Continuous Delivery
Software-Deployments ohne graue Haare

3. April 2012 – Corsin Decurtins

Some numbers...

4	**15**
deployments per year	deployments per day
bank, insurance company, government, transport authority	social network, Web 2.0, Google Mail, Flickr, Facebook

Some numbers…

On the left

- Traditional application from something like a bank, an insurance company, a government agency, a transport authority … you name it.

On the right

- One of those new fancy social network, Web 2.0-ish application.
- Think Facebook, Google Mail, Flickr, …

Both are serious applications with serious business value.

Some numbers…

Both applications need a high up-time.

Deployments to the production environment are risky.

Both companies try to reduce the risk of deployments.

Two completely different approaches to do that:

- "Banks" try to reduce the risk and effort for deployments by reducing the number of deployments.
- The "social networking" site tries to reduce the risk of deployments by making much, much smaller deployments … but a lot of them.

Some numbers...

The questions now are:

a) Which organization is right?

b) Where in this range are you?

c) Where in this range should you be?

That's what I want to talk about today.

That and why moving toward the right side might be the better choice for the future.

Deployment

Installation of a **software release** on a **target environment**

Replacement of existing software releases

Data migration

System reconfiguration

Deployment Complexity

Data migration

Multiple components

Synchronized deployments

Dependencies

Deployment Risks

Server not starting

Out of disk space

Database migration failure

Configuration typos

Communication failure

Version mismatch

Bad Performance

Tuning issues

Deployment Time

Wrong dependency version

Missing tools or libraries

Missing certificates

Wrong access rights

…

Cost of a Deployment

Complexity

Risks

Staffing

Time, On-Call Duty

Off-Hour Work

Down-Time

Stability

Our goals (yours might be different)

Improve Quality

Fewer errors, less (unexpected) down-time, less stress.

Standardization

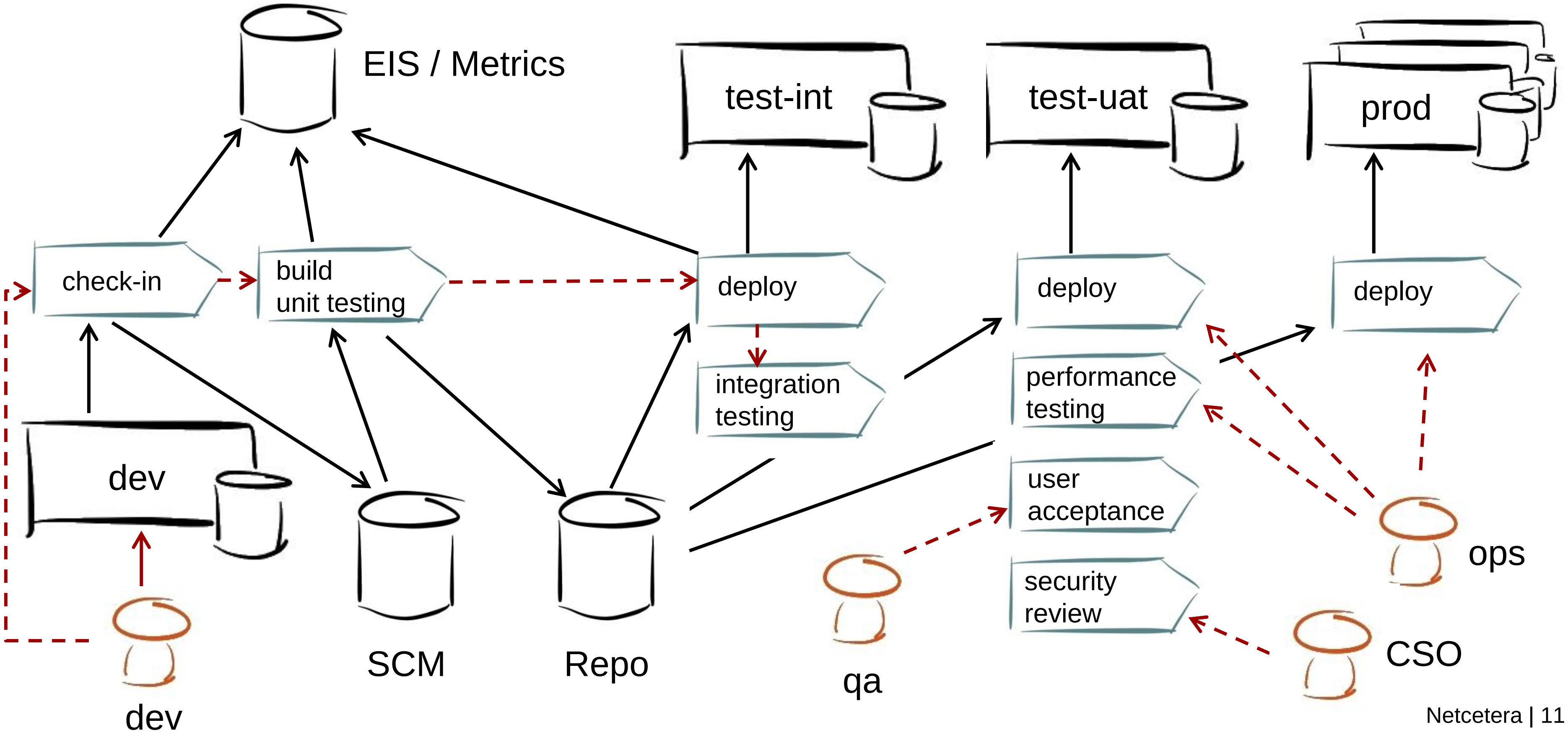
Different applications should deploy in a similar way.

Makes it easier for the operations and the development teams.

Faster Deployments

Less down-time, smaller need for off-hour deployments.

Software Delivery Pipeline



Software Delivery Pipeline

The Software Delivery Pipeline is the combined set of processes, procedures and tools that you use to bring code from the development environment into production.

Of course you would make the pipeline even wider by including requirements engineering etc. but for the context of this talk, we will start with code.

Software Delivery Pipeline

Builds Propagate through the Pipeline

Successful completion of a step triggers the next steps.

Errors interrupt the Pipeline

In case of an error, the pipeline processing is interrupted.

Expensive steps are only done if the previous steps worked

For example (manual) User Testing is only done on builds that passed the automated tests.

Some Pipeline steps might require manual triggering

Deployment to uat or prod is probably still triggered manually

Software Delivery Pipeline

A few things that you should notice right here:

Everybody has a Software Delivery Pipeline.

- You might not have thought about it yet, it might be automated or manual, very simple or complex, but you have a pipeline.
- Whether you want it or not, whether you manage it or not.

Your pipeline might look differently.

- Depends on the project, the customer, the business domain, the degree of maturity, …

Software Delivery Pipeline

Coming back to our risk and cost analysis, we can identify some problem areas:

Deployments are probably not tested

Deployments are done manually

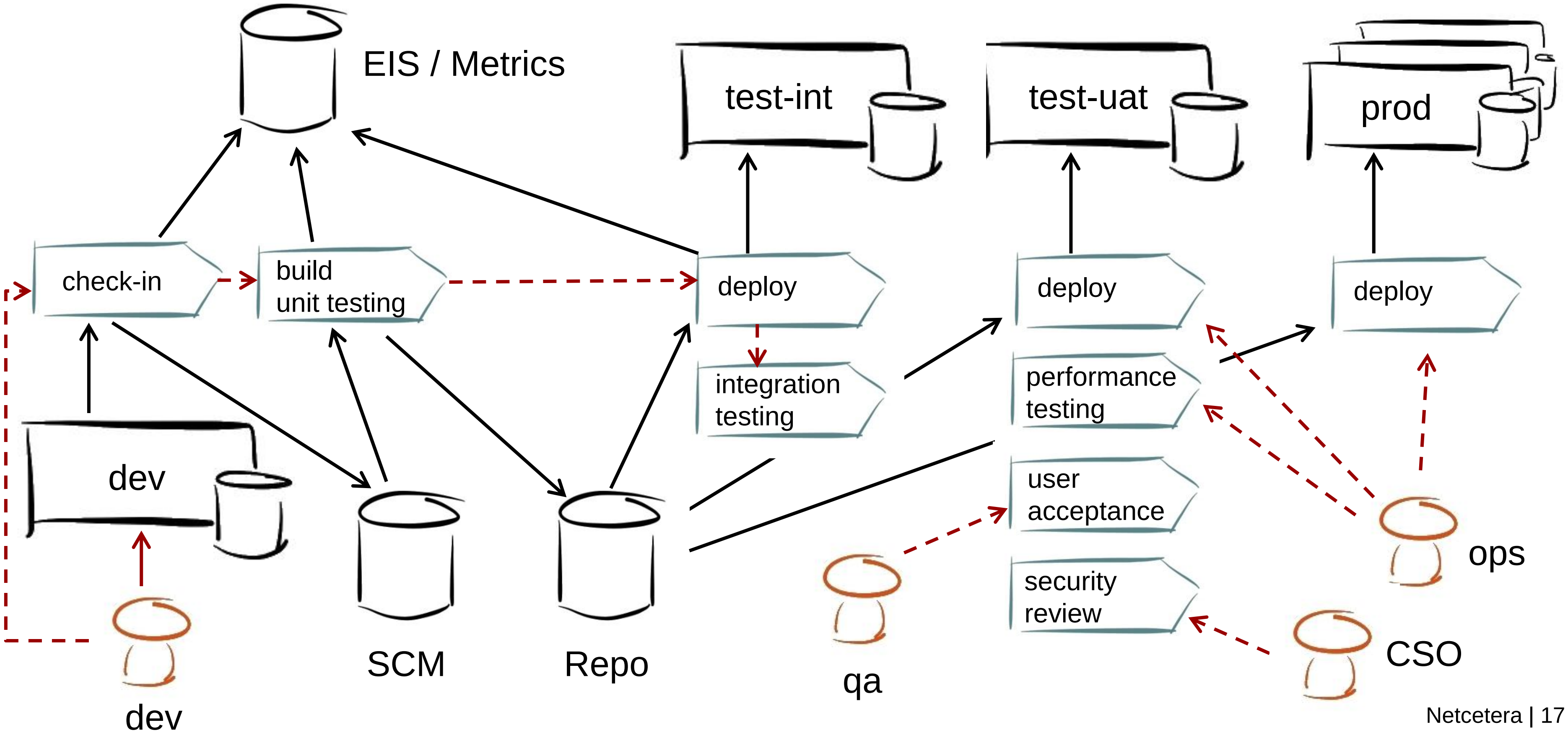
So again, the knee-jerk reaction would be to try to reduce the number of deployments. They are expensive, risky and mean a lot of stress for the development team, the customer and the operations people.

However, there are very similar issues on the left side of the diagram. We have some risky and expensive things there too (building, unit testing, deploying, integration testing, …). But the approach there is not too do it less frequently, but to automate it and do it more often. The underlying principle that we see there is …

If it hurts,

do it often.

Software Delivery Pipeline



Continuous Integration

What people did on the left side was to automate the pipeline steps and run them automatically and very frequently. Automation improves the quality by making the steps reproducible, predictable, verifiable, repeatable, …

This is essentially known as **Continuous Integration**.

A lot of companies already do this. But the automation often ends after the run of the integration tests and the creation of the build artifacts.

One way of looking at Continuous Delivery is to extend the principles from Continuous Integration and bring them further down the pipeline to the right side.

Continuous Delivery

Automation of Software Delivery Pipeline

Automation of the Deployment

Automated Triggering (Continuous Execution)

Automated Testing / Verification

Continuous Delivery

Automation

Automation of the delivery pipeline and the deployment process to the various environments. This includes deploying code, running database migrations, configuration, restarting, …

Automated Triggering (Continuous Execution)

Deployments (and subsequent steps) are triggered automatically, usually by upstream pipeline steps.

Automated Testing / Verification

Deployments are tested and verified by automated tests.

Automation of the Deployment Process

```
$ deploy <application> <version> <environment>
Deployment was successful
```

```
$ deploy <application> <version> <environment>
Deployment failed, rolled back to old version
```

Anatomy of a Deployment

- Notify the running application that a deployment is happing
- Set up a maintenance page
- Stop the running application
- Install the new application version
- Run a database migration
- Start the new application version
- Reconfigure the entry server
- Switch off the maintenance page

Anatomy of a Deployment

Consists of multiple consecutive steps

Steps are reusable for different deployments

Steps are reusable for different applications

Might vary from version to version

Might depend on the version that is currently installed

Anatomy of a Deployment Step

- **Verify the pre-conditions**
- **Do something**
- **Validate the post-conditions**

Pre-Conditions

Encoding of the assumptions of a deployment step.

Used to verify that the deployment step could actually work.

Make sure we do not attempt a deployment step, if the assumptions are not met.

Verification

Verification that a deployment step worked.

Verification that a deployment step had the desired effect.

Makes sure that we detect problems as early as possible in the deployment process.

Error Handling and Rollbacks

Ability to roll back at any time

Detecting errors

Automated Rollbacks

Manual Rollbacks

Error Handling and Rollbacks

Ability to roll back at any time

- Back up the preexisting state of the system
- Try to make the deployment non-destructive

Detecting errors

- Deployment process has to be able to detect errors

Automated Rollbacks

- Very desirable
- But also very complex
- Has to be tested (continuously)

Manual Rollback

It's just a workaround, but often a viable option

Probability of failing deployments is relatively small

- We are testing the deployment continuously after all.

Automated deployment process provides operations people with the necessary tools and artifacts to roll back easily

- Backups
- Non-destructive deployments

Data Migration

Data migration is a tricky thing

- Slow
- Complex
- Error-prone
- Stop-the-World

Automation and continuous testing helps

More frequent and therefore smaller deployments help

Data Migration Patterns

On-the-fly migration

Read-Only availability

Storage Layer abstraction

NoSQL storage layer

Data Migration Patterns

On-the-fly migration

- Application can deal with multiple versions of the database
- Data migration is done on-the-fly (batch job or touch-and-go)

Read-Only availability

- Database is read-only as long as the migration is on-going

Storage Layer abstraction

- Encapsulation of the storage layer

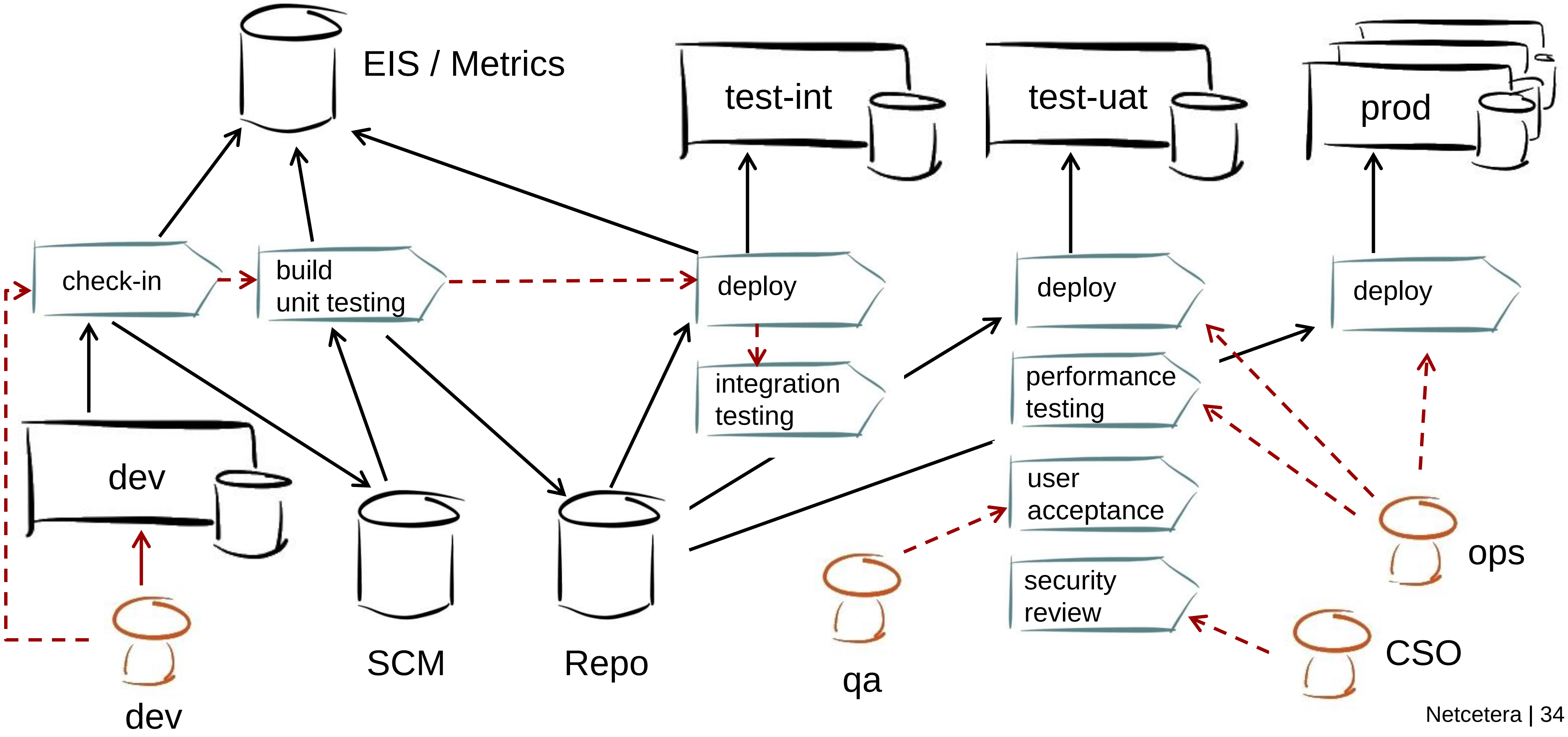
NoSQL storage layer

- Less stringent requirements on schema compliance

Deployment Testing

- Deployments are tested continuously
- Continuous Testing environment
  - Continuously, triggered automatically
- Pre-Production environment
  - Regularly, triggered manually
- "Same" configuration as the production environment
- "Same" state as the production environment

Software Delivery Pipeline



How do we get there?

It's a **process**

- Continuous delivery is not a binary thing.
- Continuous improvement of the maturity.

Forcing Teams

Just force the teams to do continuous delivery

Let them figure out how to do it

Bad idea!

Better Way

Provide Teams with Tools and Support

- Make it easy to implement continuous delivery

Iterate

- Do not try to solve everything in the first go
- Iterate and implement small steps
- Follow the pipeline down-stream

Standardization

Good way of **leveraging common requirements**

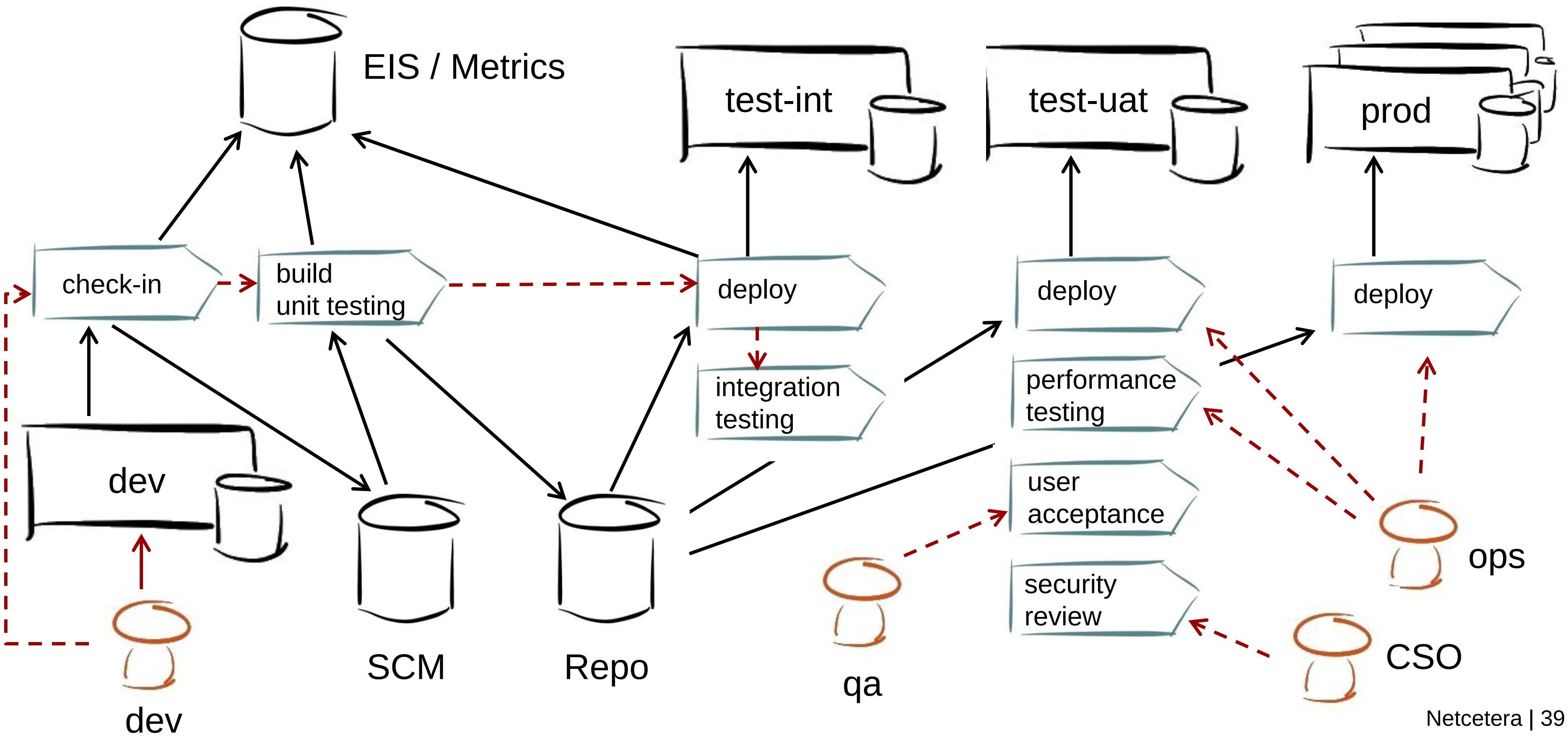
Provide building blocks for the implementation

- Common deployment steps
- Architecture, design and implementation **patterns**
- **Best practices** and **guidelines**

Make it **easy** for teams to follow the **standards**

Make it **possible** (but not too easy) to **extend** the standards

Software Delivery Pipeline



Abstractions for Runtime Environment

Common abstractions for deployment steps

Allow to abstract different runtime environments

OS versions and application server versions

Virtualization, Private Cloud, Public Cloud

Added Values for Deployments

Collect data about deployments

Verify compliance with SLAs

Access Control for deployments

Notify stakeholders on success/failure of a deployment

Configure monitoring system for deployments

Continuous Delivery and Agile Methodologies

Agile

Perfect match

Automation and automated tests
Continuous testing and verification
Short release and deployment cycles
Short feedback loop

Waterfall

Not really a match, but valuable anyway

Continuous Testing of Deployments
Automation of very complex processes
Delivery of hot-fixes etc.

Continuous Delivery

Software Delivery Pipeline Automation

Continuation of Continuous Integration

Continuous…

- … Builds, Integration, Testing
- … Deployment, Migration

Deployment Testing

Smaller and faster deployments

Reduction of Deployment Risks

Quality Improvement

Faster Deployments

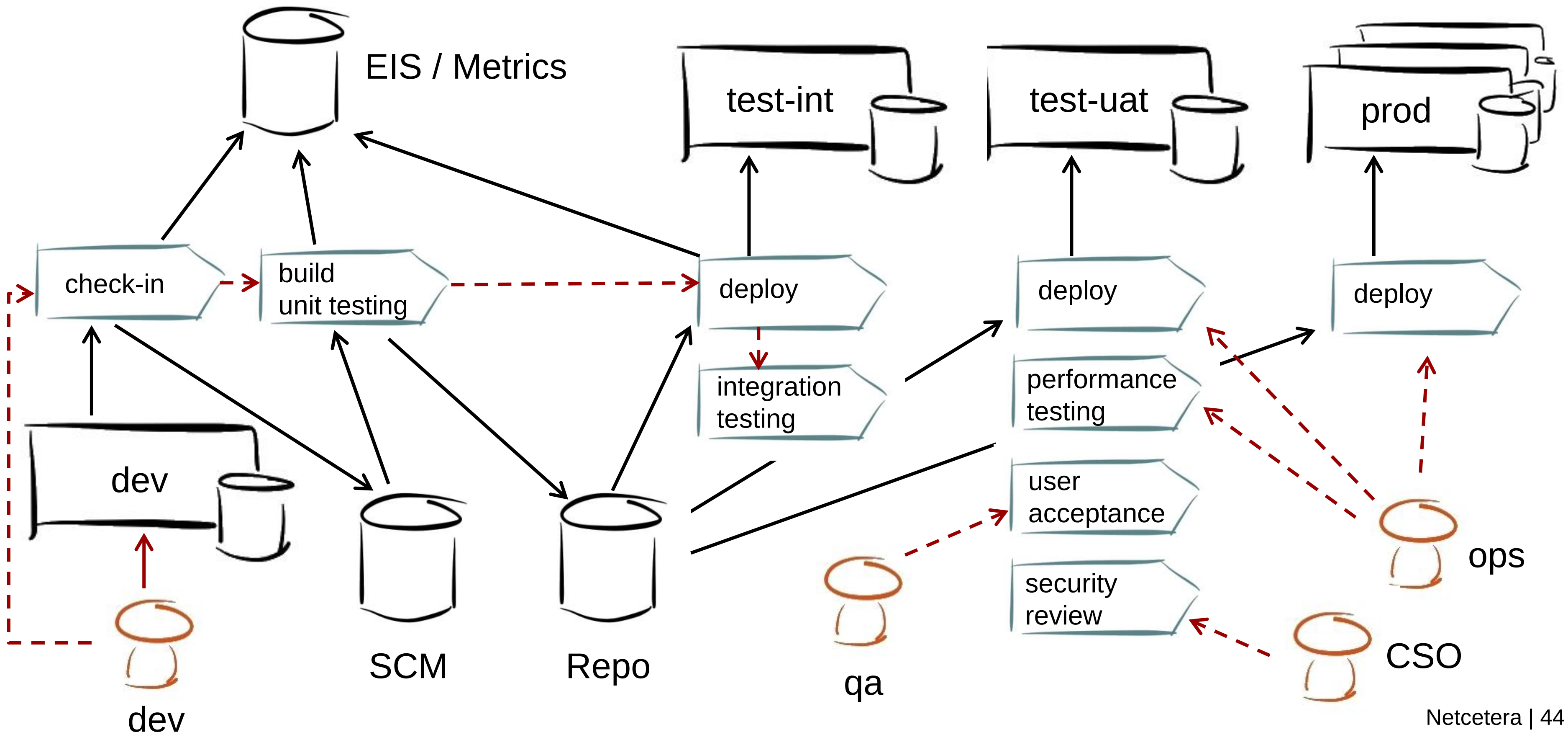
Down-Time minimalization

Shorter time-to-production

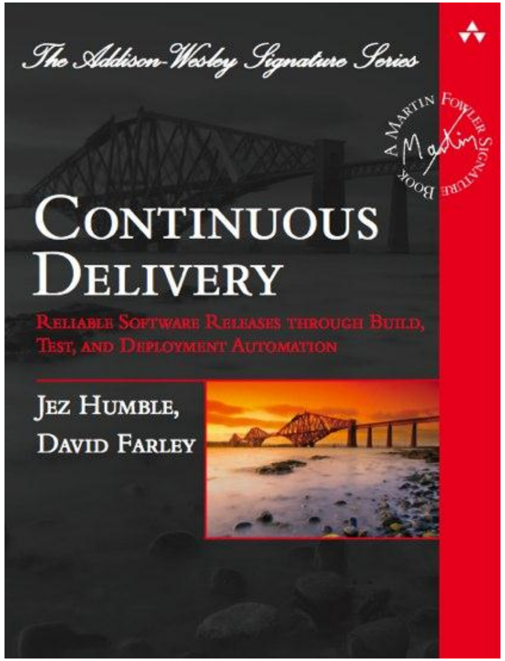
Less stressful deployments

Potential for more frequent deployments

Continuous Delivery



Further Reading


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble, David Farley

Continuous Delivery

Jez Humble, David Farley

Literally the book on "Continuous Delivery".

Further "Reading"

Continuous Delivery (Talk by Jez Humble, DevOps Day 2012)

http://www.infoq.com/presentations/Continuous-Delivery

Interview with Martin Fowler and Jez Humble on Continuous Delivery

http://www.infoq.com/interviews/jez-humble-martin-fowler-cd

Continuous Delivery (Blog by Jez Humble)
http://continuousdelivery.com/

Contact

Corsin Decurtins

corsin.decurtins@netcetera.com

+41 44 247 70 70