Continuous Practices

Daniel Ståhl
daniel.stahl@liu.se

LINKÖPING UNIVERSITY

Who am I?

DANIEL STÅHL

MSc 2007
Linköping University

Joined Ericsson in 2009

PhD, 2017
University of Groningen

Joined LiU in 2019

ERICSSON

Developer
Architect
Continuous Practices

…

AI Strategy
AI Governance
Research supervision

RESEARCH, WRITING

Continuous integration
Continuous delivery
Continuous deployment
Large-scale testing
Development practices
Teaming
Mob programming

Agenda

WHAT'S WHAT

CONTINUOUS INTEGRATION, DELIVERY AND DEPLOYMENT

WHY CONTEXT MATTERS

EXAMPLES OF CONTEXTS AND HOW THEY DIFFER

SOME FINAL REFLECTIONS

What's What?

CONTINUOUS INTEGRATION

A developer practice where developers integrate their work frequently […]

CONTINUOUS DELIVERY

A development practice where every change is treated as a potential release candidate [one is] able to deploy and/or release […]

CONTINUOUS DEPLOYMENT

An operations practice where release candidates [are] frequently and rapidly placed in a production environment […]

WHY DOES THIS MATTER?

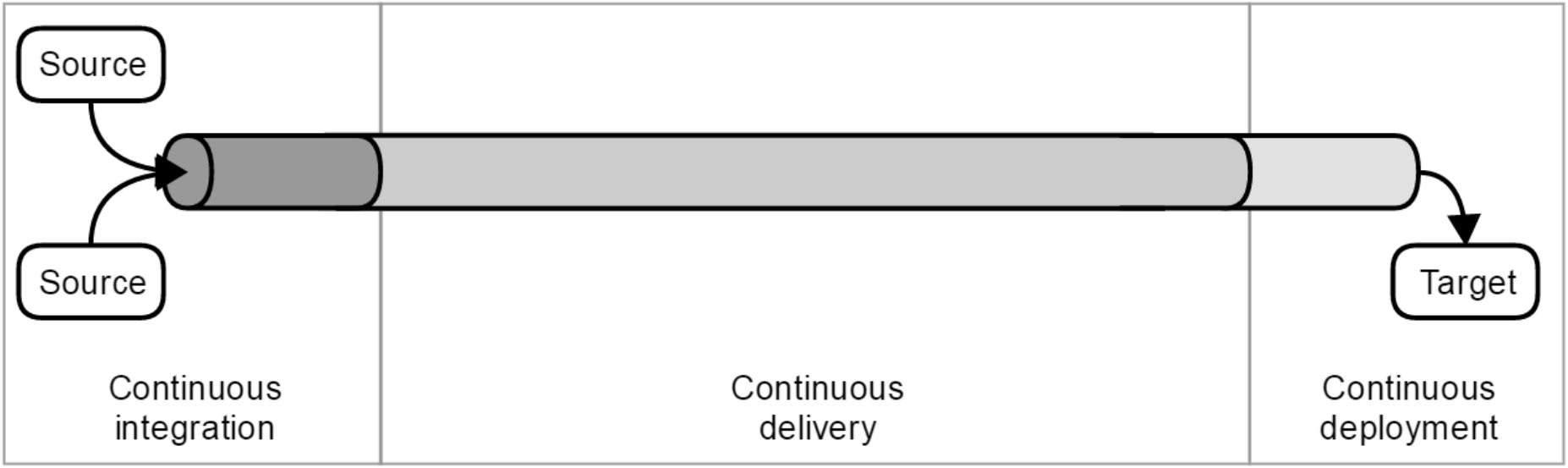
Semantics are important!

"Unlike genes, which are almost digitally coded, memes are often replicated with low fidelity as soon as they become a tad complex."

P. Kruchten 2007

Also, the exam…

Putting It All Together



Continuous Integration: Why Do We Do It?

IMPROVED QUALITY

Does it lead to improved quality? What do we mean by quality?

IMPROVED PREDICTABILITY

No big bangs means less fluctuation over time in quality. Less uncertainty, less integration overhead.

EFFICIENCY

Build and test automation can improve both efficiency and effectiveness, but are we really thinking of continuous *integration*?

SPEED

Everybody loves speed, but let's be mindful of the difference between speed and frequency. And speed of what? Time to market? Time to feedback…?

EASIER TROUBLESHOOTING

A smaller change delta makes it much easier to locate a fault!

TRANSPARENCY AND FEEDBACK

Power to the developers! But are we still not thinking of another practice?

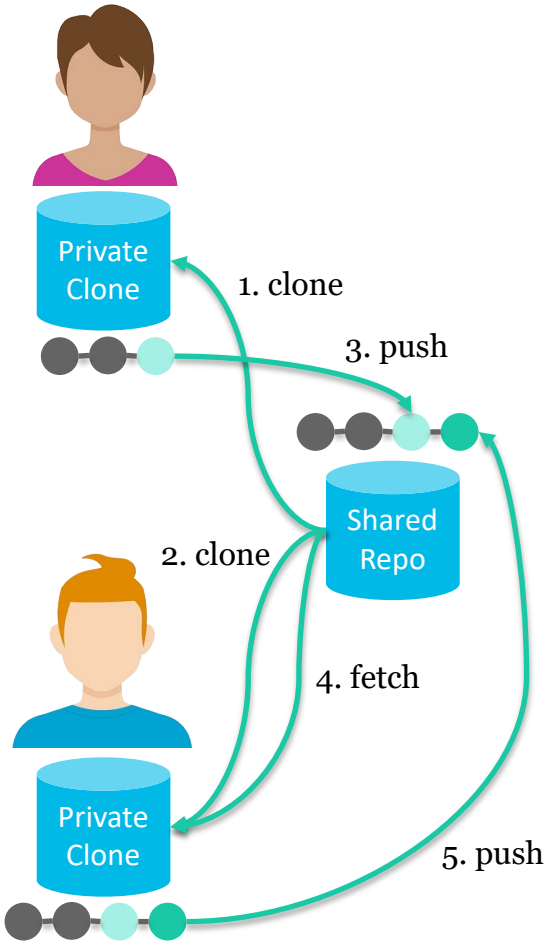
ENABLING CONTINUOUS DELIVERY

Continuous delivery pre-supposes continuous integration. Or does it?

IT FEELS GOOD?

Removes one potential source of developer anxiety.
Committing makes the code more "real". Doing "real" things is more fun!

Continuous Integration: What Does It Look Like?



Continuous Integration: Why Is It Hard?

SCALE

The more people, the more changes, the more merging.

CHOOSE YOUR POISON

Some constant pain, or very sharp pain every few weeks?

TEAM-INTERNAL INTEGRATION

Why not just integrate within our team? It seems much easier…

MANUFACTURABILITY

We speak very little in software engineering of how to design software that is easy to develop, test and integrate.

This is a crucial architectural concern!

Continuous Delivery: Why Do We Do It?

IMPROVED QUALITY

Assuming we trust our tests.
An automated pipeline is more consistent and reliable, but automated tests have weaknesses.

IMPROVED PREDICTABILITY

In theory, you always have a set of recent release candidates to choose from.

EFFICIENCY

Automated builds and tests are much cheaper, but not without CAPEX and OPEX.

SPEED

Yes! Let's still remember the difference between speed and frequency, though.
Simply being quick is easy, though…

EASIER TROUBLESHOOTING

The pipeline can provide lots of test data with very fine granularity.

TRANSPARENCY AND FEEDBACK

Yes! And remember there are numerous stakeholders with varying perspectives.

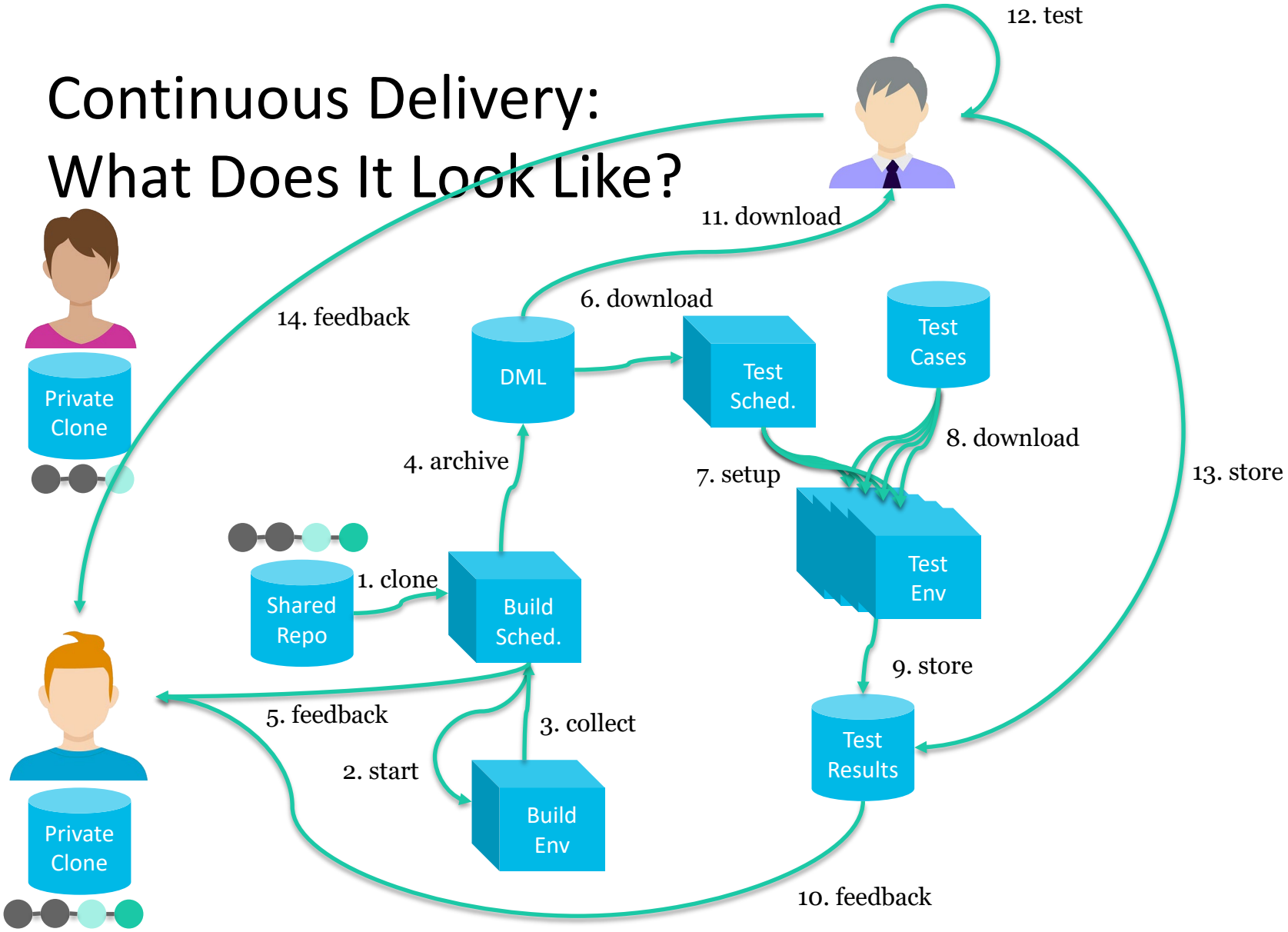
ENABLING CONTINUOUS DEPLOYMENT

Without continuous delivery, there can be no continuous deployment.

IT FEELS GOOD?

Successful tests, positive code analyses and green lights provide a sense of security.
Is it a false sense, though?

Continuous Delivery: What Does It Look Like?



Continuous Delivery: Why Is It Hard?

THE LAST 1% IS 90% OF THE EFFORT

Automatically creating an almost-release candidate is fairly easy. Creating an actual one with correct packaging, correct documentation, sufficient traceability and regulatory compliance is not.

WHAT DO I TEST?

There is limited capacity and time to test in the pipeline. There will always be more tests to run than you really have time for.

TEST FLAKINESS

Test flakiness is death. If you can't rely on your test cases, you can't rely on your release candidates.

It's not just false positives, but also false negatives!

ESTABLISHING TRUST

Switching to a heuristic way of thinking about release candidates is a huge mindset shift. Do not underestimate the change management aspect.

Continuous Deployment: Why Do We Do It?

IMPROVED QUALITY

Potentially, yes. Let us consider MTTR vs. MTTF. But this depends on context!

IMPROVED PREDICTABILITY

No more manual installation checklists, no more tweaked production environments.

EFFICIENCY

Automated deployments are much cheaper. Still, there's CAPEX and OPEX involved.

SPEED

Yes! Let's still remember the difference between speed and frequency, though.

EASIER TROUBLESHOOTING

Potential for very fine-grained in-production data.

TRANSPARENCY AND FEEDBACK

Yes! Particularly high potential for A/B testing, feature experimentation etc. Evidence based design decisions!

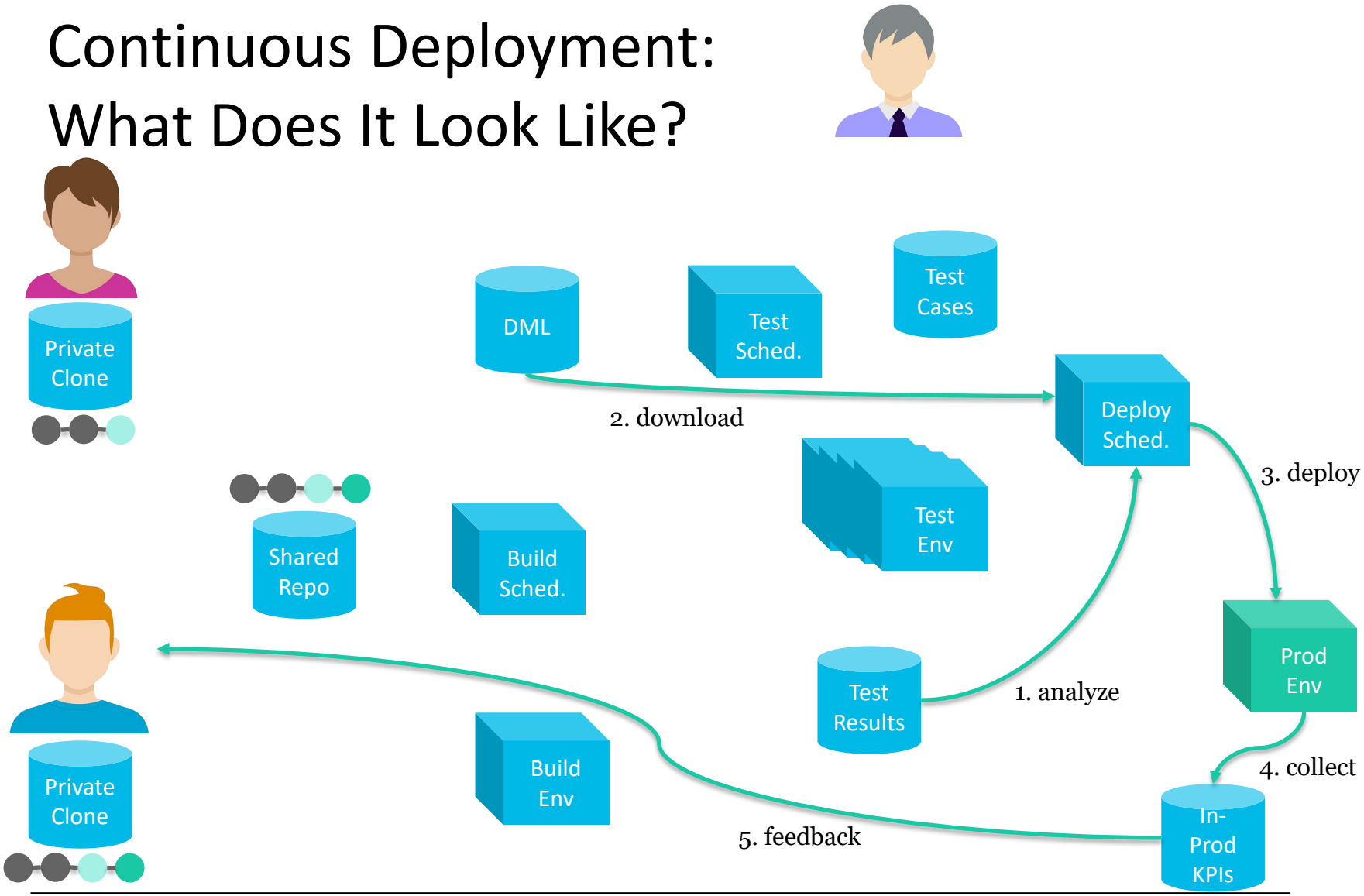
ENABLING DEVOPS

Automatically and continuously deploying turns operations into just another problem you solve by applying software to it.

IT FEELS GOOD?

Enable developers to deploy actual live software to users every day.

Continuous Deployment: What Does It Look Like?



Continuous Deployment: Why Is It Hard?

ENVIRONMENT CONTROL

Do you control the production environment? What is your distribution model? What do your customers and/or users think?

QUALITY IS PARAMOUNT

Without sufficient quality, don't even think about continuous deployment.

ZERO DOWN-TIME UPGRADES ARE HARD

How do you upgrade the software without stopping the service? Can be done, but is non-trivial.

MANAGING STATE

Upgrading a *stateless* service is doable, but what about a *stateful* one?

Context Matters!

CONTEXT ALWAYS MATTERS

A big part of being a software engineer is understanding your context, and understanding what you can get away with.

DIVERSITY

The software industry is very diverse. This puts very different requirements on software production systems.

ERR ON THE SIDE OF CAUTION

Do not assume that what once worked for you will work in a different context which you do not understand.

Contextual Factors

SAFETY CRITICALITY

What is the worst case scenario?

POWER BALANCE

What is the balance of power between you and your customers and/or consumers?

LEAD TIME

Is sooner always better?

PROXIMITY TO HARDWARE

Is your target environment Amazon Web Services or custom hardware you designed?

SCALE

Are there 2 or 2,000 developers writing the source code?

REGULATION

Are there regulations you need to stay compliant with?

DISTRIBUTION MODEL

How does your software reach its target environment?

Meet Jane
Defense Industry

Development of a new fighting vehicle

SAFETY CRITICALITY

Potential for injury and death.

SCALE

Hundreds of software engineers.
Many more hardware engineers and other professionals.

DISTRIBUTION MODEL

Software updates distributed and installed on physical media.
No over-the-air connection to installed base.

AND MORE…

Small number of large customers.
Tight coupling to hardware.
Highly regulated market.

Meet John
Gaming Industry

Development of a new computer game

SAFETY CRITICALITY

Inconvenienced and annoyed end users.

SCALE

Three developers.
Two content creators.
Two QA.

DISTRIBUTION MODEL

Downloaded by end user devices via online distribution system and marketplace.

AND MORE…

Large number of small customers.
Hardware agnostic.
Unregulated market.

A Conundrum and Some Advice

TO PUSH OR TO PULL?

How do you trigger actions in your pipeline?

EVERYTHING AS SOURCE

Version control not only your production code! Your environment definitions, your build scripts, your test code, your documentation…

CONTAINERIZATION

Containerization is not just for your product. Containerize your build and test environments!

LET HUMANS DO WHAT HUMANS DO BEST

Don’t think all testing can be automated (yet). Complement automated testing with e.g. exploratory testing.

FIGHT TEST FLAKINESS

Test flakiness is death. Having no test at all is better than having a flaky test.

DEFINITIVE MEDIA LIBRARY

Identify a single blessed archive for everything you produce. There must be no ambiguities.

MAKE COMMITTING SAFE AND EASY

If committing is scary or painful, developers will try to avoid it.

MAKE DEPLOYING SAFE AND EASY

If deploying is scary or painful, developers will try to avoid it.

Consequences of Continuous Practices

FEATURE EXPERIMENTATION

Continuous deployment enables feature experimentation.

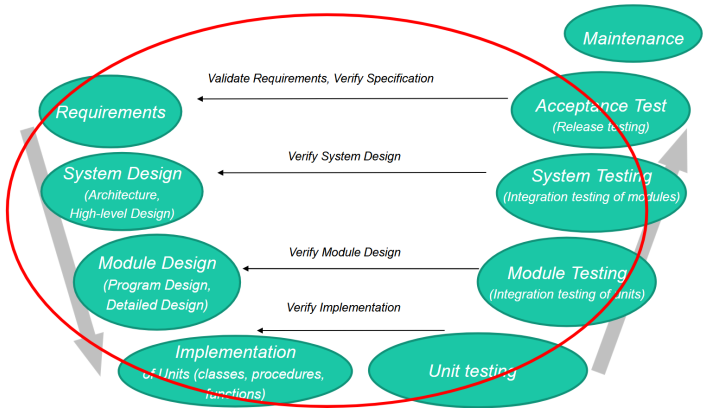
Feature experimentation enables extremely rapid hypothesis-experiment-evaluation loops.

Hypotheses may be wild guesses rather than based on thorough analysis.

Hypotheses may be phrased, tested and evaluated by a single curious developer in a single day.

WHAT ABOUT THE V-MODEL?

How does this relate to the V-model of software engineering?

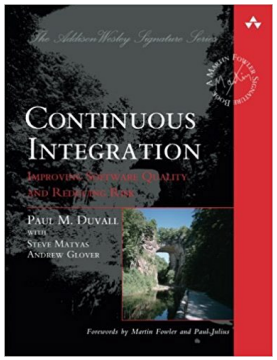


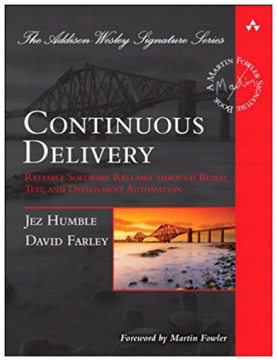
EVOLUTION OVER ENGINEERING

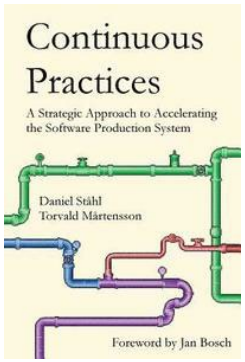
Software development becomes governed by KPIs, rather than requirements.

The end!

FURTHER READING







!?

Daniel Ståhl

www.liu.se

LINKÖPING UNIVERSITY