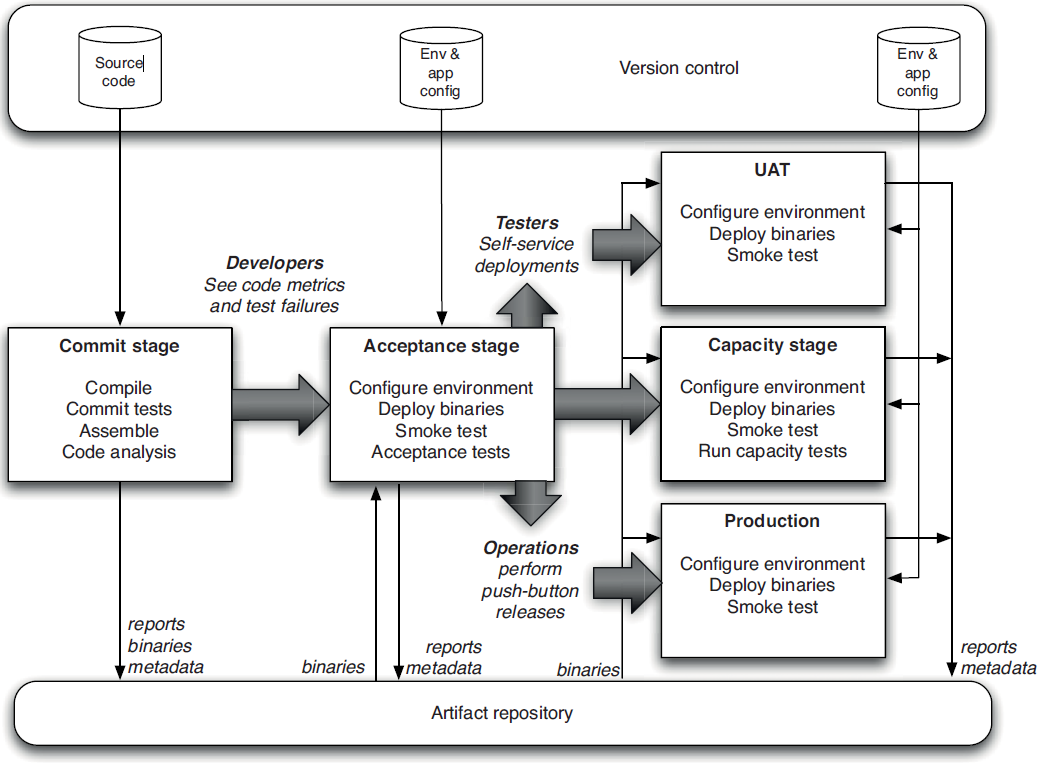


Rysunek 1 source: Continious Delivery, Jez Humble, David Farley



Rysunek 2 source: Continious Delivery, Jez Humble, David Farley

**Deployment pipeline practices:**

- build your binares once

- deploy the same way to every environment

- smoke-test your deployments after deploy

- deploy into a copy of production

- each change should propagate through the pipeline instantly

- if any part of the pipeline fails stop the line

**Commit stage**

Steps:

• Compile the code (if necessary).

• Run a set of commit tests.

• Create binaries for use by later stages.

• Perform analysis of the code to check its health.

• Prepare artifacts, such as test databases, for use by later stages.

Metrics:

• Test coverage (if your commit tests only cover 5% of your codebase, they’re

pretty useless)

• Amount of duplicated code

• Cyclomatic complexity

• Afferent and efferent coupling

• Number of warnings

• Code style

* if it fails – devlopers should quickly fix it or back their changes to last good
* commit stage should provide fast, useful feedback
* Creating Your Own Artifact Repository

**Commit Test Suite Principles and Practices**

Avoid the Database

Avoid Asynchrony in Unit Tests

Should take less than ten minutes

**Automated acceptance testing**

* Verifies that acceptance criteria of a for story are met
* It much cheaper than manual testing
* Acceptance tests are derived from acceptance criteria

**Given** some initial context – represent state of app at the beginning

**When** an event occurs – describes interaction between user and app

**Then** there are some outcomes – test should verify that app is in the ‘then’ state

**Build and Deployment Scripting**

Scripts should be stored in a repository.

***Always Use Relative Paths***

* Create a Script for Each Stage in Your Deployment Pipeline
* Initially can be one scripts
* Use the Same Scripts to Deploy to Every Environment
* Ensure the Deployment Process Is Idempotent
* Evolve Your Deployment System Incrementally



***Testing Your Environment’s Configuration***

* Check if you can retrieve a record from db
* Check if you can contact website
* Check whether message broker has proper set of messages
* Send several “pings” through firewall

**Testing Nonfunctional requirements**

• *Scalability testing.* How do the response time of an individual request and

the number of possible simultaneous users change as we add more servers,

services, or threads?

• *Longevity testing.* This involves running the system for a long time to see

if the performance changes over a protracted period of operation. This type

of testing can catch memory leaks or stability problems.

• *Throughput testing.* How many transactions, or messages, or page hits per

second can the system handle?

• *Load testing.* What happens to capacity when the load on the application

increases to production-like proportions and beyond?

**Deploying and Releasing Applications**

Blue green deployment



Rysunek 3 source: Continious Delivery, Jez Humble, David Farley

* Read only state

Canary releasing



Rysunek 4 source: Continious Delivery, Jez Humble, David Farley

The People Who Do the Deployment Should Be Involved in

Creating the Deployment Process

Log Deployment Activities

Don’t Delete the Old Files, Move Them

Deployment Is the Whole Team’s Responsibility

Server Applications Should Not Have GUIs

Have a Warm-Up Period for a New Deployment – app should be running to fill caches

Don’t Make Changes Directly on the Production Environment

**Splunk – allow to perform real-time searches of logs**

**Version Control**

Use optimistic locking (pessimistic only for binary data)

Ensure that everything you need to build your software is in a version control.

Merging is very time-consuming.

In continuous integration everybody check in to mainline at least once a day.

Create long living branches only for release but every bug fix merge into mainline immediately.

For medium and large team application should be splitted up into components.

**Branch for release**

* Develop features on mainline.
* Create branch when feature is completed
* Only fixes are commited on branches and then merge immediately into trunk.

**Keep application releasable**

* Hide new functionality until it is done

Feature flag

* Make changes incrementally as small changes, each should be releasable
* Use components to decouple parts of application
* Use branch by abstraction

Create an abstraction layer over peace of code you are going to change and when implementation is ready switch to new implementation.