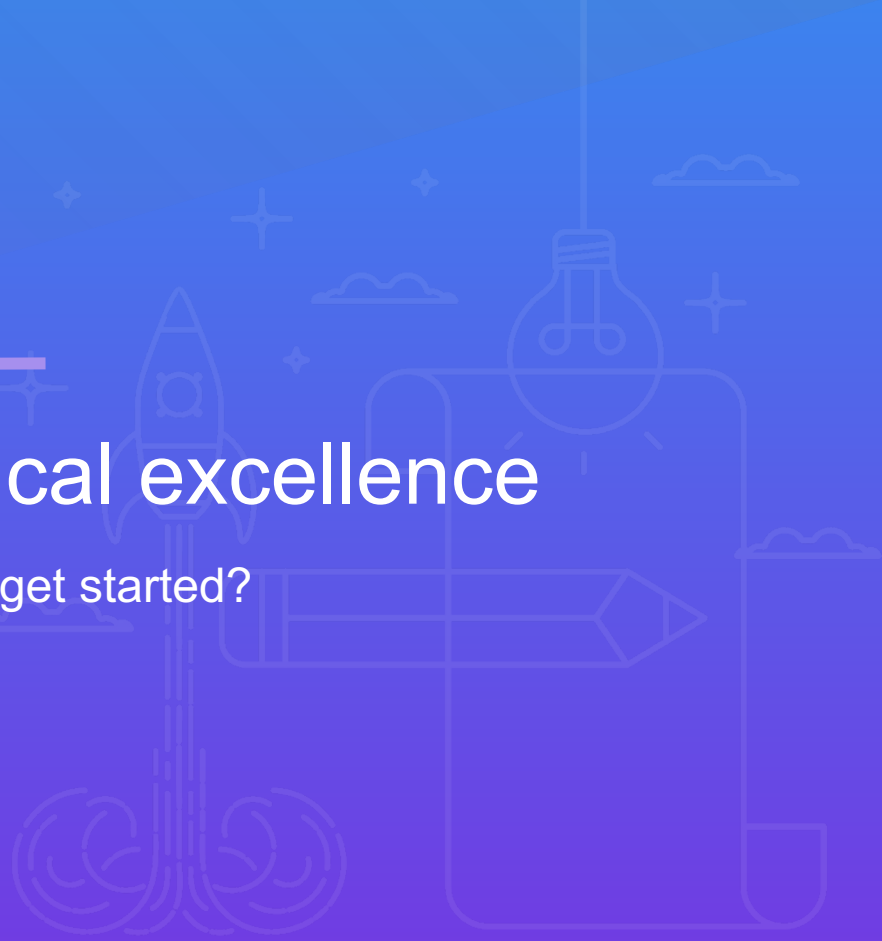
AGILE + AUTOMATED TESTING + DEVOPS



Nexient

Improve our technical excellence

Sure, but how do I get started?



Continuous improvement

ALWAYS BECOMING BETTER

- Commit to automated testing
- Establish DevOps pipelines
- Work toward released to PROD in your DoD
- Measure metrics that improve outcomes
- Continuous learning (personal and organizational)

Primary measure of progress

delivery of working software

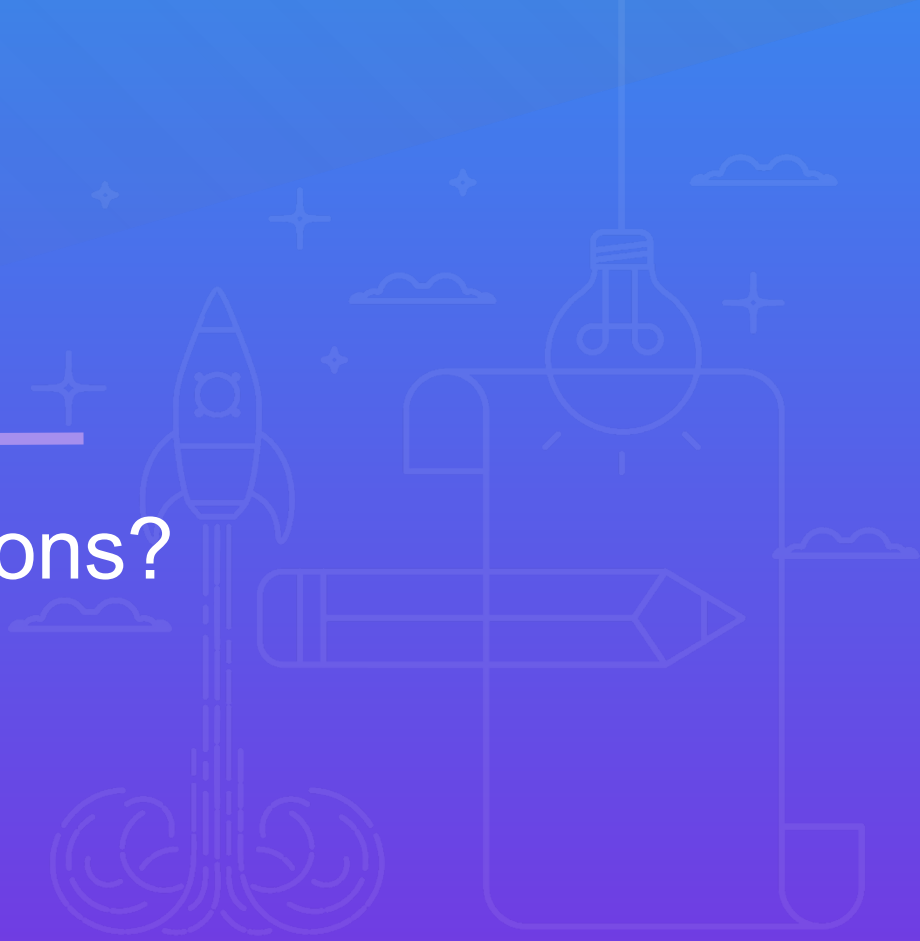
at a constant pace

designed and built with technical excellence

Key Takeaways

- Move beyond just agile process
- Apply automation and DevOps
- Systems thinking
- Continuous learning and improvement
- Measure business outcomes

Questions?



Thank you

Nate Berent-Spillson

natespillson

We’re hiring!
Nexient.com/careers

Nexient

References

Accelerate – Forsgren, Humble, Kim

DevOps Handbook – Kim, Debois

Continuous Delivery – Humble, Farley

Clean Code, Clean Coder, Clean Agile – Robert Martin