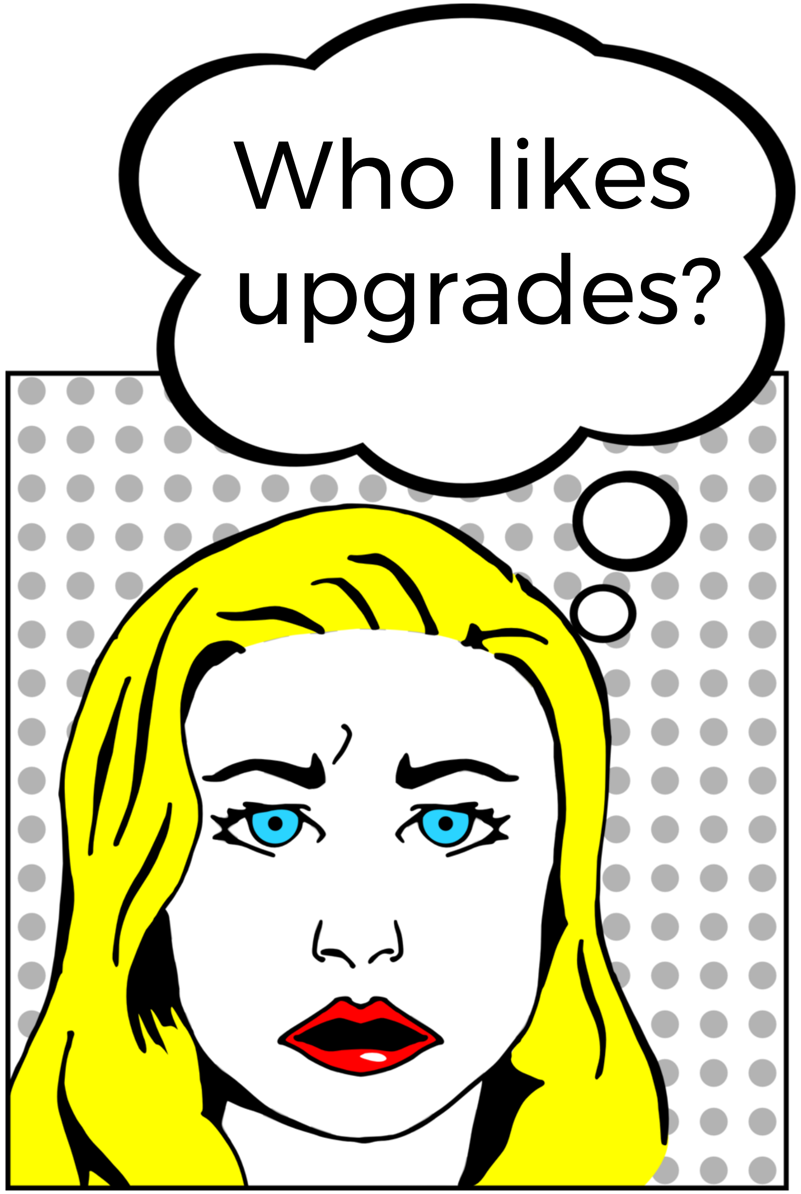
Duck Creek Technologies

Continuous Updates

September 2019

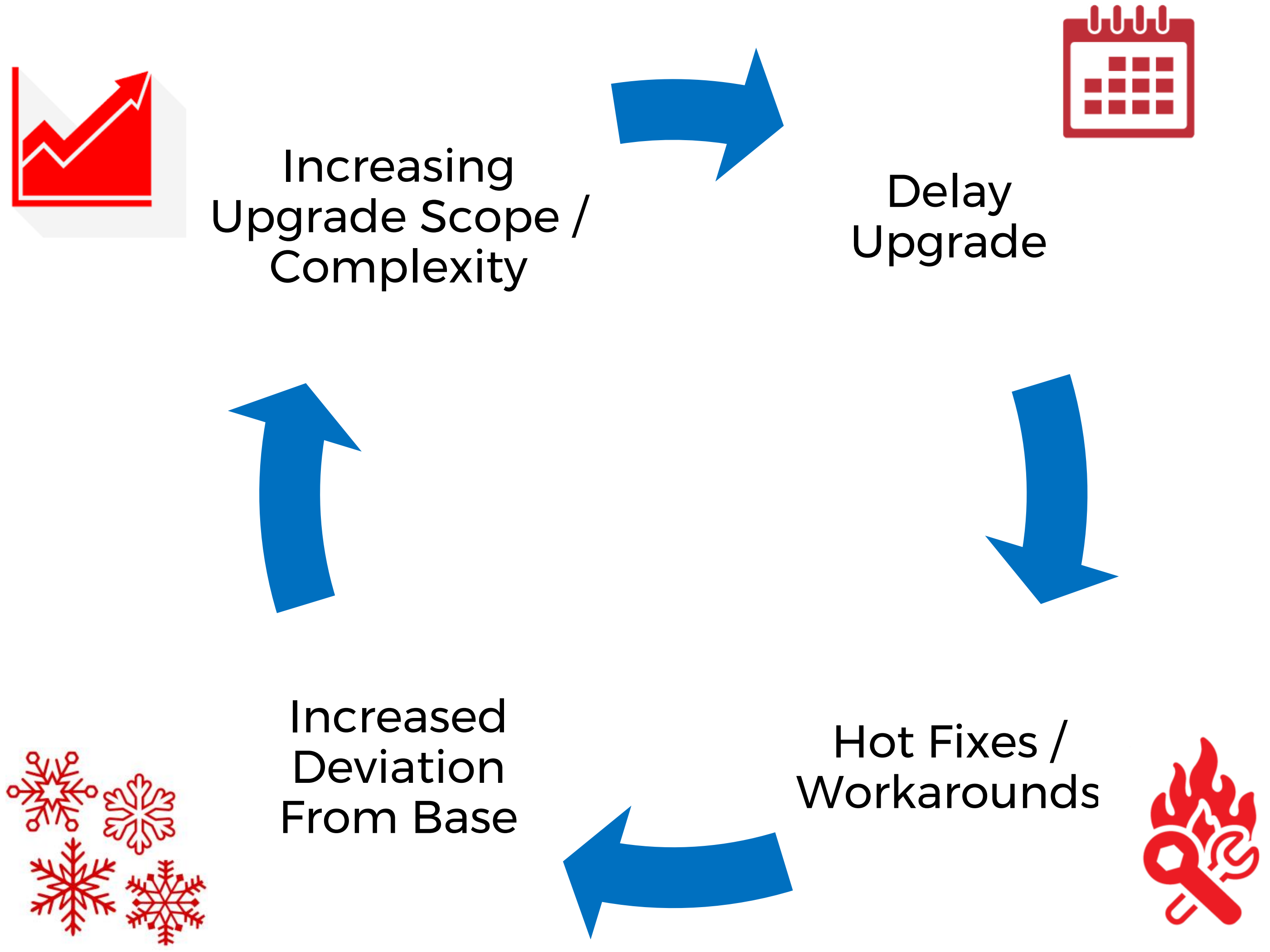
Traditional Upgrade Challenges


Who likes upgrades?

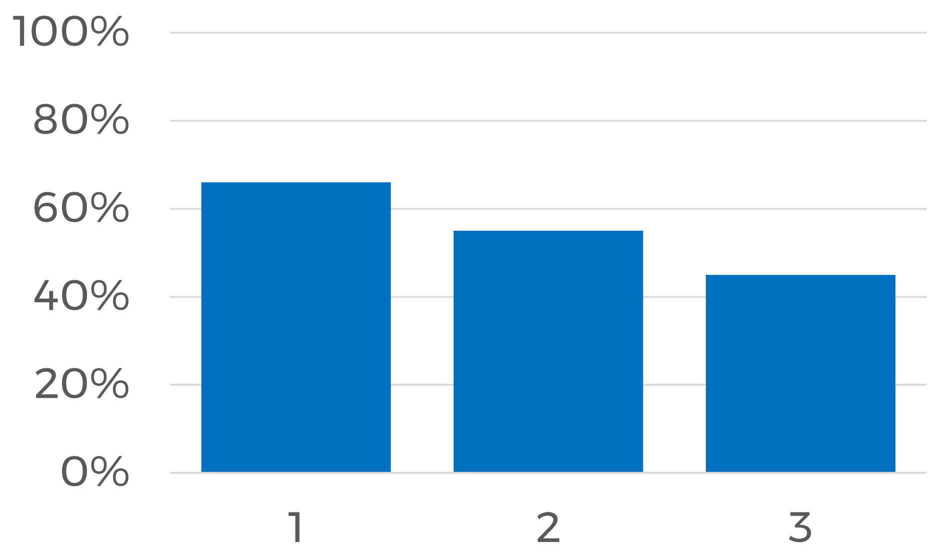
Traditional Upgrade Challenges

Problems	Root Causes
Build Conflicts	• Customizations link directly against Duck Creek libraries • Hot fixes are different for every client
Lengthy, error-prone deployment process	• Every deployment is different • Manual Processes
Unpredictable Risk of Failed Scenarios	• Cannot test all unknown client scenarios] • Large Scale Change • Manual Processes
Large Re-Training for Business Users	• Large scale changes between upgrades • “Big Bang” deployment

Vicious Upgrade Cycle


Increasing Upgrade Scope / Complexity
Delay Upgrade
Hot Fixes / Workarounds
Increased Deviation From Base

Insurers are slow to upgrade



- 66% of vendors reported that at least some customers were behind the current release.
- 55% of vendors reported that at least some customers were behind by up to 3 years
- 45% of vendors reported that at least some customers were behind by more than 3 years.

- 50+ Core System Vendors surveyed

Continuous Updates

Continuous Updates

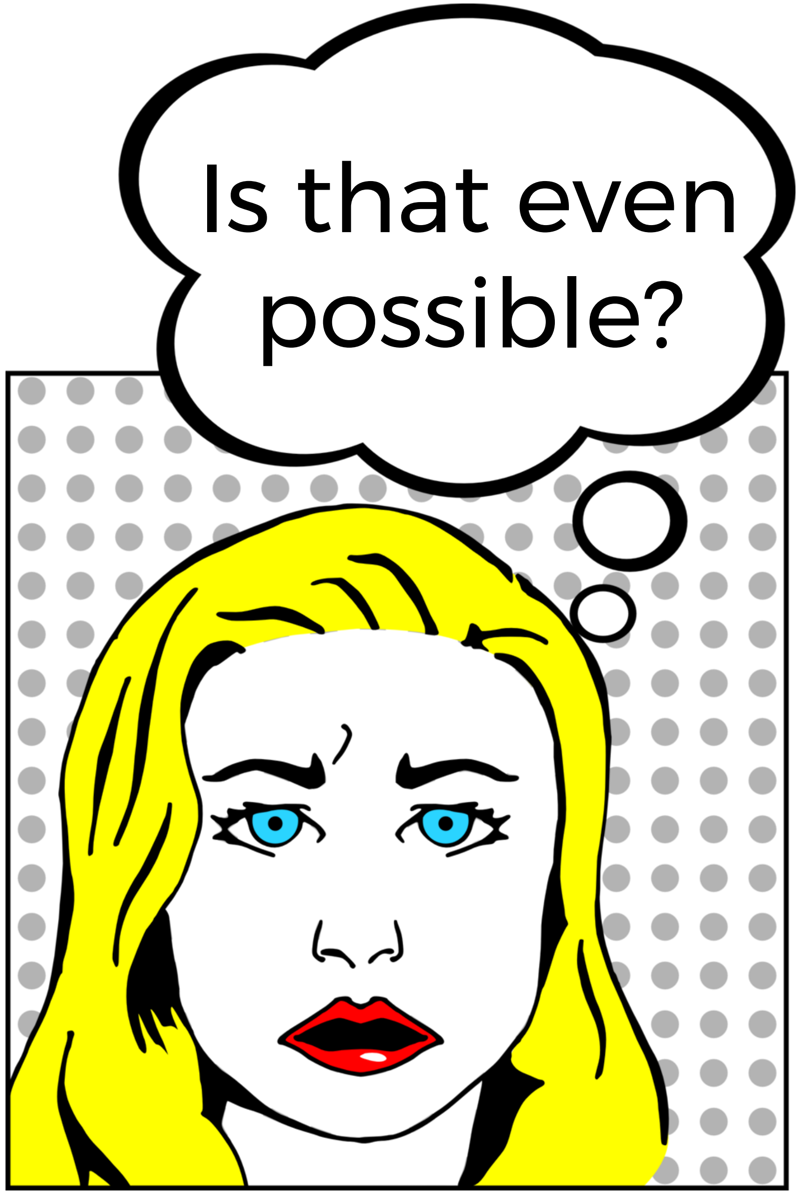
From:

- Infrequent (~ year or more) upgrades
- Expensive / Time consuming to resolve build conflicts
- Long Downtime
- Extensive User Re-Traiing
- Non-standard Hot Fix for critical defects

To:

- Frequent (~2 week) updates
- Zero-touch updates
- Zero-downtime updates
- Zero-user impact updates
- No Hot Fixes (rather, roll into 2-week upgrade cycle)

- Benefits
  - Faster access to new features / innovation / fixes
  - Eliminate costly / time consuming upgrade process
  - Improved support – all clients on same version / eliminate hot fixes

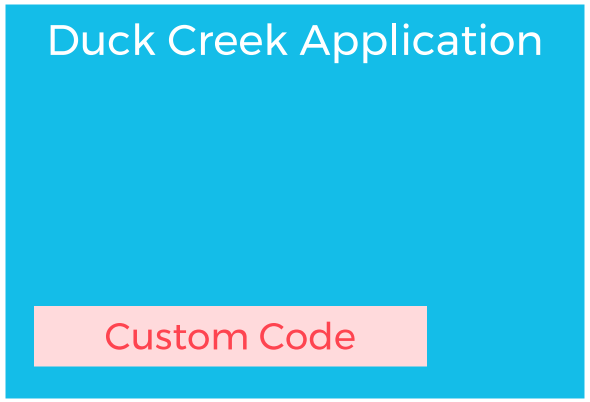

Is that even possible?

Continuous Update Solution

Problems	Solutions
Build Conflicts	• Clean separation of custom code from DC Code
Lengthy, error-prone deployment process	• Automated DevOps Pipeline
Unpredictable Risk of Failed Scenarios	• Automated regression testing • Small scale change
Large Re-Training for Business Users	• Feature Flags separate functionality change from code deployment

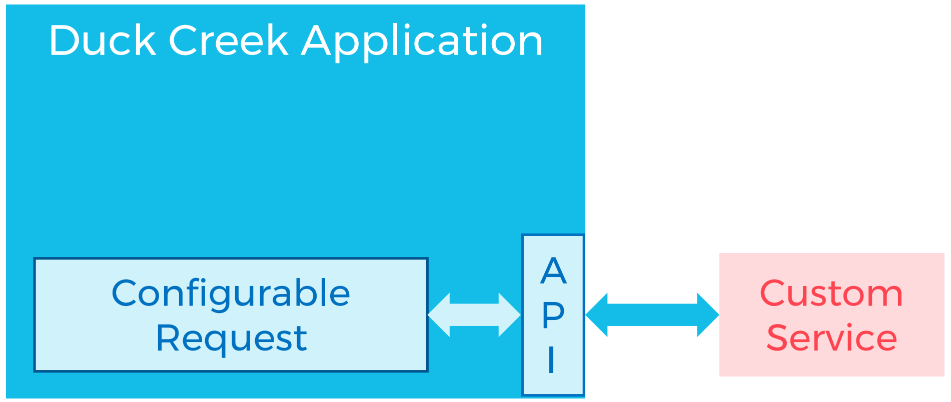
Clean Isolation of Custom Code

Existing Customization Pattern



- Customizations are complied together with Duck Creek code at **build time**
- This can introduce build dependencies, leading to conflicts at time of upgrade

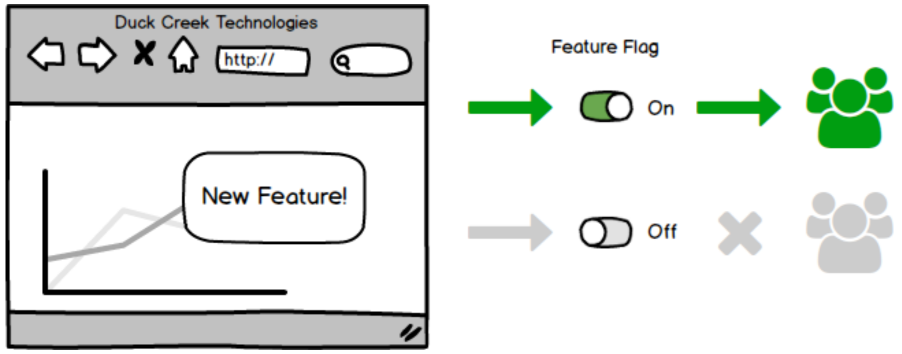
Improved Customization Pattern



- Customizations are externalized as stand-alone services, integrated via APIs at **run time**
- Standard features are used to configure requests to the external services
- The request configurations and APIs are maintained backward-compatible across releases (or feature-flagged)

Result: Eliminate all build conflicts resulting from upgrades

Feature Flags



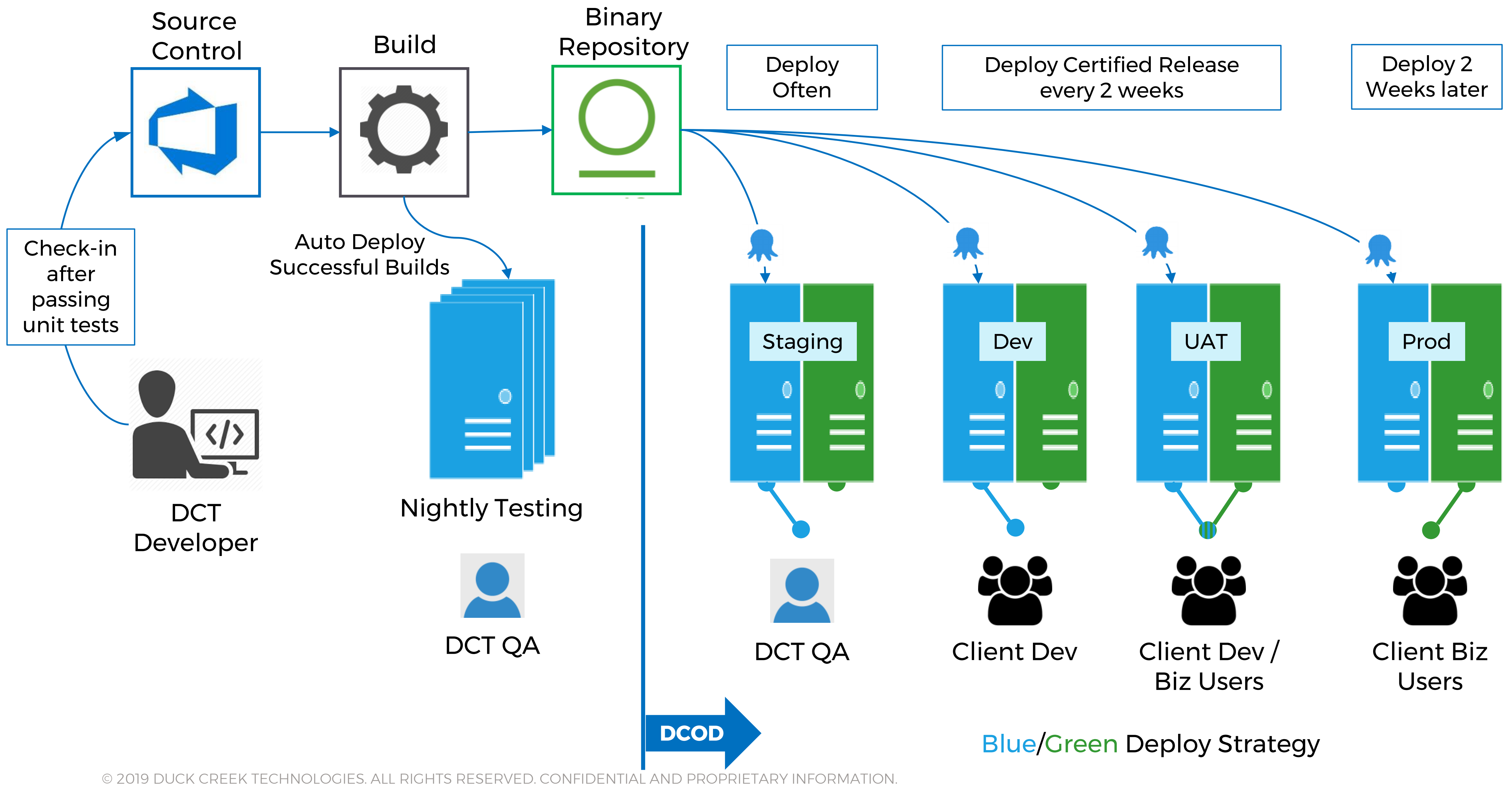
"Feature Flags" are like on/off switches for software features.

- If a flag is "on", the feature is active / follows new behavior
- If a flag is "off", the feature is not active, or continues with the previous behavior

- All features that change business user behavior or are breaking changes will be flagged
- Most defect fixes will not be feature flagged, but exception may occur
- New "passive" features will not be flagged, i.e.,
  - A new activity type is added, but until a configurator applies that activity, it sits dormant
  - A new API is added, or an existing API is versioned
- Most feature flags will have expiration dates – at which point they default to "on"
- Clients will have ability to manage state of their feature flags in each environment

Result: Adoption of features is de-coupled from code upgrades

Automated DevOps Pipeline / Test Automation



Blue/Green Deploy Strategy

Continuous Updates is Itself is a Solution

Eliminate Hot Fix:

- "Hot Fix" is a special branch of the code where a client-specific fix was created, and released only to that one client.
- Hot Fixes are necessary when the normal update frequency is too slow
- Hot Fixes can be eliminated by rolling critical fixes into continuous updates

Reduce Scope of Change Project

- Lots of change happens over the course of 1 – 4 years
- Very little changes in 2 weeks

What do Carriers need to consume updates?

- One time:
  - If on a version prior to v20, upgrade to v20
  - If any legacy customizations exist, migrate those to the new service-based pattern
  - If a robust automated regression test suite doesn't exist, create it
- Ongoing: Establish a team and processes to...
  - Maintain automated regression test suite
  - Run automated regression test suite on every update
  - Evaluate new / changed features as they become available
  - Maintain an ongoing backlog of desired / required changes
  - Execute against that backlog, being sure to keep feature flag expiration dates in mind

Summary

✓ **Continuous Updates**

Features

- Clean Separation of Custom Code
- Eliminate Hot Fixes
- Feature Flags
- Automated DevOps Pipeline
- Automated Testing
- Small Scale Change

Benefits

- Eliminate Build Conflicts
- Feature Adoption de-coupled from Deployment
- Automated Regression Testing
- No-Touch Updates
- Frequent Updates
- Faster Innovation

Carrier To-Do's

- Create robust automated regression test suite
- Create ongoing change team and processes
- Run regression tests on every update
- Evaluate changes
- Maintain backlog
- Execute against backlog

Thank You!