The Three Stages of Automation

Stage #3: Developing Services

2018-03-08

Today's Presenters

John Malzahn
Senior Manager, Cloud and Virtualization Solutions Marketing

Cisco Systems

Carl Moberg
Technology Director, Cloud Solutions and Platform Group

Cisco Systems

Todays Agenda

1. Introducing the Three Stages of Automation
2. Deep Dive on Stage #2 *Developing Services*
3. Using NSO as a Service Development Platform
4. Demo Time!
5. Wrap-up and Q&A

Key Market Trend Observations

Execution at the speed of software

- Networks provides well-known utility abstractions
- Agility, DevOps, NFV, SDN drives new expectations

Changing customer behavior and new expectations

- Everything on demand
- New services with a press of a button

Rapidly changing business models

- Cloud services, virtualization, programmable networks
- New value chains including OTT Co-opetition

All of this requires successful, flexible automation.
But complexity has destroyed many automation initiatives.

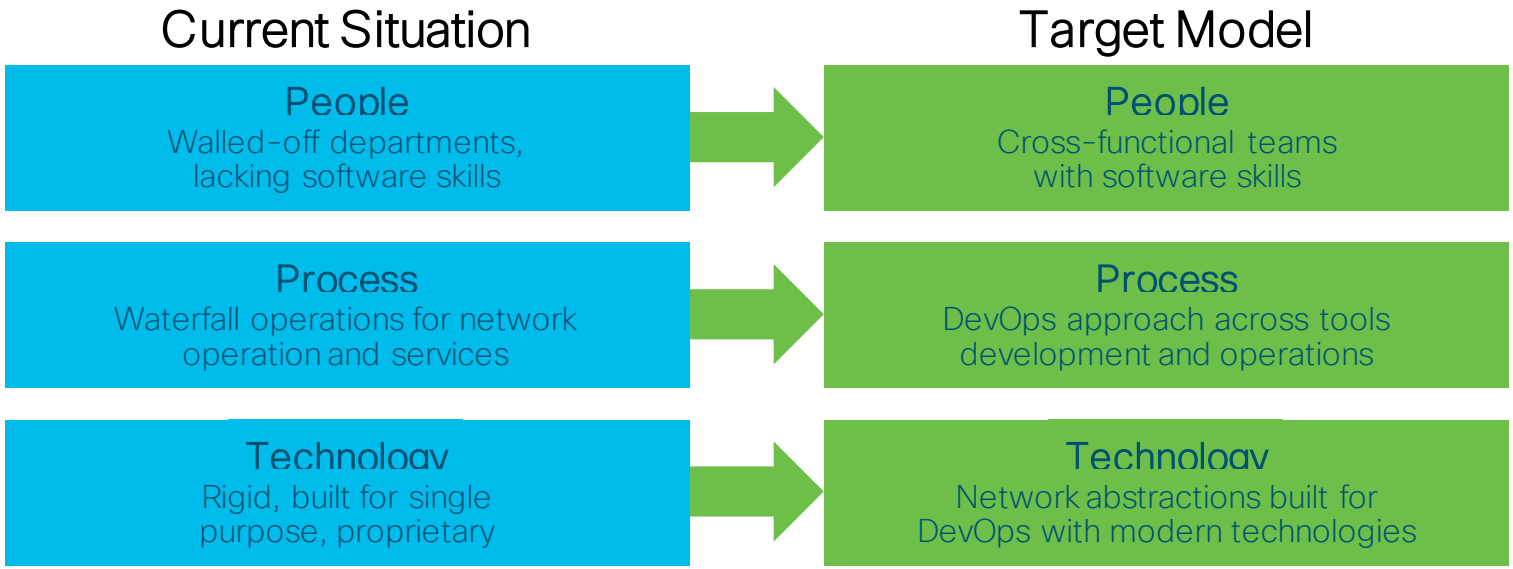
Departmental Pain Points

Network Engineer "Automation"	Ops and Provisioning Team "Customer Experience"	Service Developers "Time-to-Market"
Day-to-day management of rapidly growing, complex networks **Challenges** • Error-prone manual tasks • Growing backlog • Virtualization is coming	Provisions services and manages service quality in networks **Challenges** • No service insight • Lack of automation • Quality issues in delivery	Develops new network services on demand **Challenges** • Implementation time • Cost of change • Lack of tooling

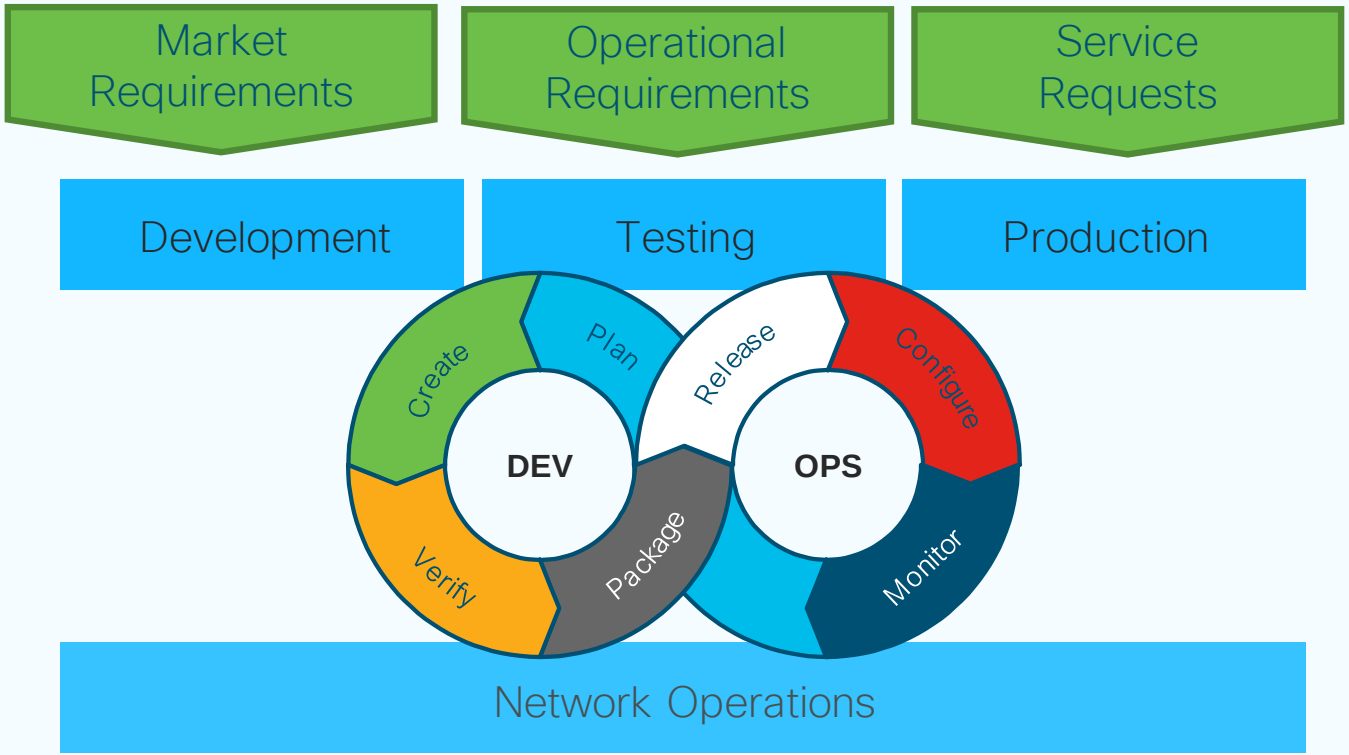
Transition Towards Automation

Network Engineer "Automation"	Ops and Provisioning Team "Customer Experience"	Service Developers "Time-to-Market"
Day-to-day management of rapidly growing, complex networks	Provisions services and manages service quality in networks	Develops new network services on demand
Challenges • Error-prone manual tasks • Growing backlog • Virtualization is coming	**Challenges** • No service insight • Lack of automation • Quality issues in delivery	**Challenges** • Implementation time • Cost of change • Lack of tooling
Network API 1 Utilize a single interface to all network devices	**Service Abstraction** 2 Leverage one central API for all services	**Transformation** 3 Develop your own services

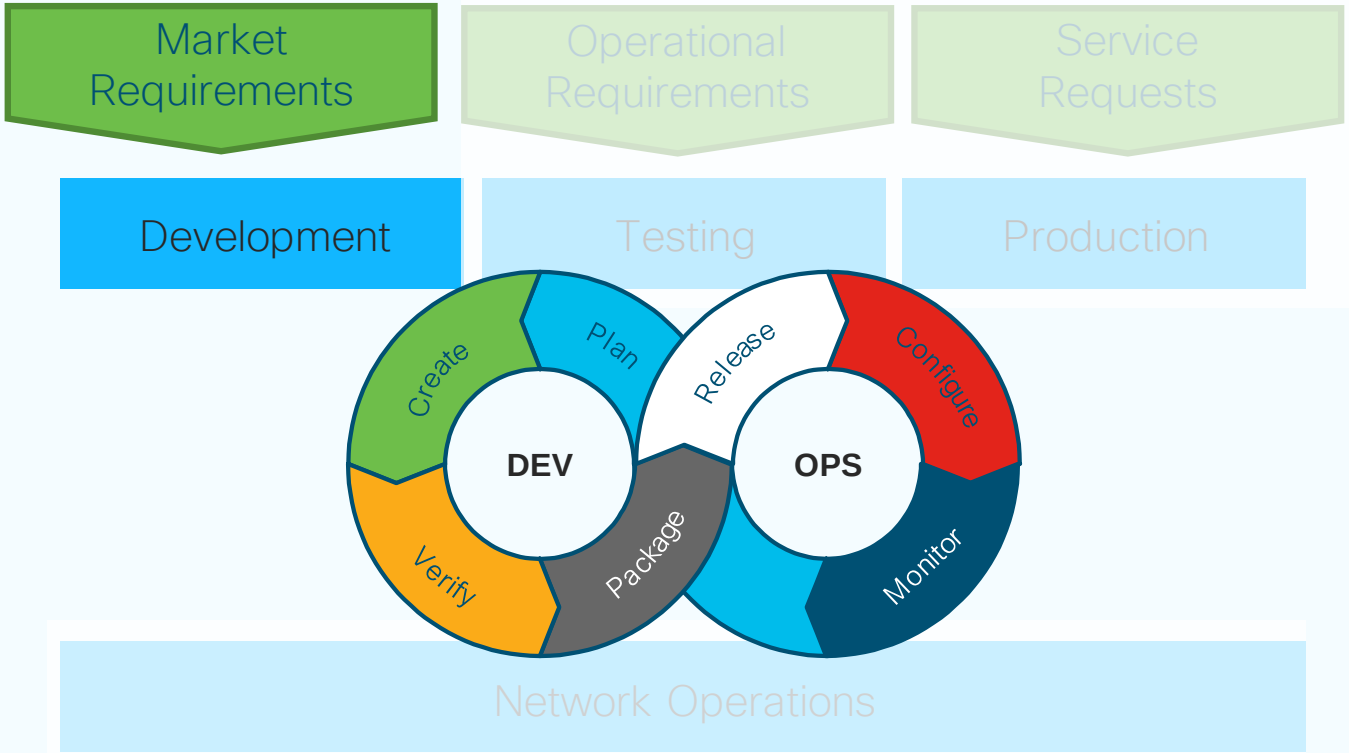
Change Requirements



Devops Virtual Cycle



Devops Virtual Cycle (Todays Focus)



Transition Towards Automation

Network Engineer "Automation"	Ops and Provisioning Team "Customer Experience"	Service Developers "Time-to-Market"
Day-to-day management of rapidly growing, complex networks Challenges • Error-prone manual tasks • Growing backlog • Virtualization is coming	Provisions services and manages service quality in networks Challenges • Lack of automation • Quality issues in delivery • No service insight	Develops new network services on demand Challenges • Implementation time • Cost of change • Lack of tooling
Network API Utilize a single interface to all network devices 1	Service Abstraction Leverage one central API for all services 2	Transformation Develop your own services 3

Challenge Mapping

Implementation Time

- Informal specifications and manual steps
- Muddled boundaries betweem services and resources

Long time-to-market for new services

Challenge Mapping

Implementation Time

- Informal specifications and manual steps
- Muddled boundaries betweem services and resources

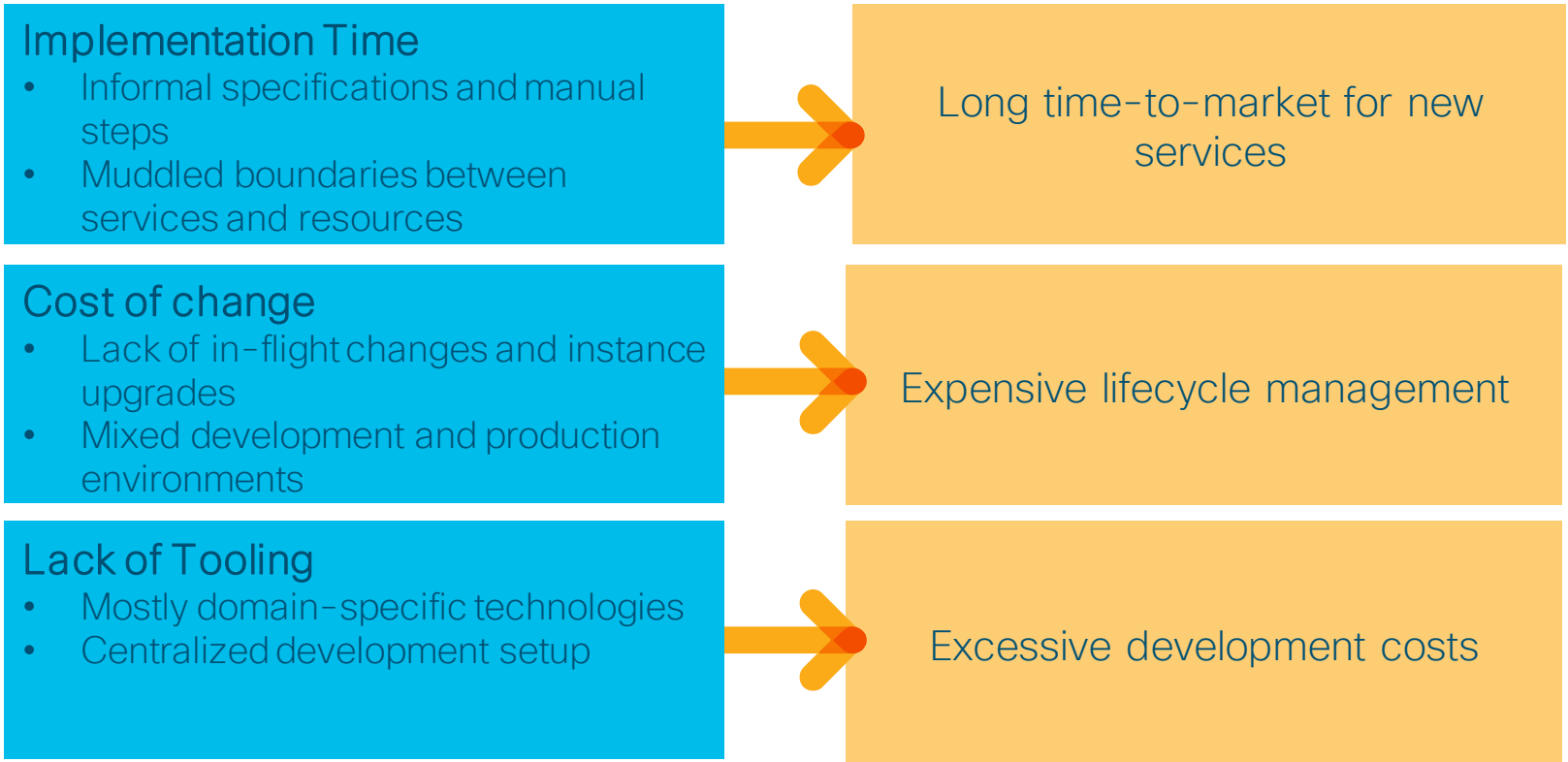
Long time-to-market for new services

Cost of change

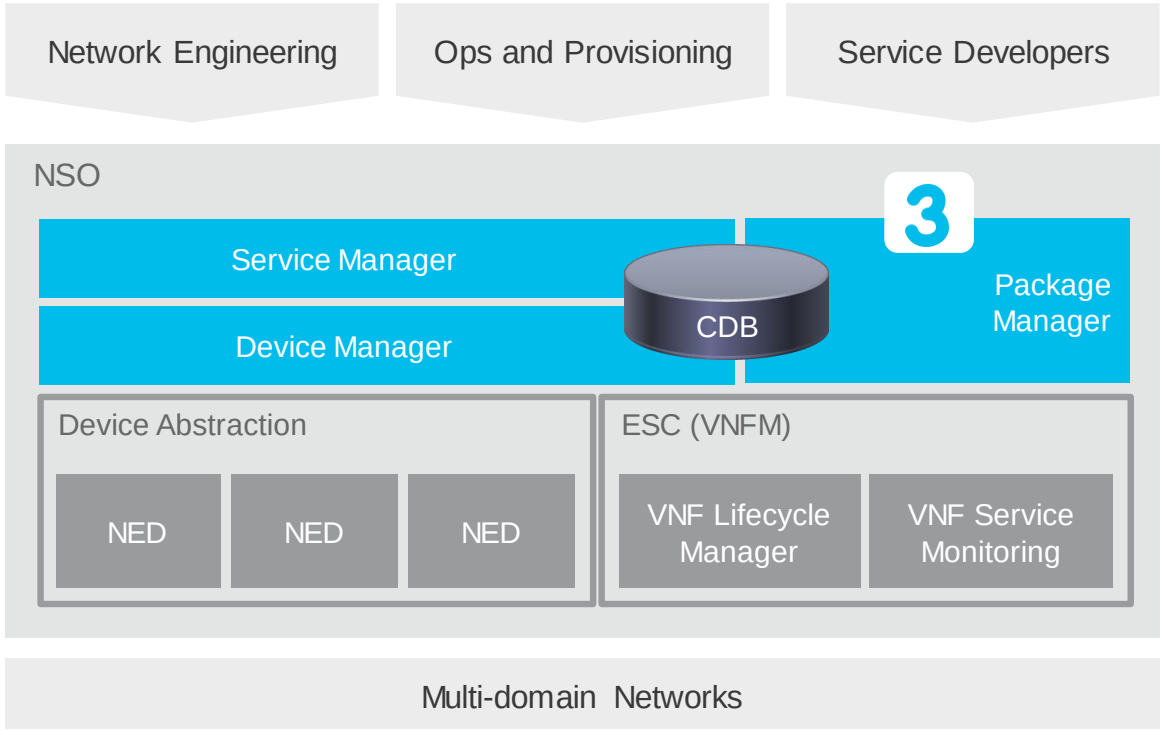
- Lack of in-flight changes and instance upgrades
- Mixed development and production environments

Expensive lifecycle management

Challenge Mapping



Quick System Overview



- Model-driven end-to-end service lifecycle and customer experience in focus
- Seamless integration with existing and future OSS/BSS environment
- Loosely-coupled and modular architecture leveraging open APIs and standard protocols
- Orchestration across multi-domain and multi-layer for centralized policy and services across entire network

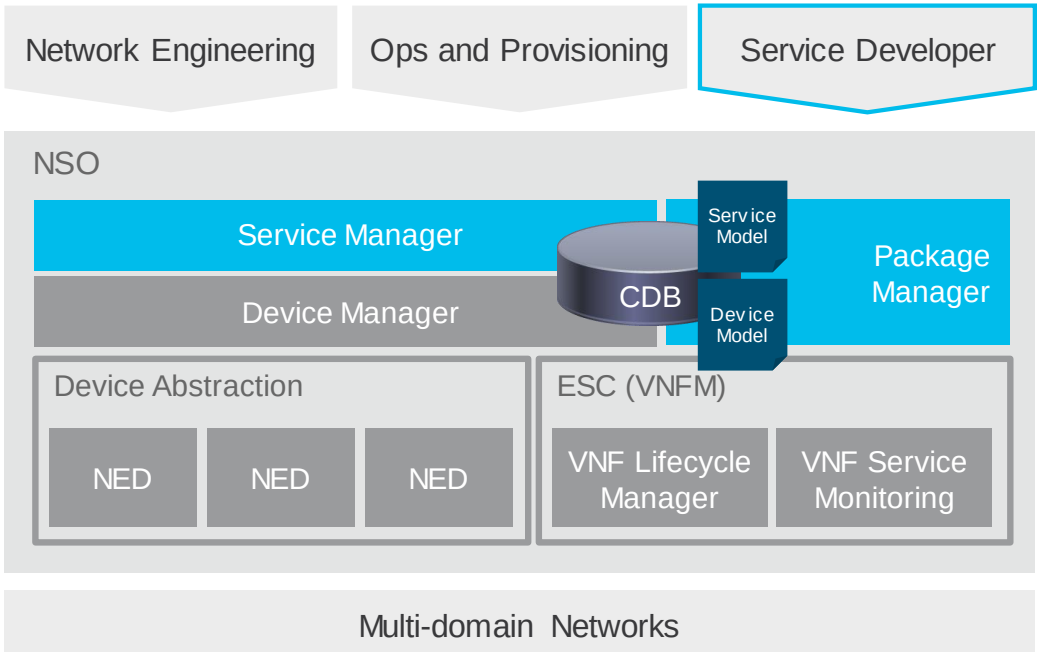
Feature Mapping #1 *Developing With Service Models*

The Challenge

- We need:
  - A development approach focused on designing network services and implementing them on the network, automated end to end…
  - allowing for rapid service development meeting end-user needs and expectation
- On:
  - Brownfield networks
  - Across place in network, vendor, device type and protocol

Leverage the formality of a model-driven approach to significantly reduce the amount of coding needed

Model-based Architecture



- NSO assumes nothing about:
  - Network services
  - Network devices
- All data sets strictly defined by YANG models
- Tree-to-tree mapping reduces coding for lifecycle to absolute minimum

Create → Update → Delete

- Easy
- Given a set of service-level inputs, provide a known and valid output to network
- May require some additional resource collection to fulfill the configuration set

- Challenging
- Allow arbitrary changes to the network service
- May require collecting or handing back resources to fulfill configuration set

- Hard
- Delete any given instance of a service and clean up the resources
- May require reference counting for shared resources

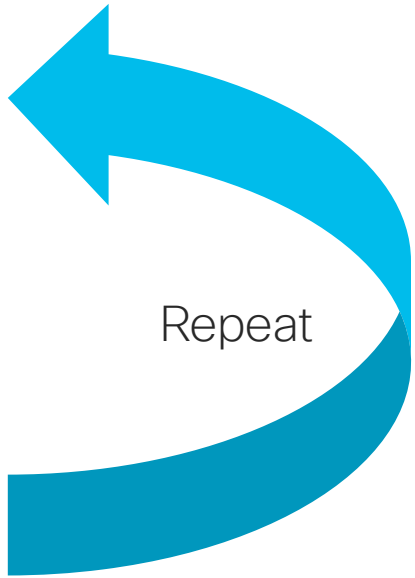
The ability to dry-run all operations is key for trust

High-level Development Process

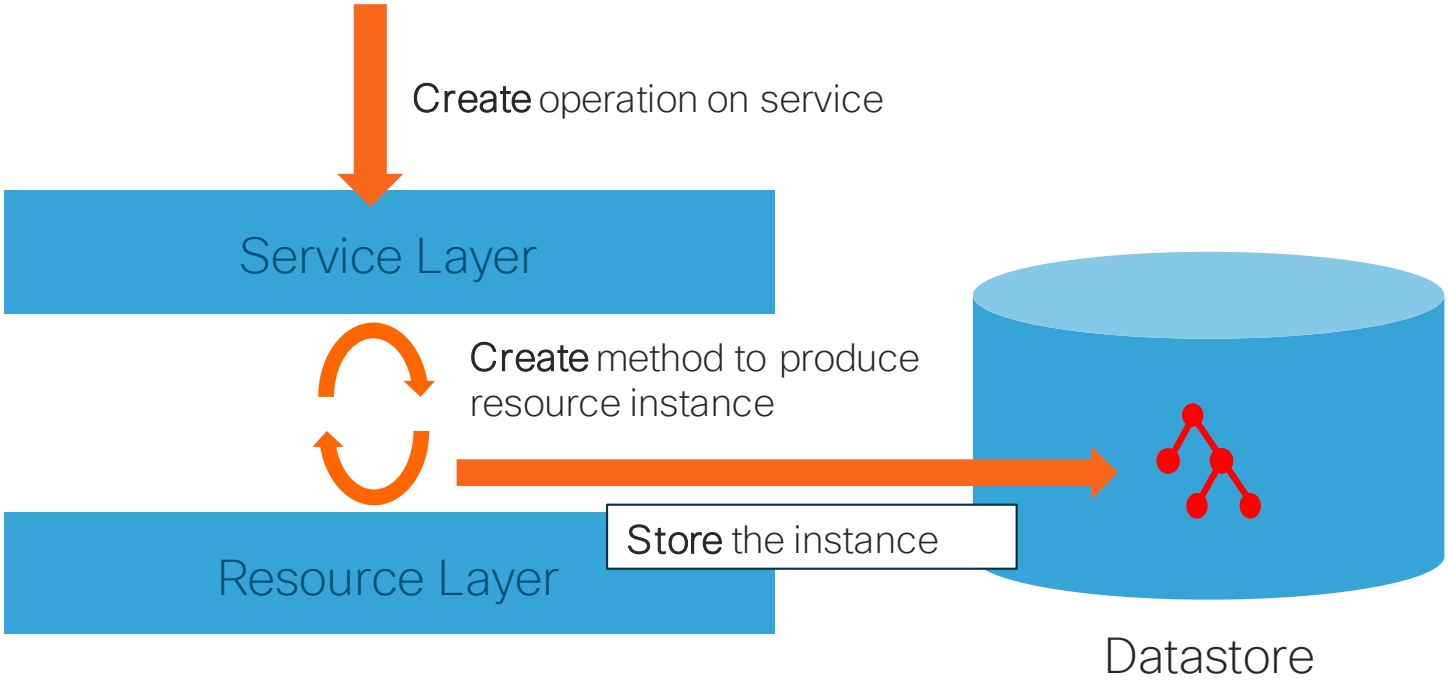
Model the service structure and syntax using YANG and the related developer tools (validator, compiler) together with the users

Map the relationship between the service model layer and the resource/device layer using templates and FASTMAP code (Python or Java)

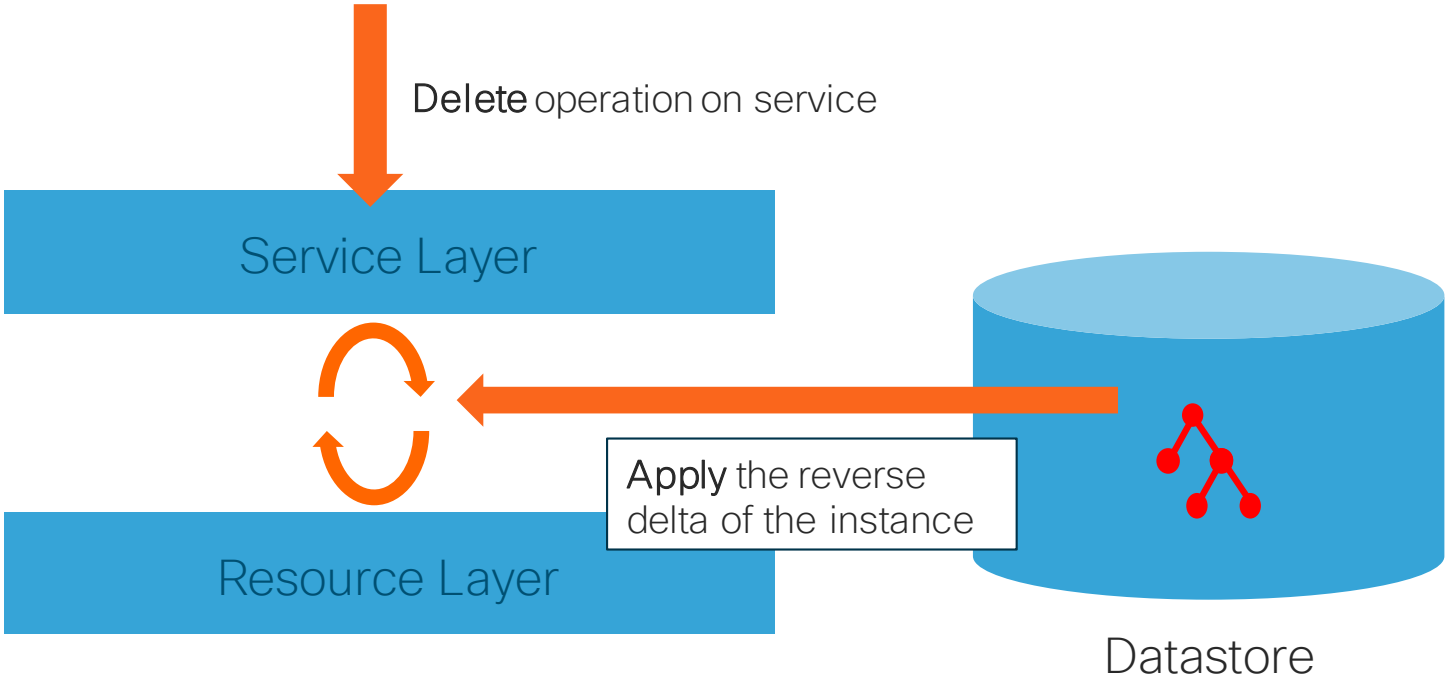
Test the mapping by running CRUD operations on the service layer using dry-runs and the netsim environment



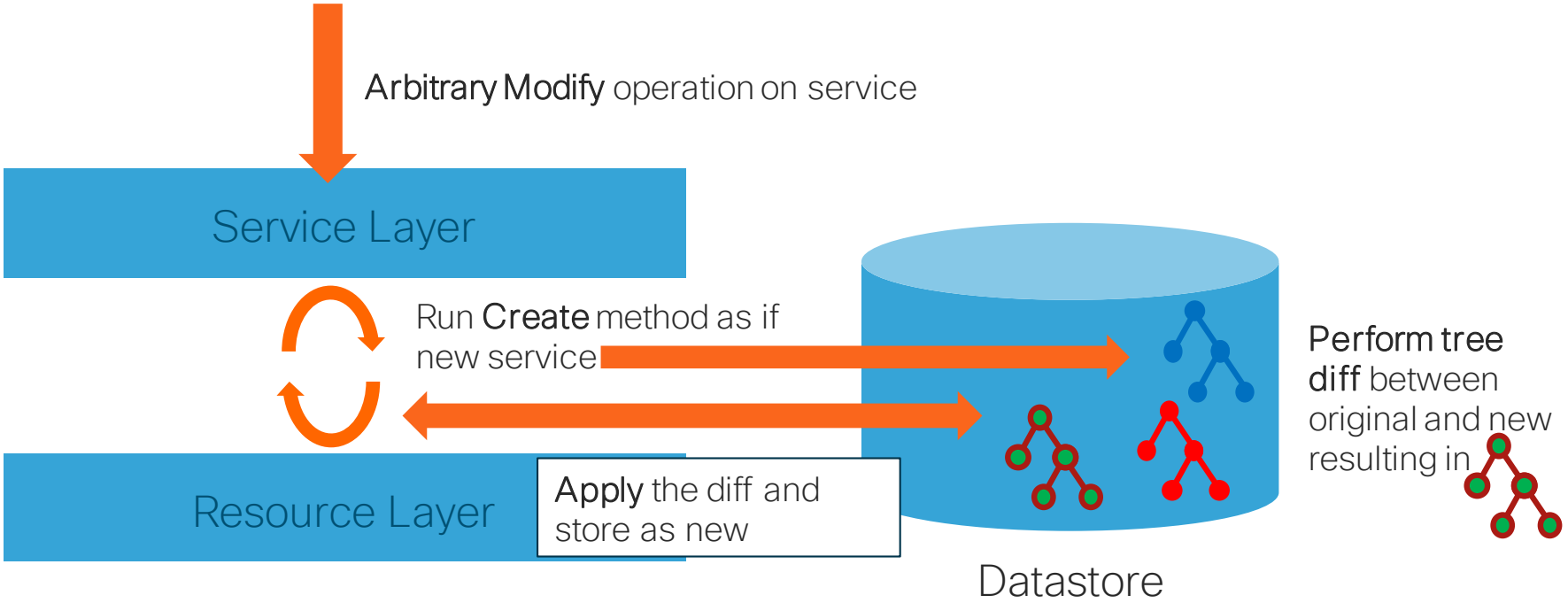
State Convergence: Create Service Instance



State Convergence: Delete Service



State Convergence: Modify Service



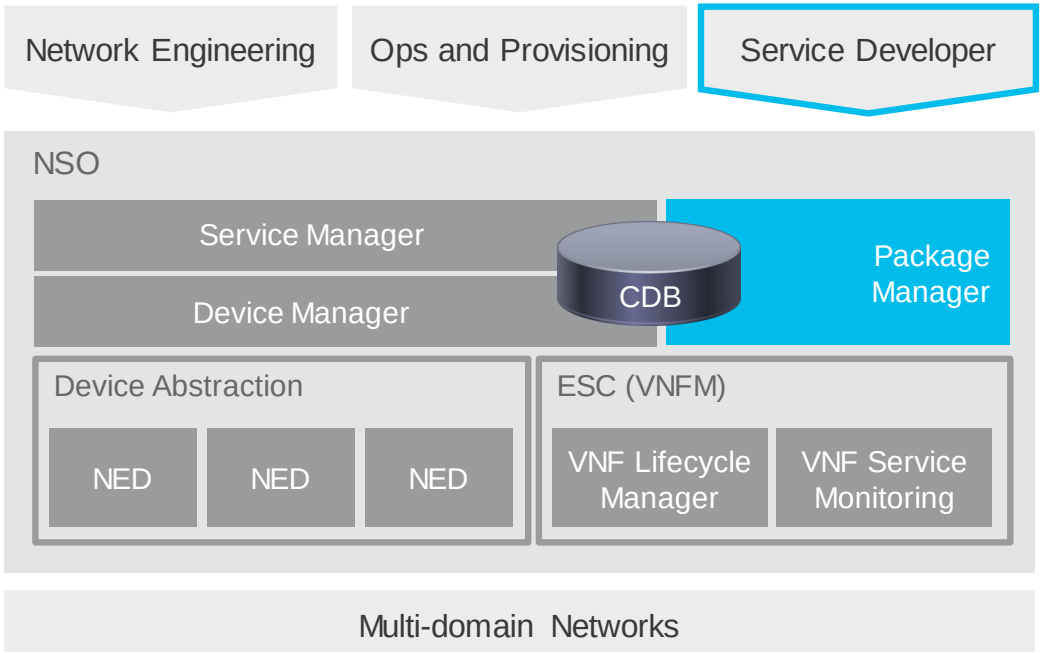
Feature Mapping #2
The Service Model Lifecycle

The Challenge

- We need:
  - To be able to comfortably support the whole lifecycle of service definitions (create, update, retire)…
  - allowing service developers to quickly bring up services and iterate over them as requirements change.
- On:
  - Brownfield networks
  - Across place in network, vendor, device type and protocol

Manage service implementations as software packages with versioning, upgrade features and lifecycle tooling

The Package Manager



Well-defined management of packaged applications, including:

- Install, upgrade, uninstall
- Strict versioning
- Dependencies resolution
- Isolation
- Bundle management
- Distribution across clusters

The Role of Service Packages

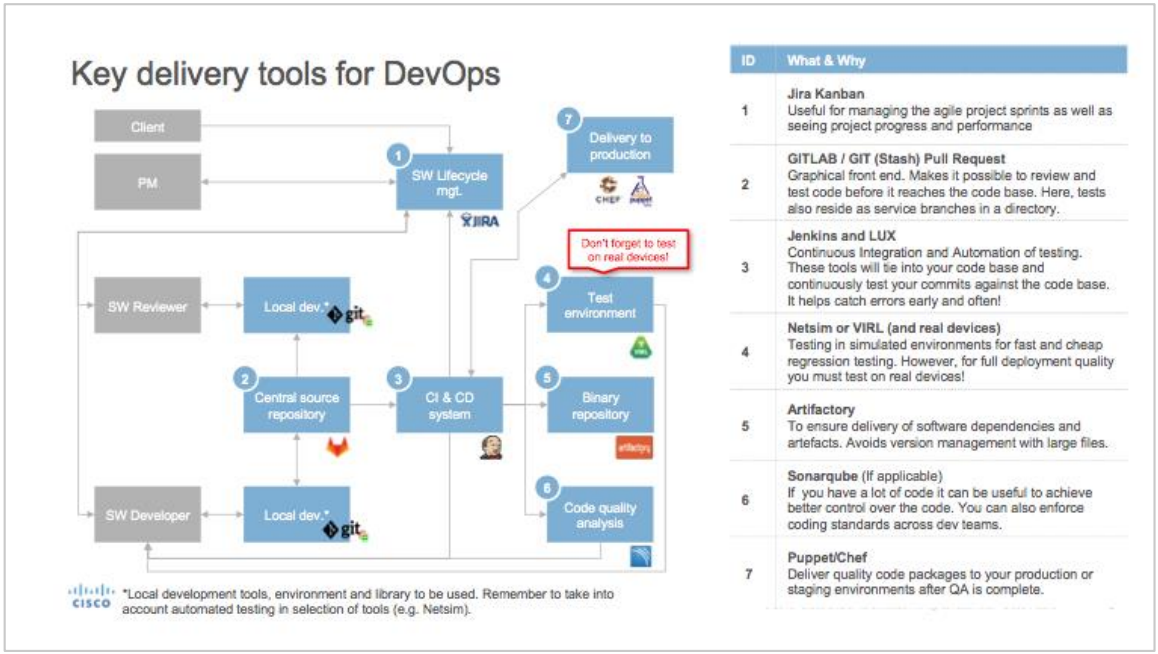
- All user-defined code that needs to run in NSO is delivered as a package
- A package is basically a directory of files with a fixed file structure consisting of:
  - Package metadata
  - YANG modules
  - FASTMAP code
- Packages are versioned and runtime loaded by NSO

Service Package Content

Path	Content
<package-name>/	The name of the package as reflected by the top-level directory name
./package-meta-data.xml	Package-related metadata including version, NSO version requirements, other packages required,
./src/yang	The YANG modules for the package
./src/java ./python ./templates	Code to manage operations on service instances

- Packages can also be delivered as tar-files or gzip-compressed tar-files
- Packages are built for specific versions of NSO

Putting it all Together



ID	What & Why
1	**Jira Kanban** Useful for managing the agile project sprints as well as seeing project progress and performance
2	**GITLAB / GIT (Stash) Pull Request** Graphical front end. Makes it possible to review and test code before it reaches the code base. Here, tests also reside as service branches in a directory.
3	**Jenkins and LUX** Continuous Integration and Automation of testing. These tools will tie into your code base and continuously test your commits against the code base. It helps catch errors early and often!
4	**Netsim or VIRL (and real devices)** Testing in simulated environments for fast and cheap regression testing. However, for full deployment quality you must test on real devices!
5	**Artifactory** To ensure delivery of software dependencies and artefacts. Avoids version management with large files.
6	**Sonarqube** (If applicable) If you have a lot of code it can be useful to achieve better control over the code. You can also enforce coding standards across dev teams.
7	**Puppet/Chef** Deliver quality code packages to your production or staging environments after QA is complete.

Much more about this topic including best practices available at the [NSO Developer Hub]

Feature Mapping #3
Development Tools

The Challenge

- We need:
  - A low barrier of entry for service developers to get productive developing service packages…
  - using well-known and robust technology choices along well-known best practice processes.
- On:
  - Brownfield networks
  - Across place in network, vendor, device type and protocol

Allow developers to plan for, create, dev-test and ship service packages from their local development environment

Developer Tools and SDK Content

- Dev-local multi-vendor network simulator
- Full production-grade installation in dev environments
- YANG tools including validator, compiler
- Project tooling for managing package sets

- Dev-local multi-vendor network simulator
- Build- and runtime validation of package content
- Offline-tools for validating version migration

- Self-contained and versioned package format
- Hitless package installation and version migration
- Local or remote project and package locations

The `netsim` Network Simulator

- A lightweight, developer-local network simulation framework
- Uses a combination of *confd* and NEDs to bring up and expose simulated devices
- Allows developers to continuously develop against reasonably realistic network equipment

```
$ ncs-netsim create-network cisco-ios 6 ios
```

The `ncs-project` tool

- Create new projects using the `ncs-project create` command
- Define what packages to use in the `project-meta-data.xml` file.
- Fetch packages with the `ncs-project update` command
  - Local files, git
- Export the project using the `ncs-project export` command

```
$ ncs-project create myproject
$ …edit the project metadata to pull in packages…
$ cd myproject && make all
$ ncs-project export
```

Demo Time

Demo Flow

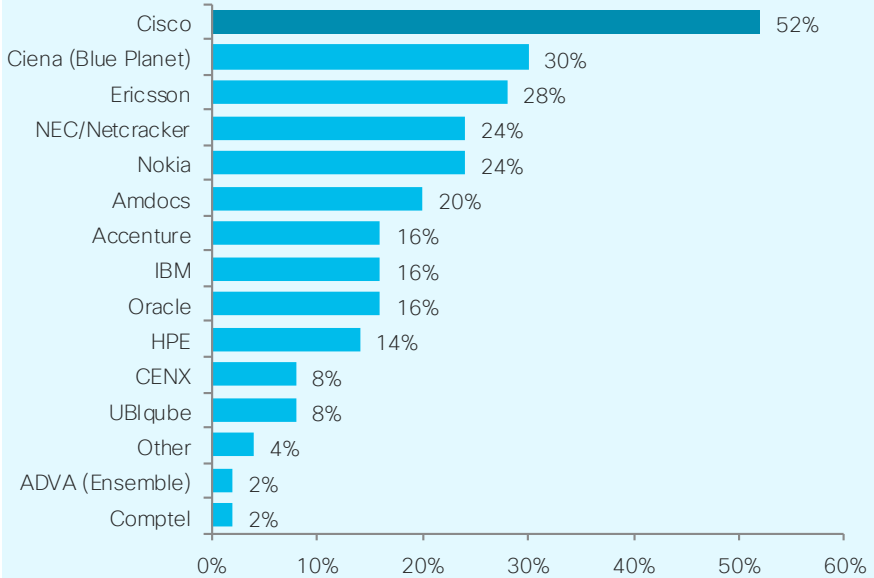
- **Prepare** a development environment including
  - NSO runtime project and NEDs
  - Netsim setup with IOS-XE and IOS-XR devices
- **Model** a simple MPLS VPN service
- **Map** service and NEDs using template
- **Test** the create-update-delete loop

Wrap-up and Q&A

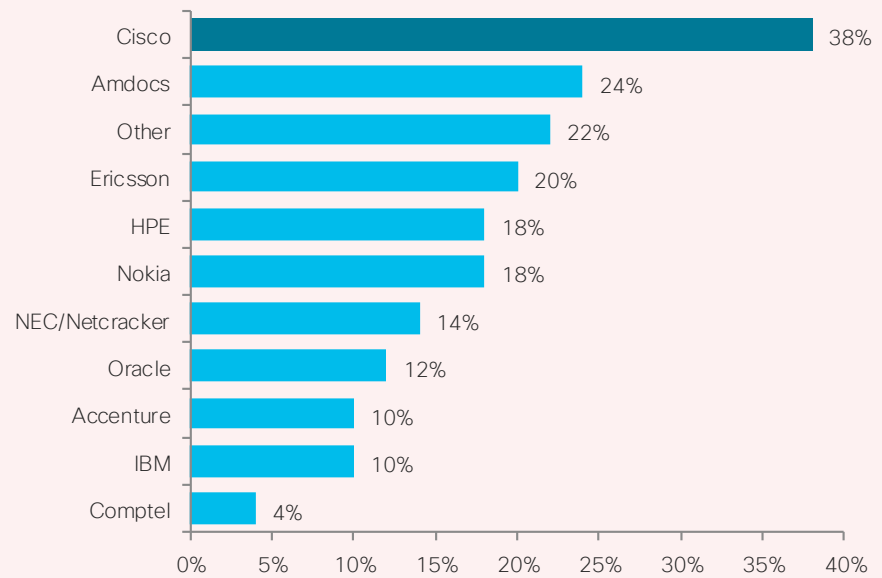
Operators Voted

Cisco Leads Industry In Lifecycle Service Orchestration

LSO and OSS Leaders



OSS Vendors Deployed Today



Source: SDxCentral 2017 Next-Gen OSS and the Rise of LSO Report

What You Gain with Cisco Network Services Orchestrator, Enabled by Tail-f

- Agility throughout service lifecycle
  - Strict YANG model-driven solution
  - Auto-rendered business logic results in 90% less code
  - Effortlessly re-deployment of updated service and device models
  - DevOps for differentiation
- Full automation
- Robust and proven in tier-1 deployments
- Industry's broadest multivendor support
- Relevant in today's and tomorrow's networks

NSO DevNet – Key Highlights

The one place to use for sharing, finding and collaborating on NSO public knowledge!

Light start through DevNet content page and Learning-Labs

Constant news and updates to help you keep up to date

Large searchable content pool

Cisco customers, partners and employees all have access

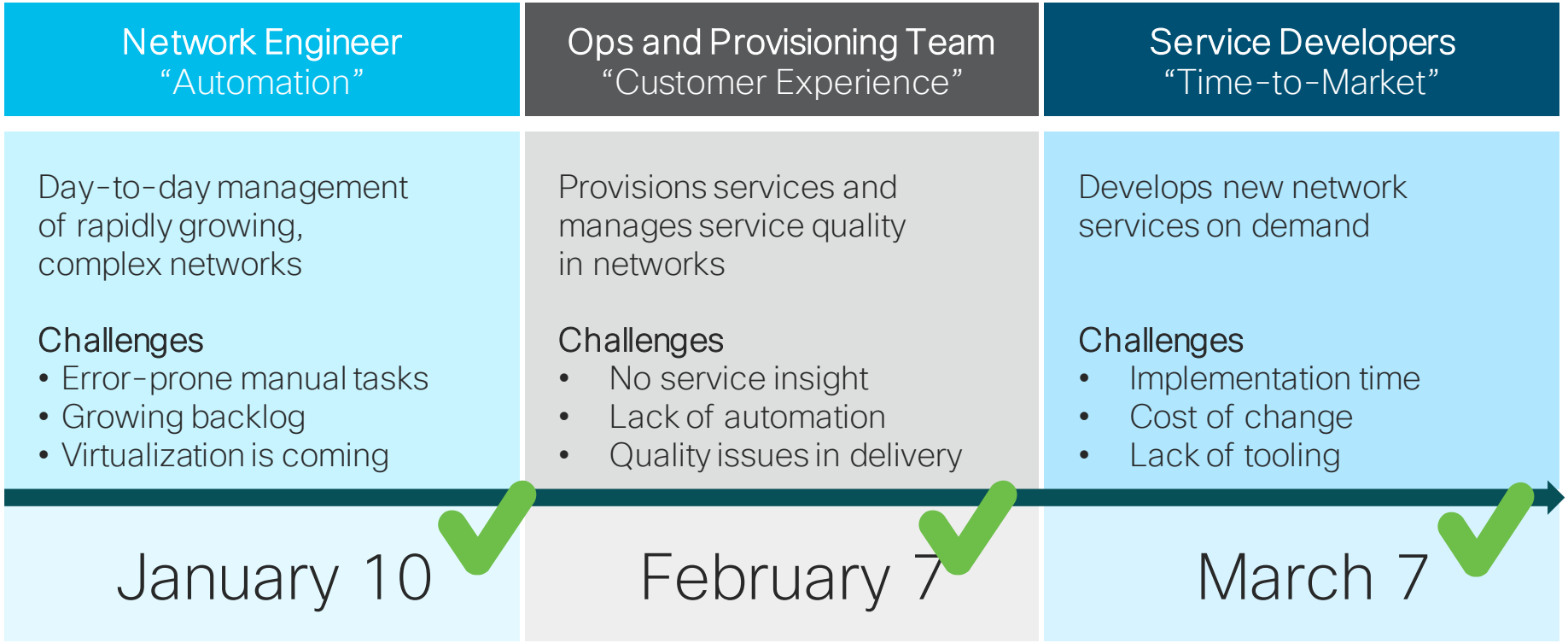
Got a question, ask! We will help ensure a fast response

Easy to share and find public content

Code sharing through public GitHub

developer.cisco.com/site/nso

All Webinars in the *Three Stages* Series



Questions?


CISCO