Rachel Laycock @rachellaycock
Continuous Delivery Explained

ORLANDO
FLORIDA
AUGUST 7-11

AGILE 2017
Delivering value early and often
“How long would it take your organization to deploy a change that involves just one single line of code?”

“Can you do this on a repeatable, reliable basis?”
Continuous Delivery is **BIG**

- Organisational Alignment
- Release Management
- Architecture
- Quality Assurance
- Continuous Integration
- Configuration Management
- Data Management
- Environments & Deployment
Who are You?

Manager

DBA

BA

UX

QA

Operations

Developer
Agile Transformation

Manager

DBA

BA

Developer

UX

QA

Operations
Continuous Delivery

Manager

DBA

BA

UX

QA

Operations

Developer
Agile 101

Iteration: 0 1 2 3 4
- Analysis + Design
- Development
- Testing + Showcase

"Agile" team

Centralized QA
Integration + QA
Release and operation

Customer

IT Operations

The "last mile"
No matter how it looks at first, it's always a people problem.
Continuous Integration

Integration early and often.

Everyone checks into trunk at least once a day.
Bring the pain forward.
eager vs. late
Continuous Integration

Fast, automated feedback on the correctness of your application every time there is a change to code
Integration

Integration early and often.

Everyone checks into trunk at least once a day.

Continuous Deployment

Deploy as the final stage of continuous integration.
Integration
Integration early and often.
Everyone checks into **trunk** at least once a day.

Deployment
Deploy as the final stage of continuous integration.

Continuous Delivery
Software is always in a deployable state.
So what’s your plan?
Deployment Pipelines
Deployment Pipeline

Fast, automated feedback on the production readiness of your application every time there is a change — to code, infrastructure or configuration.
Pipeline Construction

Pipeline stages = feedback opportunities

increasing confidence in production readiness
commit Stage

Run against each check-in

Starts building a release candidate

If it fails, fix it immediately
**UAT Stage**

- Version control
  - acceptance tests
  - deployment scripts
  - configuration data

- Test reports
- Artifact repository

End-to-end tests in production-like environment

Triggered when upstream stage passes

First DevOps-centric build
UAT, staging, integration, production, ...

Push versus Pull model

Deployments self-serviced through push-button process
Machinery

continuous integration ++

Jenkins

www.thoughtworks.com/products/go-continuous-delivery
Pipeline Anti-patterns

insufficient parallelization

Commit stage
- Compile
- Unit test
- Analysis
- Build installers

Acceptance test stage

User acceptance testing

Performance testing

Production

ideal time: < 10 minutes
Pipeline Anti-patterns

insufficient parallelization

Commit stage
- Compile
- Unit test
- Analysis
- Build installers

Acceptance test stage

User acceptance testing

Performance testing

Production

Mingle:
3,282 test / 53 computers = ~1 hour
Insufficient Parallelization Heuristic:

make your pipeline *wide*, not *long*

parallelize each stage as much as you can

create more stages if necessary to optimize feedback
Pipeline Anti-patterns

inflexible workflow

pipeline fans out as soon as it makes sense to do so
Principles

automate almost everything

keep everything you need to build, deploy, test, & release in version control

- requirements documents
- test scripts
- automated test cases
- network configuration scripts
- technical documentation

- database creation, upgrade, downgrade, and initialization scripts
- application stack configuration scripts
- libraries
- deployment scripts
- tool chains
Infrastructure Consistency

boxen.github.com
Identify & remove friction
Continuous Delivery Metrics

**lead time**
the time between the initiation and completion of a production process.

**cycle time**
the total elapsed time to move a unit of work from the beginning to the end of a physical process.
Potential Hindrances

- Lead time is too long
- Last mile is too painful
- Poor collaboration
Prerequisites

- comprehensive configuration management
- continuous integration
- excellent automated testing at all levels
Modern software is complex!

https://www.thoughtworks.com/insights/blog/implications-tech-stack-complexity-executives
Testing

effort

audience

feedback
source: Brian Marrick, *Continuous Delivery* (Humble/Farley), with modifications
Testing Quadrants

- automated functional acceptance
- integration
- automated unit
- nonfunctional acceptance / quality of service
  - manual/automated
  - showcases usability
  - exploratory

source: Brian Marrick, Continuous Delivery (Humble/Farley), with modifications
prefer test-driven to test-after development

prefer pragmatism over dogmatic metrics

optimize for the target audience

how much time?
Common Anti-pattern

mixed unit/
functional tests
Testing Quadrants

- **Functional acceptance**
  - Automated

- **Integration**
  - Automated

- **Unit**
  - Automated

- **Nonfunctional acceptance / Quality of service**
  - Manual/automated

- **Contracts**

- **Heartbeats**

- **Plumbing**

source: Brian Marrick, *Continuous Delivery* (Humble/Farley), with modifications
Testing Quadrants

- **automated functional acceptance**
- **manual nonfunctional acceptance**
- **integration**
- **unit**

- business facing
- technology facing
- critique project
- support programming

- showcases usability
- exploratory
- automated
- nonfunctional acceptance / quality of service
- manual/automated

source: Brian Marrick, *Continuous Delivery* (Humble/Farley), with modifications
To Cuke or not to Cuke...

BDD ≠

use when it provides useful feedback to the target audience

http://www.thoughtworks.com/insights/blog/3-misconceptions-about-bdd
Anti-pattern: Ice-cream Cone
Cupcake Anti-pattern

http://www.thoughtworks.com/insights/blog/introducing-software-testing-cupcake-anti-pattern
Avoiding Cupcakes

collaborate
work in sync
cross-role pair programming
story kickoff

test at the lowest level

merge teams
when possible

agree on goals
and metrics
local workstation

version control

continuous integration server
continuous integration
server

version control

build
version control

continuous integration server

build
everyone commits to trunk at least once a day
Feature Branching
Professor Plum
P1  P2  P3
Mainline
B1
Reverend Green
G1  G2  G1-2
B1
B1

Feature Branching
Feature Branching
merge
ambush!

Feature Branching
Continuous Integration removes the pain...
Config File

options:

- wobblyFoobars: true
- flightyForkHandles: false

some.jsp

```java
forkHandle = (featureConfig.isOn('flightyForkHandles')) ? new FlightyForkHandler(aCandle) : new ForkHandler(aCandle)
```

other.java
Togglz

What is it about?

Togglz is an implementation of the Feature Toggles pattern for Java. Feature Toggles are a very common agile development practices in the context of continuous deployment and delivery. The basic idea is to associate a toggle with each new feature you are working on. This allows you to enable or disable these features at application runtime, even for individual users.

Want to learn more? Have a look at an usage example or check the quickstart guide.

News

01-Jul-2013

Togglz 2.0.0.Final released

I’m very happy to announce the release of Togglz 2.0.0.Final. This new version is the result of many months of hard work. Many core concepts of Togglz have been revised to provide much more flexibility.

The most noteworthy change in Togglz 2.0.0.Final is the new extendible feature activation mechanism that allows to implement custom strategies for activating features. Beside that there are many other updates.
removed as soon as feature decision is resolved

Feature toggles are purposeful technical debt added to support engineering practices like Continuous Delivery.
build-time vs. run-time
Branch by Abstraction
Application implements Interface

Interface implements New Library

New Library

Library
Application

Interface

Library
implements

New Library
Application

New Library

Interface

implements
“Strangler” Pattern

make something new that obsoletes a small percentage of something old

put them live together

rinse, repeat
Release branches are OK...

Long-lived branches damage continuous integration.
Release Strategies
blue-green deployments

canary releases

Incremental Release Strategies

dark launching
blue-green deployments
Canary Releasing
Canary Releasing
Canary Releasing

- reduce risk of release
- multi-variant testing
- performance testing
Dark Launching
Continuous Delivery

- Organisational Alignment
- Release Management

- Architecture
- Quality Assurance
- Continuous Integration
- Configuration Management
- Data Management
- Environments & Deployment
essential complexity in data

persistent

impedance mismatch

bulky
accidental complexity in data

refactoring

poor collaboration

late integration

manual work
DB Evolution & Deployment

- scripting all db changes incrementally
- db refactoring
- decouple db migration from app migration
Continuous Integration for Databases
Prepare environment
Deploy app
Create dbs, apply schema
Add app reference data
Run acceptance tests

version control
commit stage
acceptance stage

test
double

service 1

service 2

Test runner
DbDeploy Pattern
DbBeploy Tool

db updates are code

small incremental deltas

metadata in the database

fail fast

001_create_initial_tables.sql:

CREATE TABLE customer (  
id BIGINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1)  
PRIMARY KEY,  
firstname VARCHAR(255),  
lastname VARCHAR(255)  
);

002_add_customer_date_of_birth.sql

ALTER TABLE customer ADD COLUMN dateofbirth DATETIME;

--//@UNDO

ALTER TABLE customer DROP COLUMN dateofbirth;
For DB CI We Need To:

- start with a clean database
- apply changes incrementally
- use the same process everywhere
- be comprehensive in change management
Apply Deltas

run each delta in order

stop the line if one delta fails

auto-rollback if possible

record success in db metadata table
Refactoring Databases
Decouple DB Updates: the Expand/contract Pattern

Every change you make must be backward compatible

Decouple DB Updates:

Decouple DB Updates:

Decouple DB Updates:

Decouple DB Updates:

Decouple DB Updates:

Decouple DB Updates:

Decouple DB Updates:

Decouple DB Updates:

Decouple DB Updates:

decouple db updates

- dark launch db updates
- long-running upgrades
- abstraction layer in code or stored procs / views

Never tie DB migrations to application deploys
DB Deployments Still Hard

practice, practice, practice

fail fast

bring the pain forward

refactor the db

update engineering practices
Managing Environments & Infrastructure
The Pain of Operations

legacy applications
The Pain of Operations

heterogeneous platforms
The Pain of Operations

poor quality software thrown over a wall
The Pain of Operations

inordinate amount of firefighting
The Pain of Operations

conservative, process heavy
The Pain of Operations

huge budget for operations
Horror Stories

dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/

DOUG SEVEN

“bankrupt in 45 minutes”

Knighmare: A DevOps Cautionary Tale

I was speaking at a conference last year on the topics of DevOps, Configuration as Code, and Continuous Delivery and used the following story to demonstrate the importance making deployments fully automated and repeatable as part of a DevOps/Continuous Delivery initiative. Since that conference I have been asked by several people to share the story through my blog. This story is true – this really happened. This is my telling of the story based on what I have read (I was not involved in this).

This is the story of how a company with nearly $400 million in assets went bankrupt in 45-minutes because of a failed deployment.

Background
DevOps

not it’s own silo, but a liaison between operations and developers

at inceptions, showcases, retros

devs work in ops and carry pagers

devs create more deployable software
Managing Infrastructure

infrastructure = environments and supporting services (networking, vcs, storage, mail, dns... )

desired state specified in version control

autonomic (self-corrects to desired state)

state should be known through monitoring
Infrastructure as Code

version all the things

definition files

self-documented systems & processes

keep services available continuously

continuously test systems & processes

small changes over large batches
If someone threw a server out of the window, how long would it take to recreate it?
Tools

- manage many systems
- manage configuration
- enforce consistency
- treat infrastructure as code
Continuous Delivery

Organisational Alignment

Release Management

Architecture  Quality Assurance  Continuous Integration  Configuration Management  Data Management  Environments & Deployment
Components are deployed.

Features are released.

Applications consist of routing.
Conway’s Law

“organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations”

—Melvin Conway

Siloed functional teams... ... lead to silod application architectures. Because Conway’s Law
Inverse Conway Maneuver

Cross-functional teams... organised around capabilities
Because Conway's Law
“team designs are the first draft of your architecture”

- Michael Nygard
“...it isn’t the methodologies that succeed or fail, it’s the teams that succeed or fail. Taking on a process can help a team raise it’s game, but in the end it’s the team that matters and carries the responsibility to do what works for them.”

MARTIN FOWLER
(FLACCID SCRUM, 2009)
Continuous Delivery

reduce friction

automate everything you can

incorporate everyone into Continuous Delivery practices

measure success via cycle time

continue to improve
Thank you / Questions