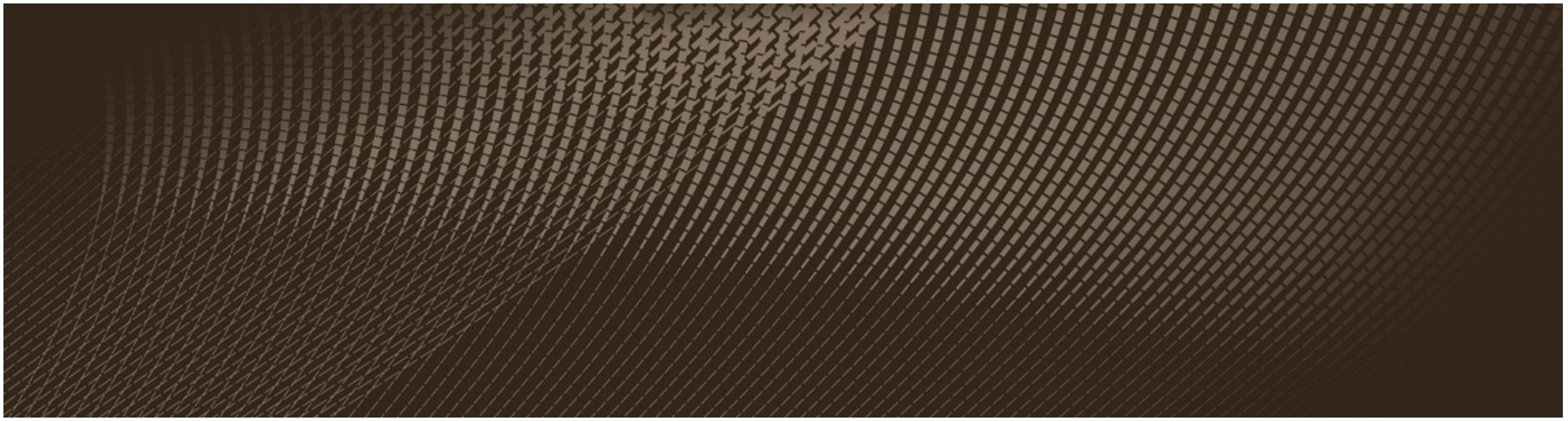


Continuous Delivery

Ole Christian Rynning & Stein Inge Morisbak

Roots 2011

Agenda

- Who are we, where do we come from and what do we work on?
- What does Continuous Delivery *really* mean?
- Collaboration, Version Control and Continuous Integration.
- Controlled deployment
- Zero-down-time redeploy (and roll-back).
- Robust applications (architecture)
- Provisioning of Infrastructure for Continuous Delivery.
- Error handling, Monitoring and health verification.
- Migrating databases.
- Summary, organizational impacts and business value.

Stein Inge Morisbak

digipost

- Information Science, University of Bergen
- 10 years +
  - Operator (≈ 2)
  - Consultant (≈ 5)
  - NorgesGruppen Data (≈ 6)
- Many roles:
  - Developer
  - Operator
  - Tech lead
  - Architect
  - Agile/Tech coach
  - Manager

Digipost

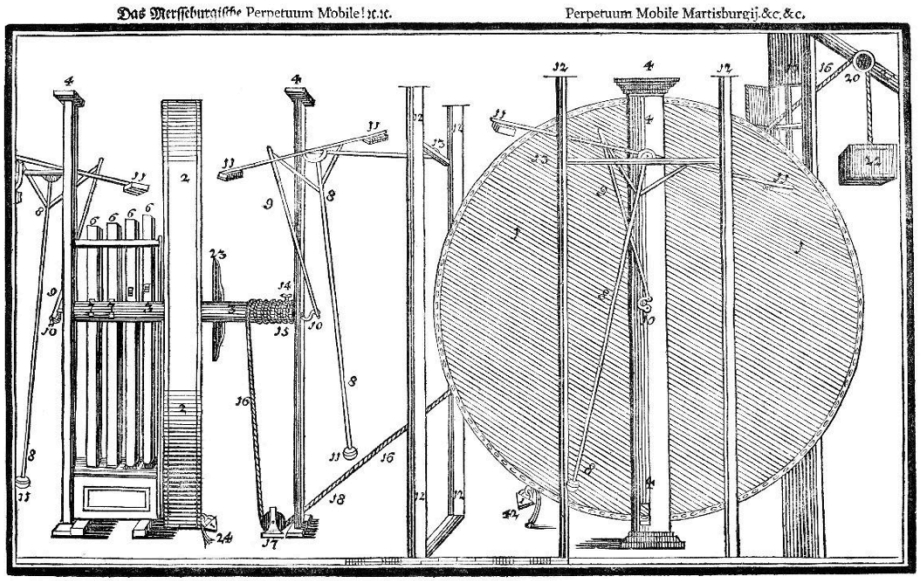
- National service for digital mail.
- Replacement for physical box.
- Posten Norge AS.
- Launched april 4th.
- Java.
- 13 developers on the team.
- I'm tech lead.
- Agile process.
- Self organizing.
- Continuous Delivery.
- Deployment of new features every week.

Ole Christian Rynning

- Master IT, UiO
- 5 years +
  - Consultant (≈ 4)
- Many roles:
  - Developer
  - UI Developer
  - Operator
  - Tech lead
  - Architect
  - Creative Leader

Bring Development Team

- Bring
  - Business side (B2B) of Norway Post
  - Consists of 8 specialists (Express, Parcels, Cargo, Frigo, Mail, …)
- Several projects
  - Tracking (public, B2B, APIs, mobile applications) (6 rels 2011)
  - Mybring (reports, calculators, APIs) (2 rels 2011)
  - Fraktguide (B2B, B2C, price quoting service) (2 rels 2011)
  - Booking (Order products across all specialists) (42 rels 2011)
- Java & Ruby
- 10 developers, 2 UX, 1/3 operators, 4 product managers, …
- Post-Agile process
- Delivers Continuously *



What does Continuous Delivery *really* mean?

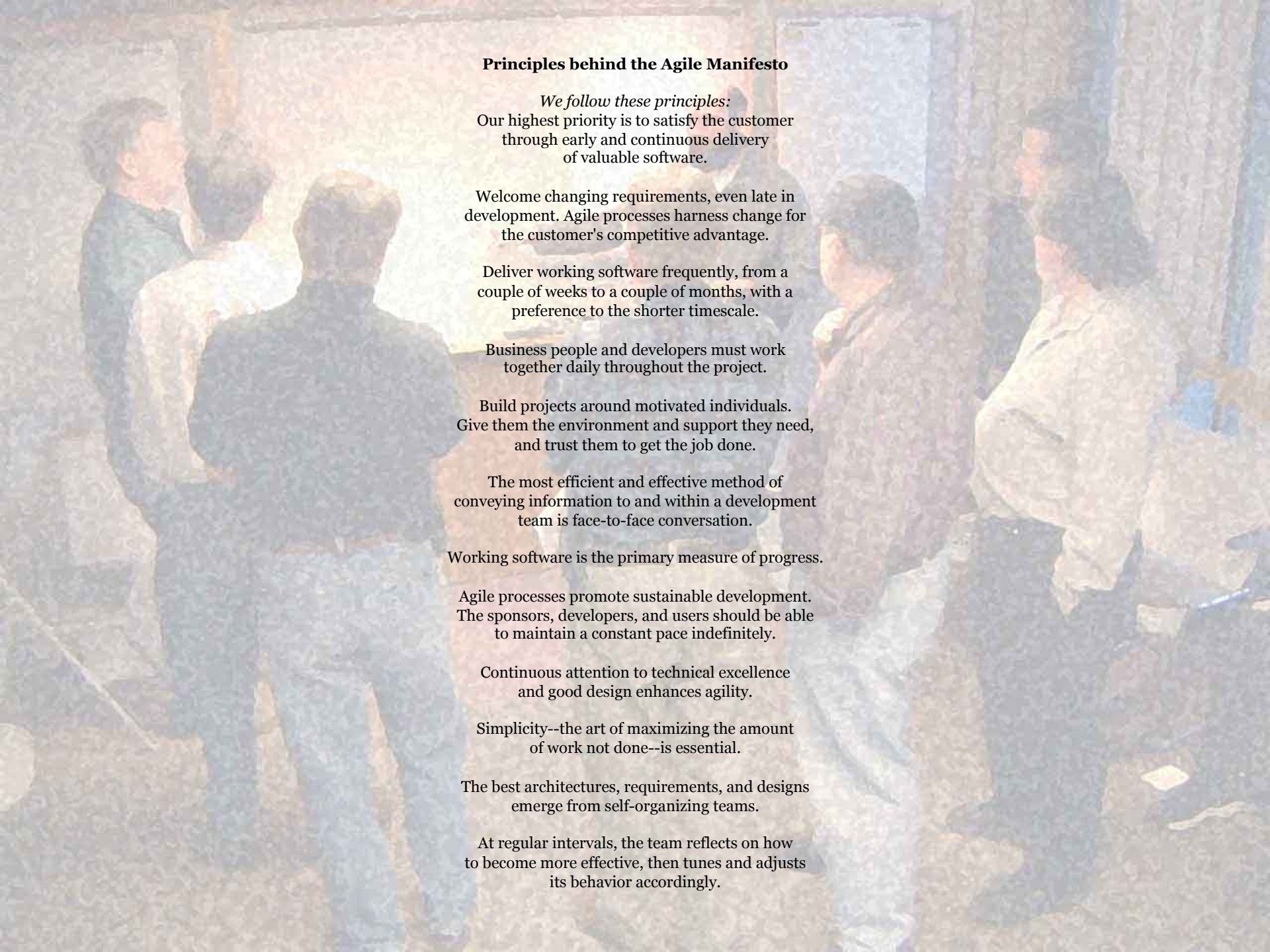
Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas



Principles behind the Agile Manifesto

We follow these principles:
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

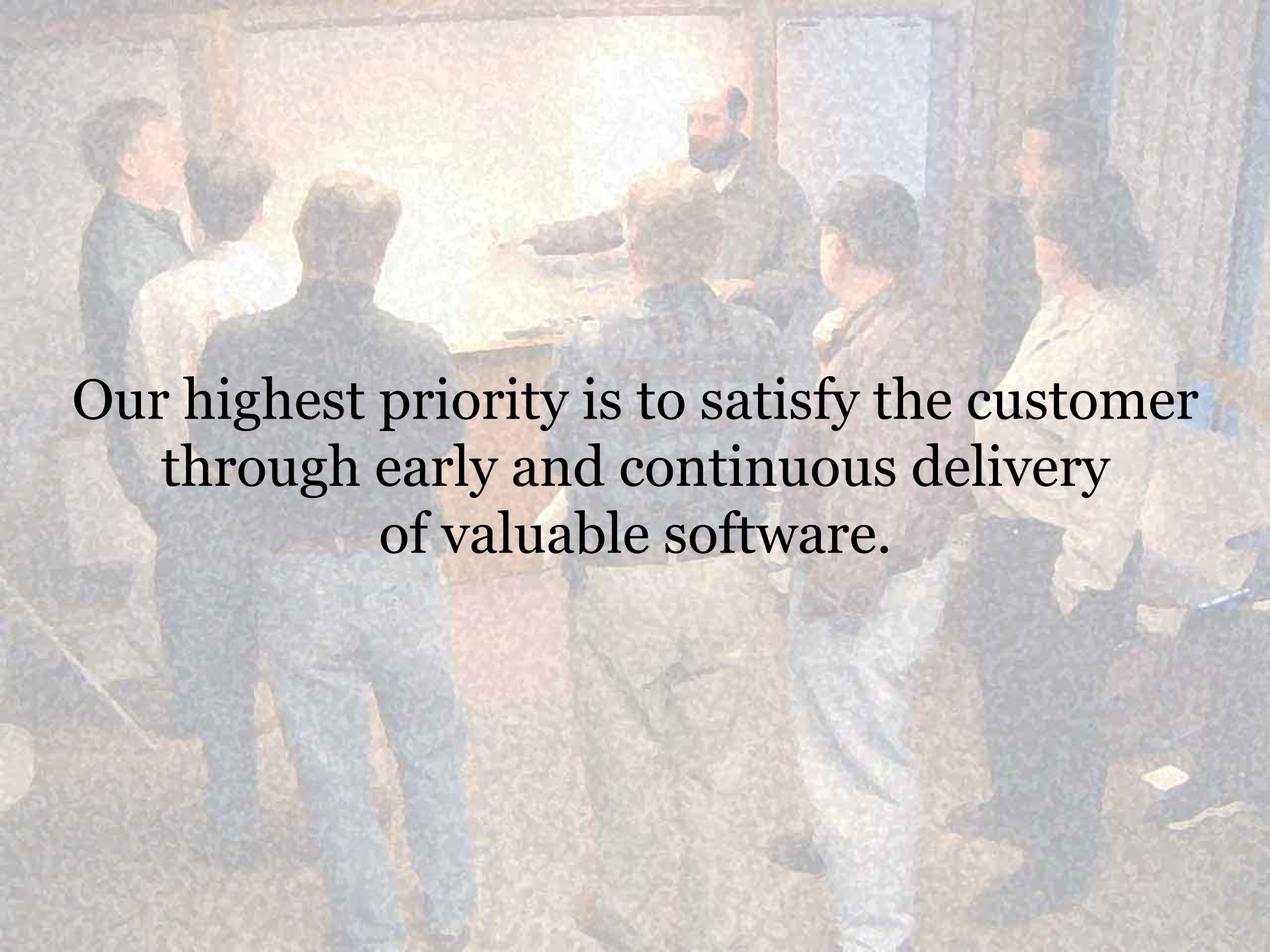
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.


Our highest priority is to satisfy the customer
through early and continuous delivery
of valuable software.

What is Continuous Delivery?

Continuous delivery is about putting the release schedule in the hands of the business, not in the hands of IT.

Implementing continuous delivery means making sure your software is always production ready throughout its entire lifecycle – that any build could potentially be released to users at the touch of a button using a fully automated process in a matter of seconds or minutes.

- Jez Humble (http://continuousdelivery.com/)

The Big Picture

It's all about Business Value

- Reduce Operational Expenditure (OPEX)
  - Reduce manual bottlenecks
  - Reduce overhead in communication (bust **dev-qa-ops** silos)
  - Increased control over software's life cycles
- Reduce Capital Expenditure (CAPEX)
  - Increase/optimize utilization (i.e. virtualization)
  - Reduce startup costs
- Increase Customer Satisfaction (Protection)
  - Improve usability, performance, security, requirements
  - Respond quicker to customer demands
- Increase brand value (Revenue)

Questions you need to answer

- Are you able to put new features into production within a reasonable amount of time (i. e. less than a day)?
- You don´t have that requirement?
- What about bug-fixes?
- Would your customer be happier if she made a decision and saw it in production the same day?
- Would your customer be happier if you could deploy without down-time?
- Would you trust your deployment process more if you did it more often?
- Would you feel more safe if you deployed fewer features to production at a time?
- Would you feel more safe if you had less things that could go wrong?
- Would you be happy with a heavy manual deployment process if you where to release several times a week?
- Would operations be happier (and everyone else safer) if deployment was automated instead of documented.
- Would you be happier (and not so lonely) if you could deploy during work hours instead of in the middle of the night?
- Are you able to roll-back (alternatively roll forward) if deployment fails?

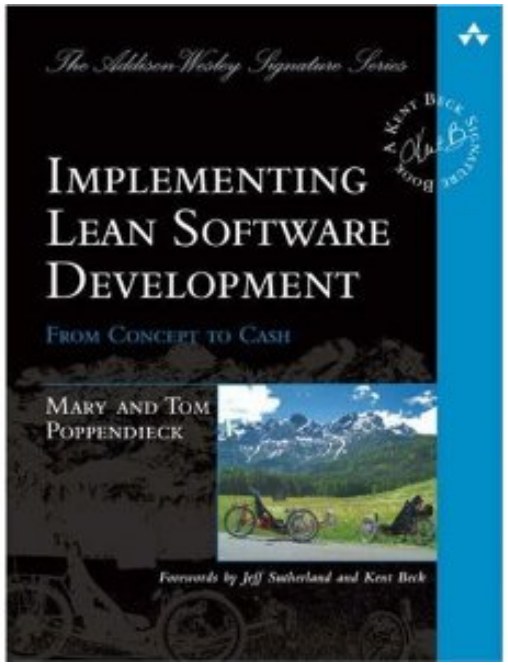
Lean for Software Development

The Challenge

How long would it take your organization to deploy a change that involves just one single line of code?

Do you do this on a repeatable reliable basis?

- Mary and Tom Poppendieck



Conway's law

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

- Melvin Conway, 1968

www.melconway.com/research/committees.html

An Agile team

Continuous Delivery team

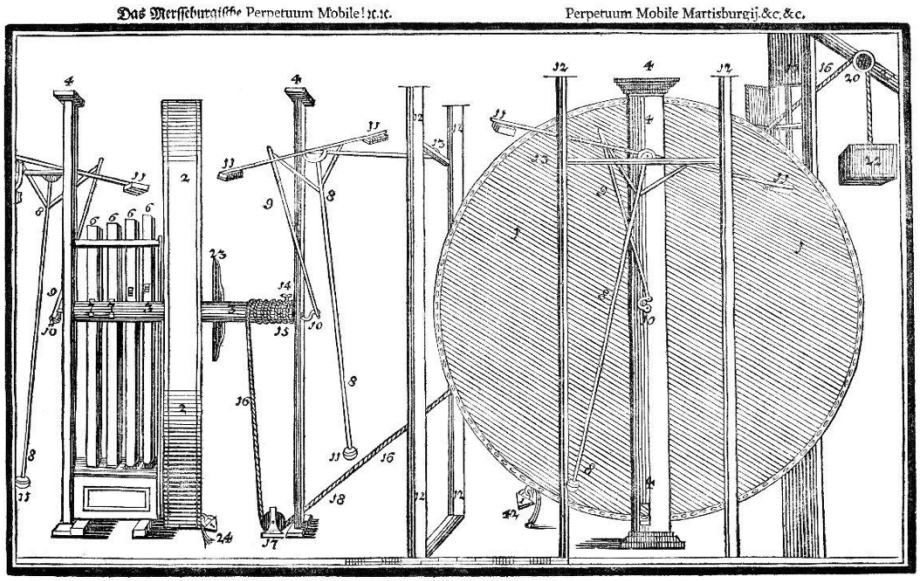
- Trust and collaboration.
- Transparent (big visible displays, demos and retrospects).
- Releases driven by business needs and fast feedback.
- Fearless (config. mgmt., automated testing at all levels, CI)
- Disciplined (don’t break/bend the rules)
- Self sufficient and cross-functional (devs, customers, testers, ops, dbas)
- Rehearses deployment and roll back.
- Everybody is responsible for delivery.
- Everyone can self-service deployments.
- Continuously improving automation speed (fast build etc.)
- **The team is continuously improving.**

Continuous Improvement

- A self organizing team is like evolution, only with a shorter time span.

In biological systems self-organization is a process in which pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern. (Wikipedia)

- You must improve!
  - Process.
  - Build quality in (don't waste time on mass inspection).
- You must become better!
  - Skills.
- You must learn!
  - Try things you don't know the outcome from.
  - Fail fast.
  - If it hurts, do it more often, and bring the pain forward.



Robust Applications / Architecture

Open Source == Less bugs

But! That is not my stack!

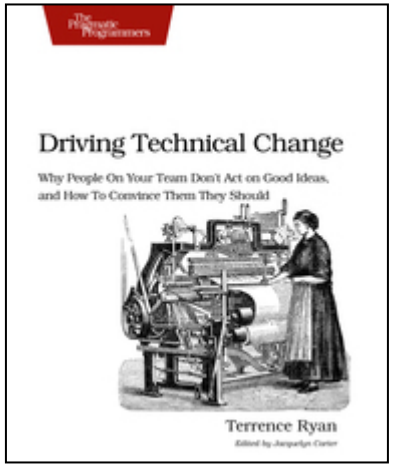
- Not all technologies are easy to work with when implementing Continuous Delivery.
- What stack are you (forced to be) using?
- Know each dependency in any stack
  - Why is it there?
  - What does it do?
  - Why the version?

Nightmare Architects

- Ivory Tower Architects
  - Strives towards higher levels of abstractions portable across platforms and systems. (Believes there is a single “the Solution” to architecture).
  - Loves boxes and straight arrows.
  - May code frameworks that future projects have to use*
  - Usually won’t listen to neither developers nor users / customers.
  - More detrimental to any Enterprise than they will ever realize.
- Tell-tales:
  - “Use JSF on all web-applications”.
  - “You haven’t used all eight layers that our guidelines prescribe”.
  - “OK. As long as it is Oracle/IBM”.
  - “Everything has to go through the service bus”.
  - “All applications should follow <title of 500 page guideline document>”.

Ah well. I agree but…
How to handle transition?

- Gain expertise (you're here, right).
- Picking up new technologies or tools is work.
- Prove it!
- Know how to handle sceptics.
- Look for win-win situations and synergy effects.
- Tear down silos. (Communication, collaboration, connectivity).
- Make it compelling and get publicity.
- Improve your soft skills.
- Go around them (NB!)



- Ignore the Irrational
- Target the Willing
- Harness the Converted
- Convince Management

My kind of Architect

- Pragmatic Architects
  - Codes or are at least highly able to jump into code.
  - Uses foresight, but truly believes that architectures are grown (incrementally) and constantly thinks about the dynamic nature of changing architecture.
  - Accepts that there is no “one solution”.
  - Does not hesitate to change an abstraction if it does not fit.
- Tell-tales:
  - “The ESB is too slow for handling this load, use the endpoint directly”.
  - “Have you considered a SoftReference there in order to not OOM?”.
  - “Set a Socket Timeout on that endpoint.”.
  - “We use <dependency> <version> because of <esoteric patch>”.
  - “How will the servers handle the load if one node dies?”.

Be Cynical with architecture.

- Whatever you do not test against, **WILL** happen.
- Assume all Integration Points will go down, congest, return corrupt data, incomplete data, be slow, …
- Assume all low-level connections (pools, concurrency, …) will at sometime fail.
- Assume your application will die.
- Know your application's life cycle, and failure modes
- Keep environments as equal as possible.
  - Attack environment-based configuration with vigor.

Choose automatable Technology

- Use an embedded container. (Executable jar with war)
- No need to deploy, just start and stop.
- Trivial to setup.
- Full control over the JVM.
- Easy to automate.
- Easy to post-mortem (kill -3 <pid>)
- Some containers that work:
  - Jetty
  - Netty / Grizzly
  - Glassfish (at least before Oracle)
- See: https://github.com/bekkopen/jetty-pkg

Understand Application/ Failure modes

- Traditional (UNIX) application Life cycles
  - start, stop, status, reload
  - Use pidfiles, locks
- Application-critical life cycles
  - graceful-stop/drain, set-read-only
- Support inspection of application state (status)
  - health-check, session-status, build version, active feature toggles, configuration
- Support quick inspection of failed applications
  - thread-dump (kill -3), backup-db, log-backup

Understand (and automate) Application Life Cycles

- Understand logging concerns
  - Operation/application status events (errors) - AUTOMATE
  - Business events/Audit events (warn)
  - Debugging events (info, debug, …)
- Keep configuration minimal
  - Separate environment and application configuration*
  - Keep environment configuration to an absolute minimum
- Know your hardware limitations
  - If one of four balanced nodes die, and the nodes are 50% utilized. The load on a single node will be significant (66% utilized). If another dies the system is on the edge.

Longevity: Impulse and Strain

- Impulses (Stress)
  - Christmas
  - Getting Slashdotted
  - Annual settlement adding 100 million transactions to a queue
  - Counter: Heavy load tests, Fallback switches
- Strain (Longevity)
  - Memory leakage and consumption
  - Data growth
  - Dying connections / low level
  - Counter: Timeouts, Fallbacks

Continuous Integration

- Run against develop branch.
  - Clone builds for long-lived branches.
- Automate testing (against a production like environment).
  - Unit tests, integration tests, system tests, performance tests, security tests, (functional acceptance tests)…
- Keep the build fast.
  - Unit tests, integration tests, slow tests.
- Deploy the latest executable to a repository (Nexus, Artifactory)
- Visibility (Lava lamps, monitors, air strike alarm…, ~~e-mail~~)
- Automate deployment.
- See: http://martinfowler.com/articles/continuousIntegration.html

Safety measures (Continuously run)

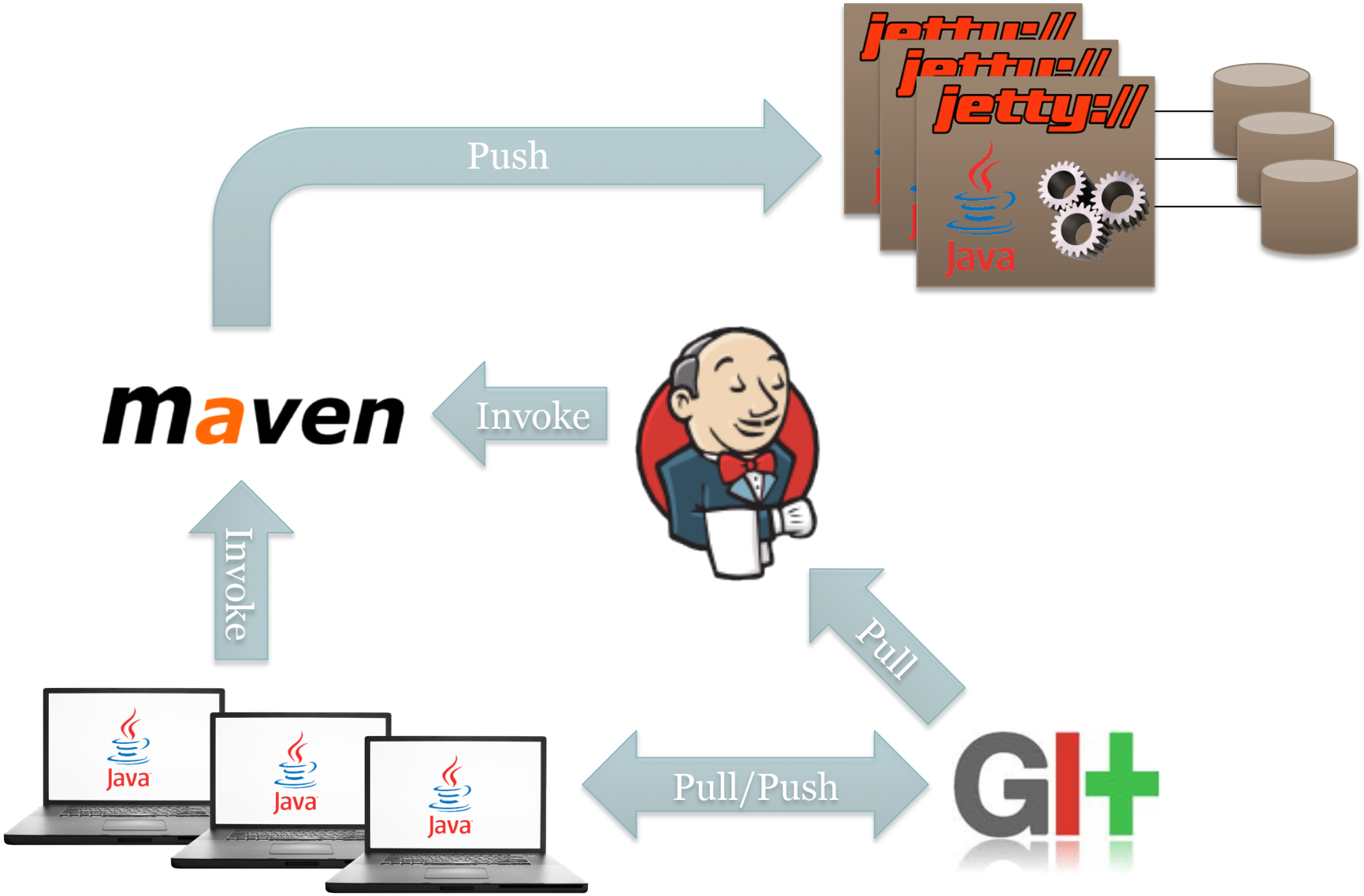
- Fail fast
- Build
- Metrics
- System tests
- Environment tests
- Integration tests
- Application tests

Metrics

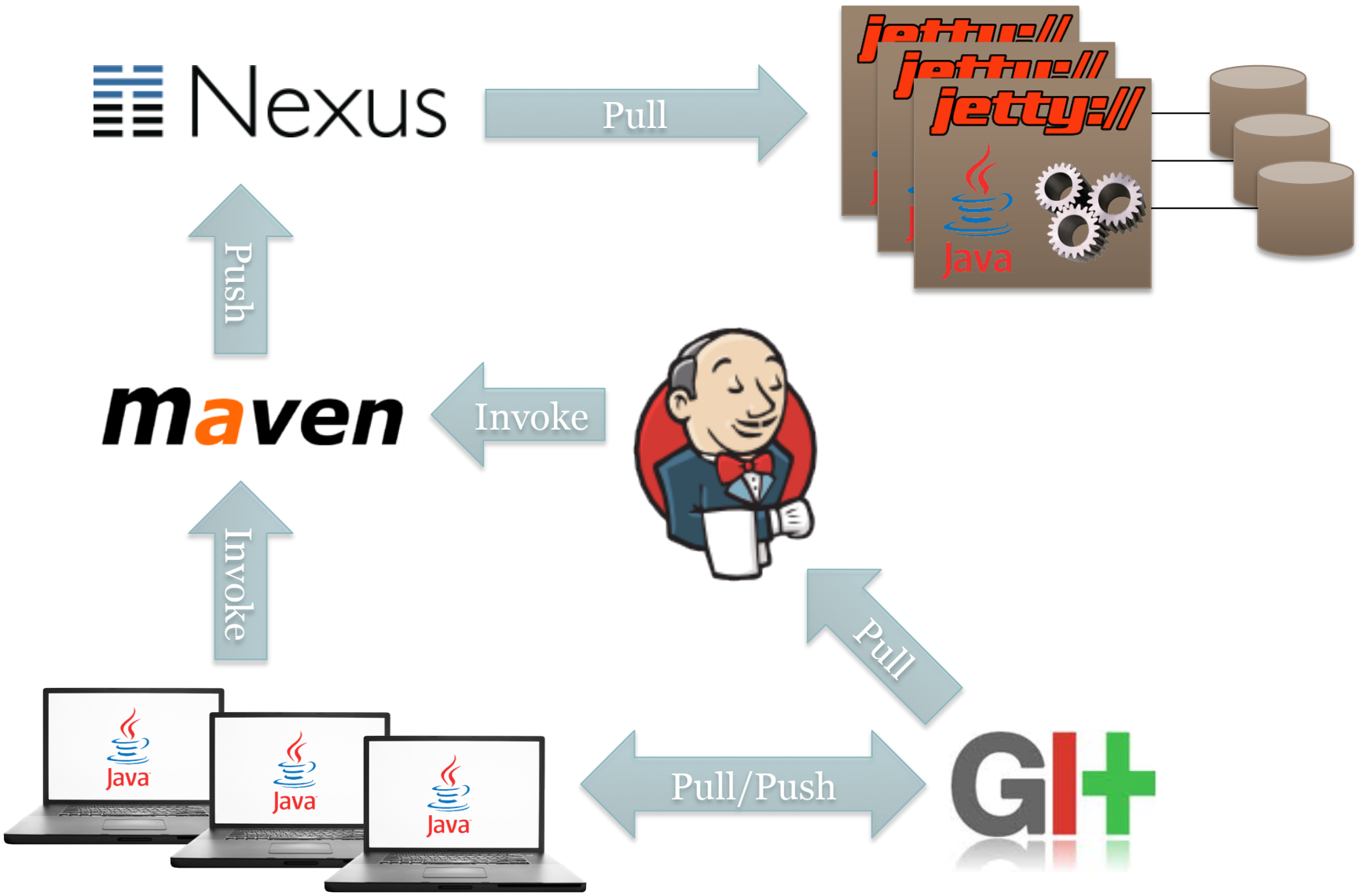
- What do you want to measure?
  - Test coverage?
  - Lines of code?
  - Code metrics?
  - Checkstyle violations?
- Do you automate metrics and create reports (e. g. Sonar)?
- How often do you look at those reports?

Alternative: Automate and break the build if you violate your rules.

Continuous Deployment (Push)



Continuous Deployment (Pull)



When to push and when to pull?

- Push:
  - Allows anyone to push.
  - Automated.
  - Caller configures server.
  - Easier?
- Pull:
  - Initiated by authorized caller.
  - Same artifact is pulled to different environments.
  - Server configures itself.
  - Safer?
- Vague difference:
  - Push -> Pull: Server requests a push.
  - Pull -> Push: Ssh and issue pull command.

Version Control in Teams

What is Version Control?

1. A repository of content.
2. Access to historical editions of each datum.
3. A log of all changes.

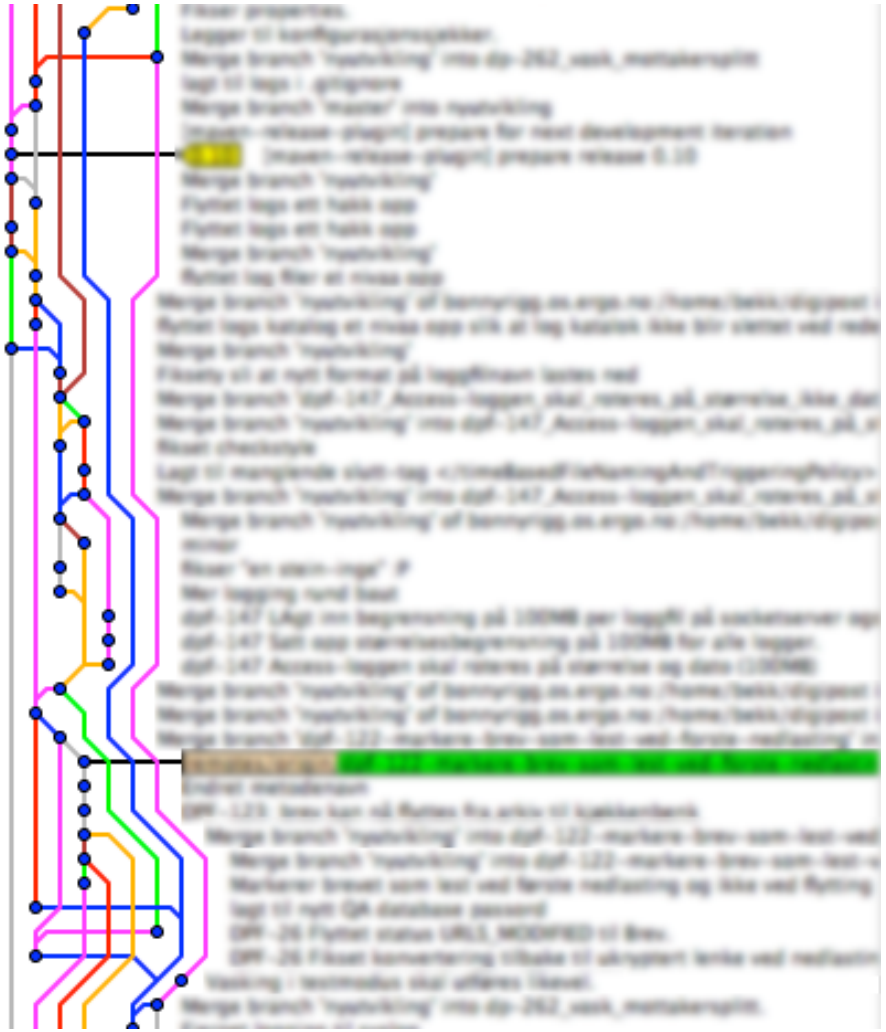
A tool that manages and tracks different versions of software or other content.

Version Control for Continuous Delivery

- Keep (almost) *everything* in version control!
- WiP must be separated from code in production.
- Hot-fixes must happen on code in production.
- Release branches prevent freezes and delays.
- Avoid big scary merges.

From this ...

...to this



Work in Progress (WiP)

F1

F2

F3

F4

F5

F6

F7

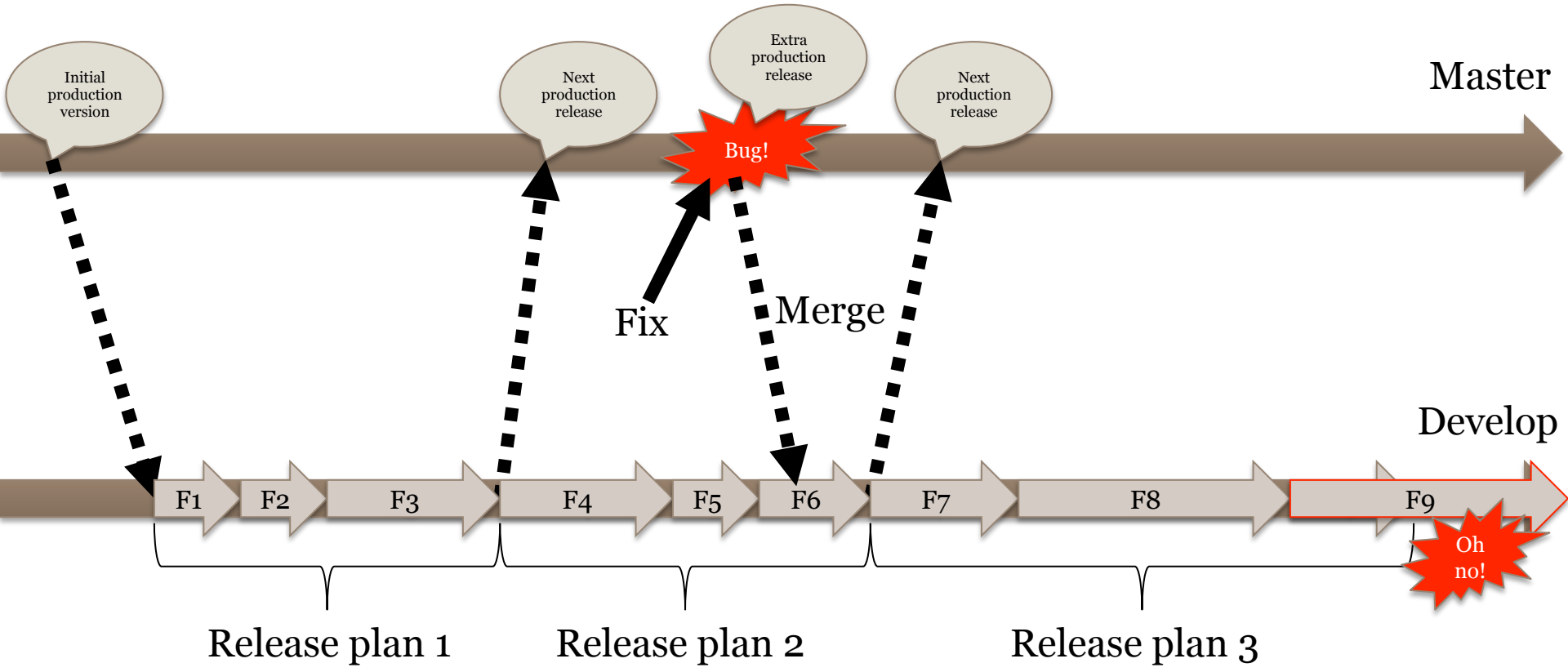
F8

F9

F10

Time

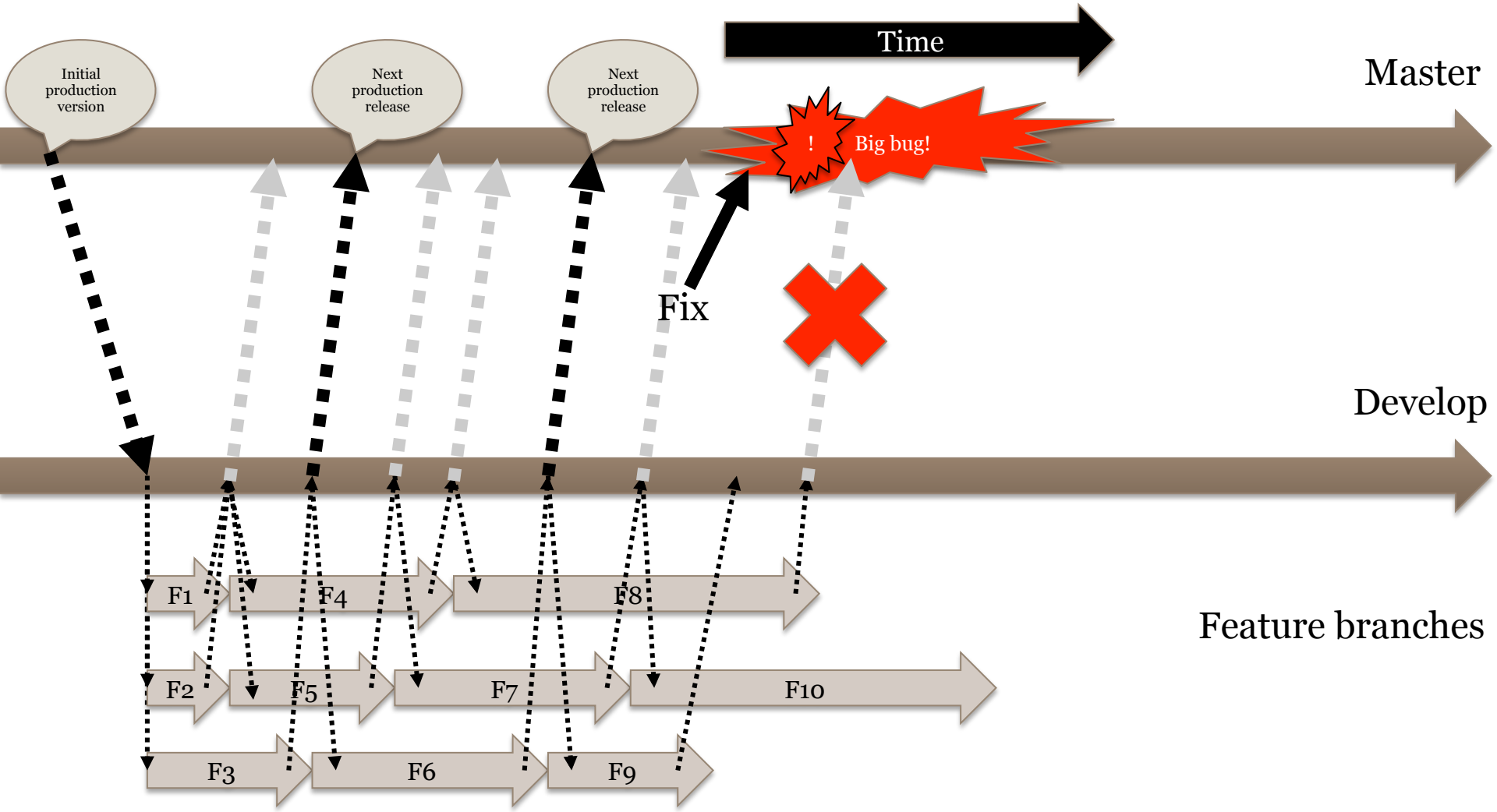
We have to separate WiP from production ready code



Git – Create development branch

```
$ git checkout -b develop
Switched to a new branch ”develop"
```

We have to detach features from the release plans



Git – Feature branches

Create:

```
$ git checkout -b <id>_<description>
Switched to a new branch "<id>_<description>”
```

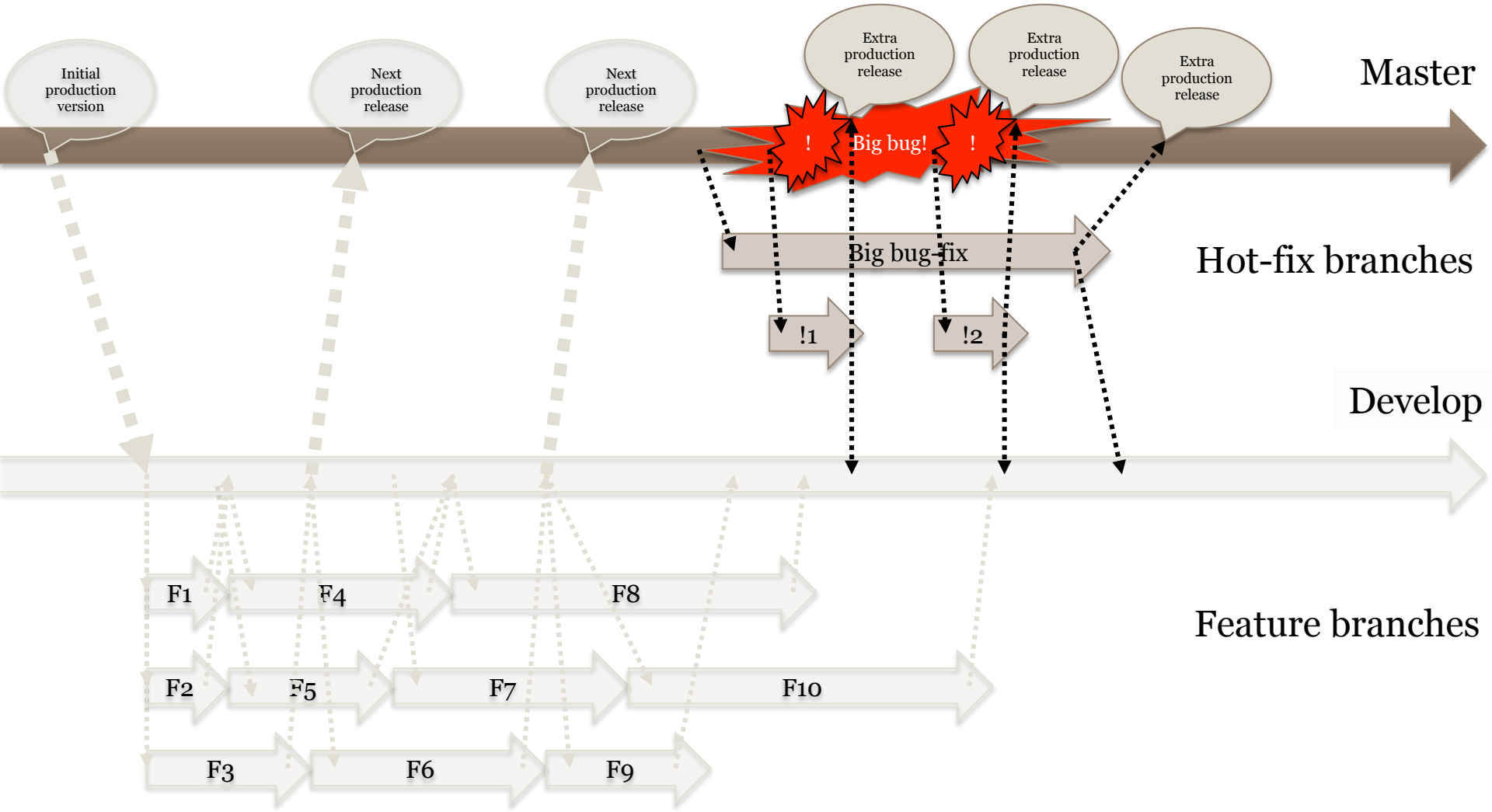
Merge:

```
$ git checkout develop
Switched to branch ’develop'
$ git merge --no-ff <id>_<description>
Updating ea1b82a..05e9557
(Summary of changes)
$ git push origin develop
```

Delete:

```
$ git branch -d <id>_<description>
Deleted branch <id>_<description>
```

We have to detach hot-fixes from production and develop



Git – Hot-fix branches

Create:

```
$ git checkout -b hot-fix-<version>-<id>_<description>
Switched to a new branch "hot-fix-<version>-<id>_<description>”
```

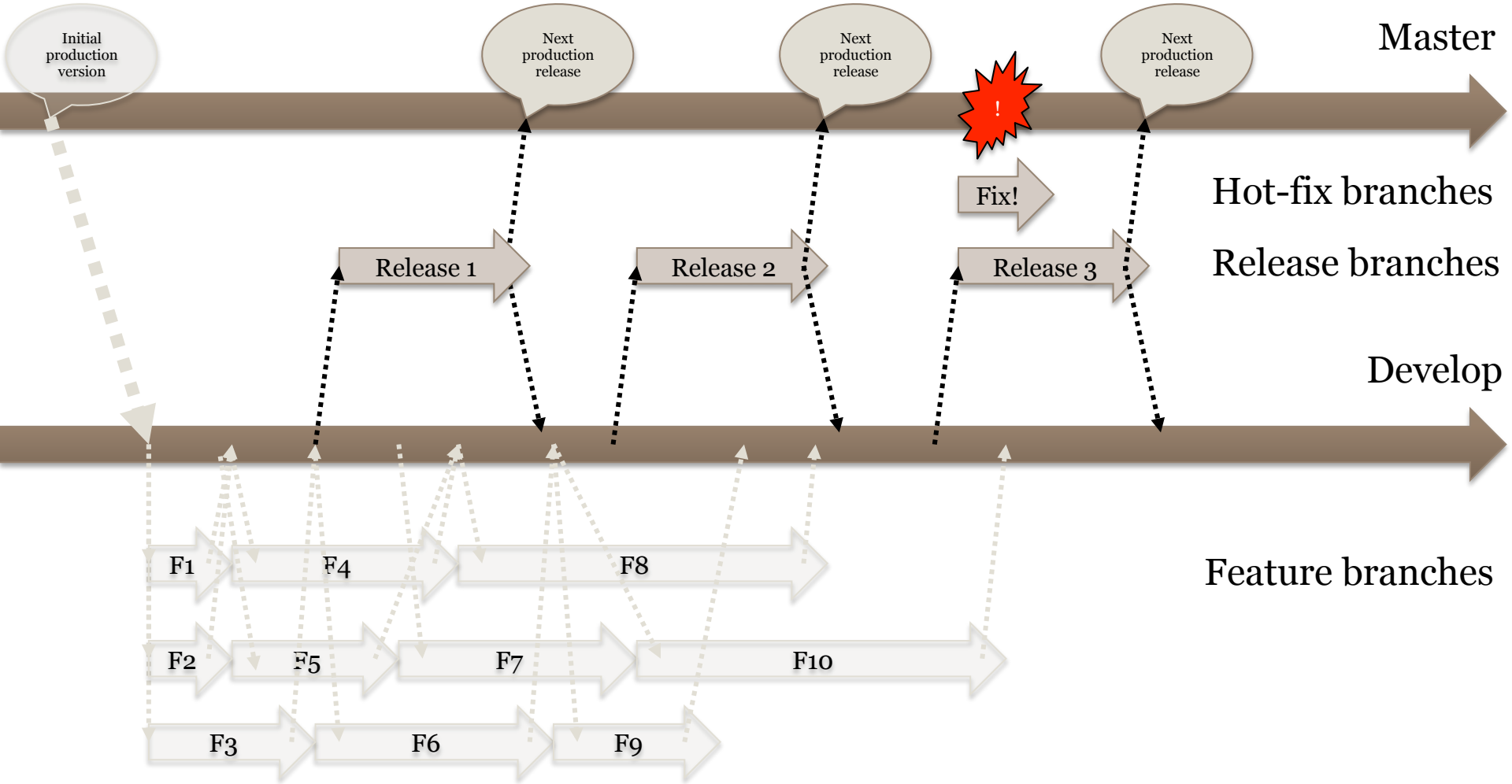
Commit:

```
git commit -m ”<message>"
[hot-fix-<version>-<id>_<description> abbe5d6] <message>
5 files changed, 32 insertions(+), 17 deletions(-)
```

Git – Hot-fix branches -> merge and deploy

```
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff hot-fix-<version>-<id>_<description>
Merge made by recursive.
(Summary of changes)

$ mvn clean release:prepare
$ mvn clean release:perform -Dgoals=deploy
$ mvn clean deploy

$ git checkout develop
Switched to branch ’develop'
$ git merge --no-ff hot-fix-<version>-<id>_<description>
Merge made by recursive.
(Summary of changes)

$ git branch -d hot-fix-<version>-<id>_<description>
Deleted branch hot-fix-<version>-<id>_<description> (was ff452fe).
```

Code freezes, testing, preparation ... - Release branches



Git – release branches

Create:

```
$ git checkout -b release-<version> develop
Switched to a new branch "release-<version>”
```

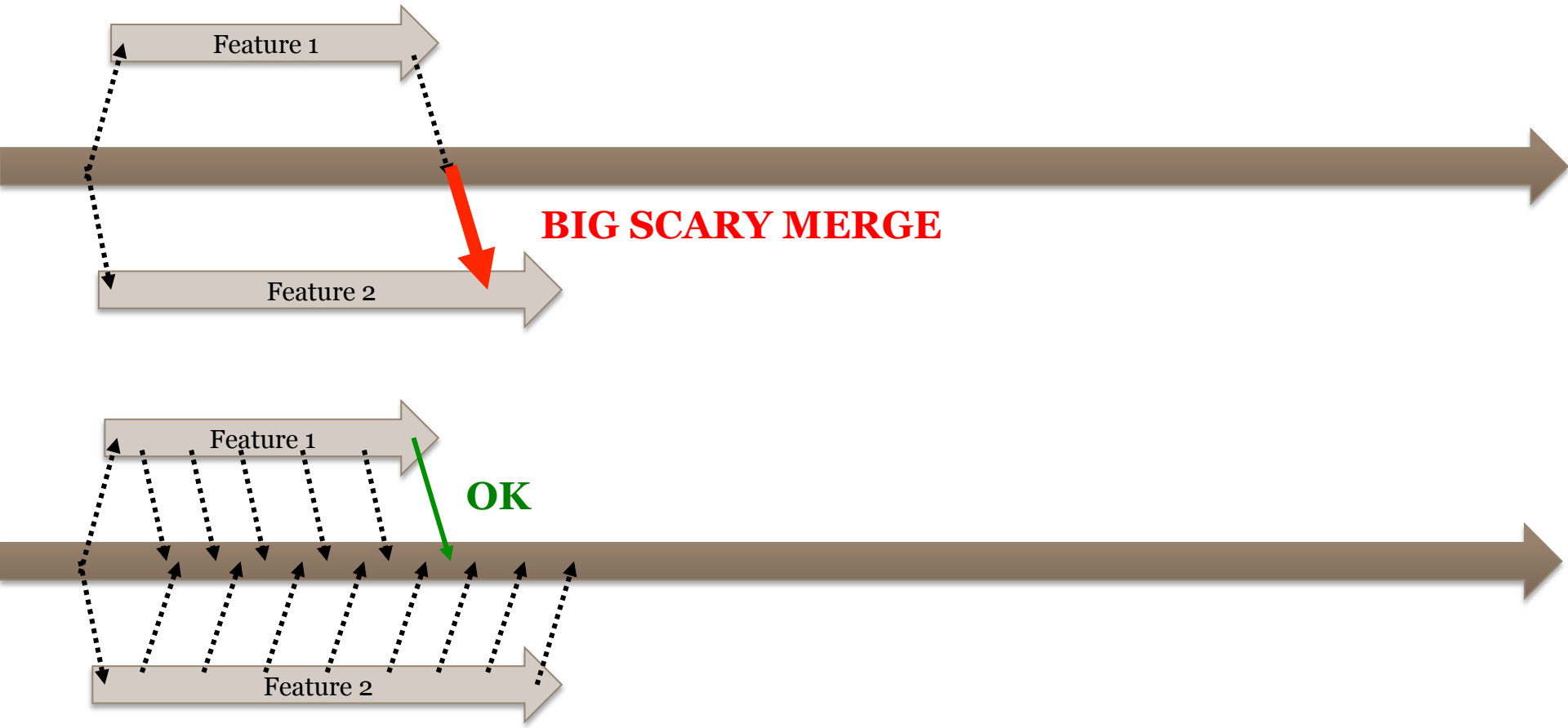
Commit:

```
git commit -m ”<message>"
[release-<version> abbe5d6] <message>
5 files changed, 32 insertions(+), 17 deletions(-)
```

Git – release branches -> merge and deploy

```
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff release-<version>
Merge made by recursive.
(Summary of changes)

$ mvn clean release:prepare
$ mvn clean release:perform -Dgoals=deploy
$ mvn clean deploy

$ git checkout develop
Switched to branch ’develop'
$ git merge --no-ff release-<version>
Merge made by recursive.
(Summary of changes)

$ git branch -d release-<version>
Deleted branch release-<version> (was ff452fe).
```

Feature branching vs. Continuous Integration



Avoiding the Big Scary Merge

- The problem with continuous integration:
  - Potentially unfinished features in mainline.
  - Hence you can not deploy whenever you want.
- The Digipost project: 

  - 13 developers.
  - uses feature branching and releases to production every week.
  - have never had serious merge problems.
- **Keep your features small!**
- **Improve the features iteratively.**
- You can check in unfinished features (!)
  - As long as it doesn't compromise the rest of the system.
  - Use feature toggles.

Feature toggles

feature_toggles.properties

```
mail.enabled=true
sms.enabled=false
```

send_message.jsp

```
<toggle name=mail.enabled>
        . mail UI elements
</toggle>
```

SmsService.java

```
...
        boolean smsEnabled;
        if (smsEnabled) {
                sendSms();
        }
...
```

- Have a common strategy/framework.
- Configuration in one place.
- Toggle in as few places as possible.
- Avoid polluting your code.

Error handling, Monitoring and health verfication.

GET /application/health

```
bring-shipment-number           [OK]
bring-messagebroker             [OK]
bring-freightguide-postal-code  [OK]
nets-epayment                   [OK]
bring-freightguide              [OK]
bring-label-generator           [OK]
bring-label-repository          [OK]

Version: 1.4.4
Build:   0b33b727908
Node:    <censored>.bd.bring.no
```

How?

```
package no.bring.booking.operations;

public interface HealthCheckable {

    Health healthCheck();

}
```

Sample VO

```
package no.bring.booking.operations;

public class Health {
    final String name;
    final Boolean status;

    public Health(String name, Boolean status) {
        this.name = name;
        this.status = status;
    }
}
```

Example implementation

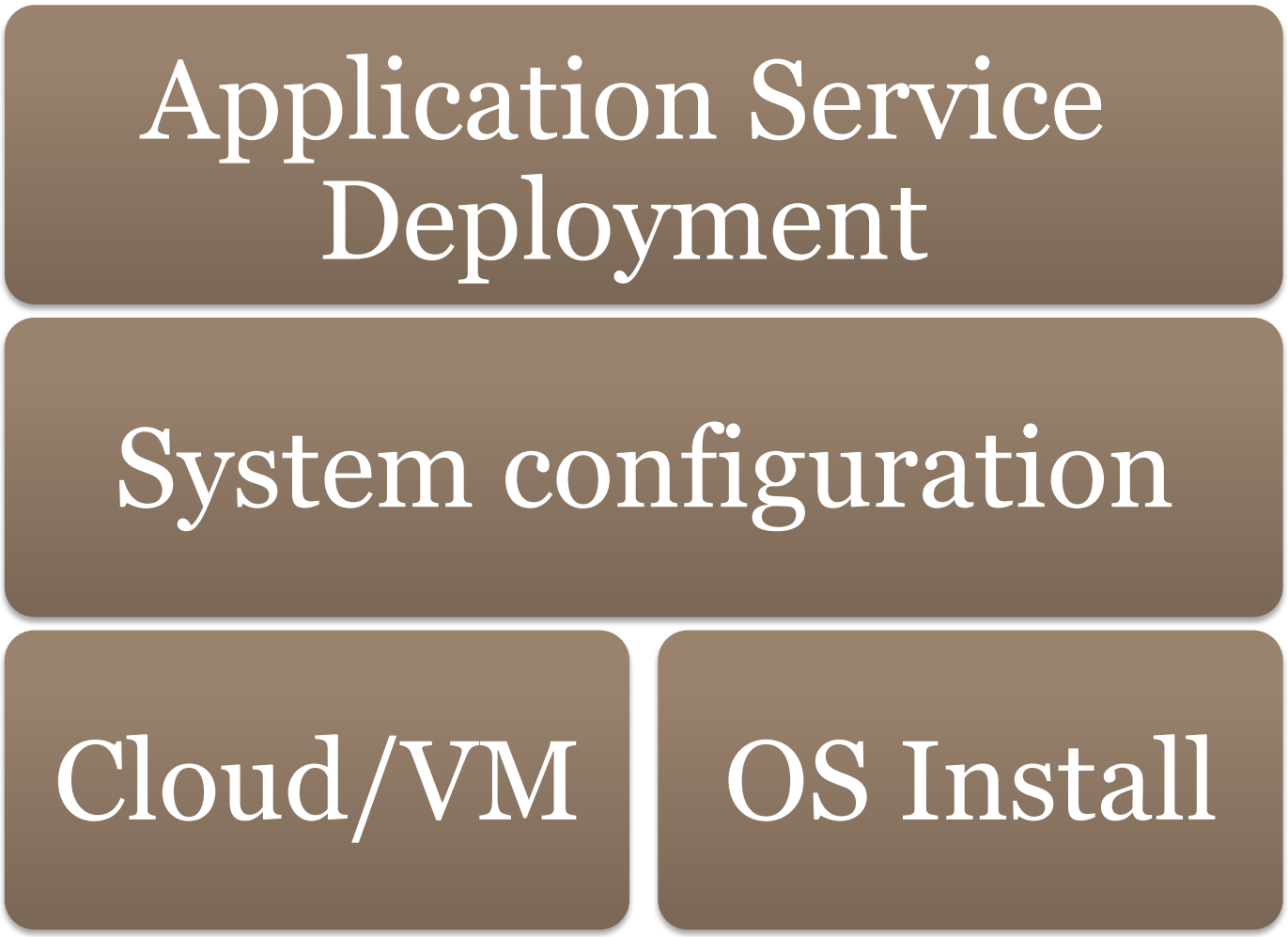
```
class ShipmentNumberClient implements HealthCheckable {

  // initialized from environment cfg
  private final String healthCheckUri;

  // ...

  @Override
  public Health healthCheck() {
    return new Health("bring-shipment-number", isHealthy());
  }

  private boolean isHealthy() {
    try {
      Client client = HttpClientFactory.createTimingoutClient(2000);
      WebResource webResource = client.resource(healthCheckUri);
      return fromStatusCode(webResource.head().getStatus()) == OK;
  } catch (Exception ignored) {
      return false;
  }
}
```

Build information

- Maven: buildnumber-maven-plugin
  - Generates ${buildNumber}
  - Can generate timestamp and other build info
- Maven: git-commit-id-plugin
  - Generates various git metadata. Such as ${git.commit.id}
- Date: ${maven.build.timestamp}
- For another example: http://www.blog.project13.pl/index.php/fun/1174/release-maven-git-commit-id-plugin/

Controlled deployment
Zero-down-time redeploy
(and roll-back).

Tool Chain For Setting up Architecture

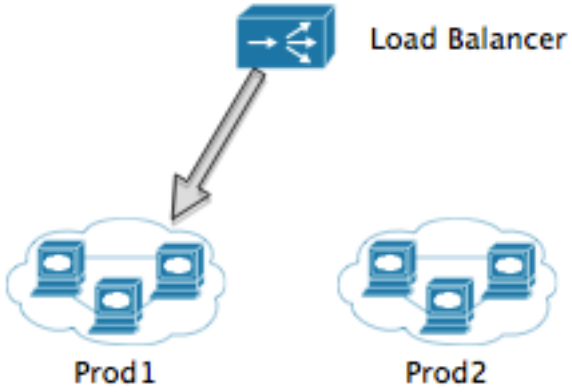


Scripting
Fabric
Capistrano
MCollective
ControlTier
Go

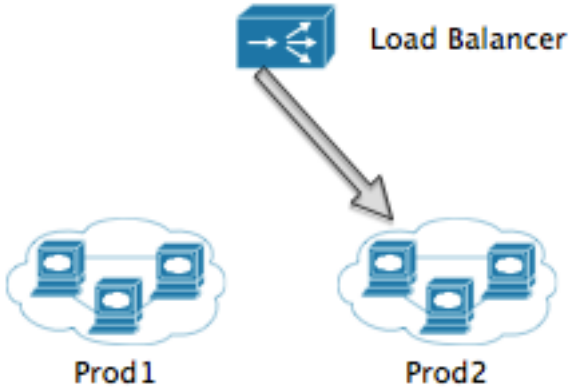
Puppet
Chef
Cfengine

Kickstart
Jumpstart
Solaris LiveUpdate
tftp...

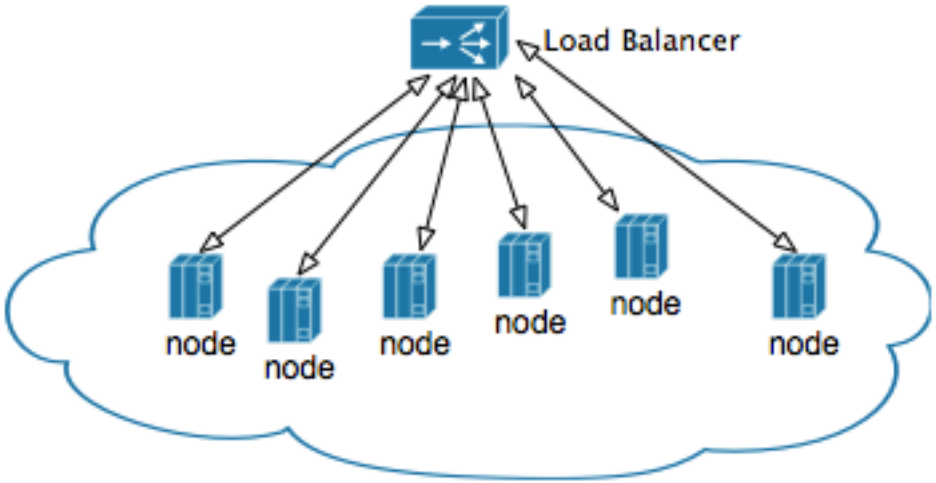
Switch environments



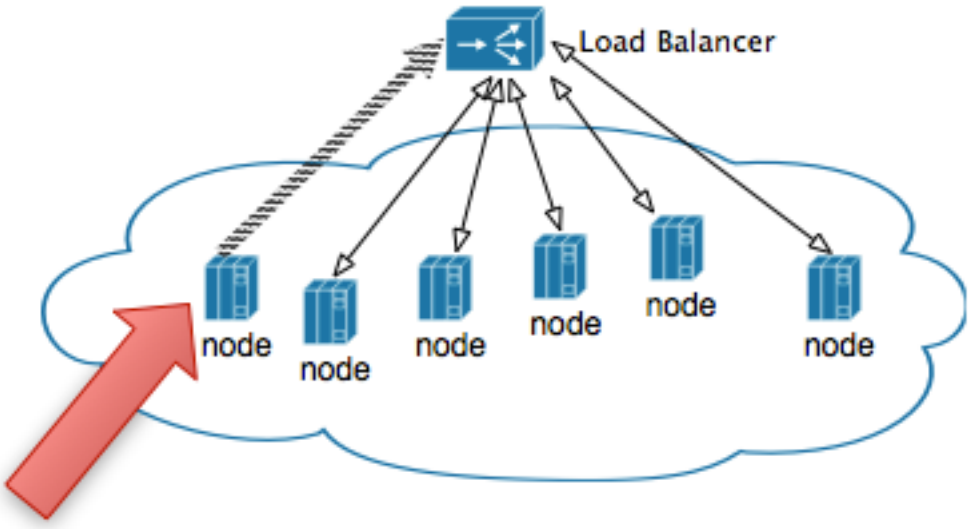
Switch environments



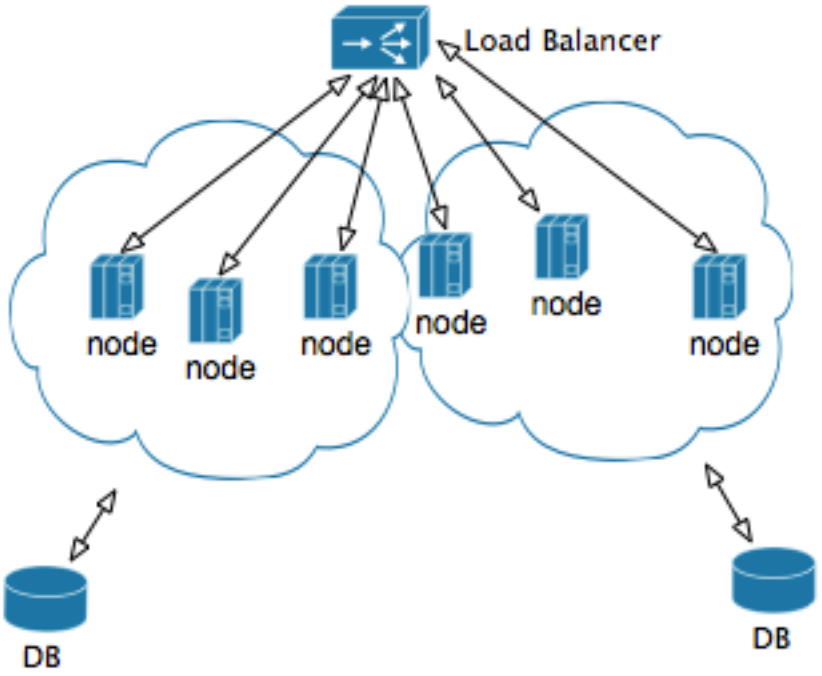
Drain nodes

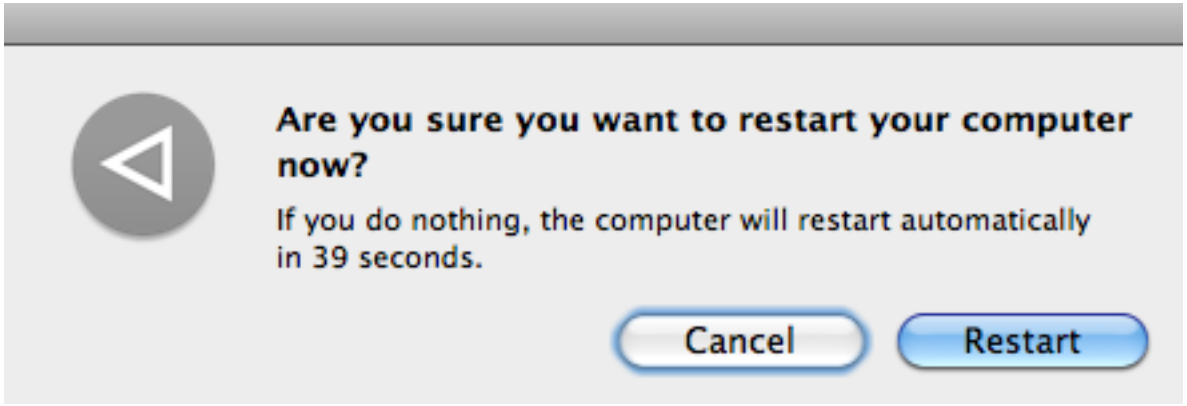


Drain nodes



Next? A/B?





Migrating databases.

Not impossible, but still difficult

- Practice, practice, practice.
- Fail fast.
- Bring the pain forward.
- Refactor the DB.
- Deploy with confidence.

Summary, organizational impact and business value.

3 pillars of Continuous Delivery

1. **Automation** of build, test, deployment, database migrations, and infrastructure
2. **Practices**, such as continuous integration, good configuration management, and testing
3. **People**; having everyone involved in delivery work together throughout the software delivery lifecycle.

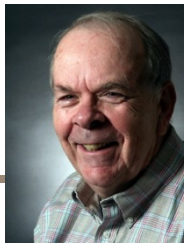
Business Value

- Reduction of cost (through light-weight infrastructure and simplicity).
- Reduction of risk (through more automation and practice).

But most important:

- Frequent release of new software functionality.
- How can we benefit from releasing new functionality monthly, weekly, or even daily?
- How will our organization and business processes need to change?

Agile/adaptive maturity



Continuous delivery may be the next big step in delivering strategic business value to clients quickly, with lower risk and possibly lower cost. However, it won't happen unless leadership understands its potential strategic impact and the organizational adaptability necessary to implement it.

- Jim Highsmith

- Many organizations are stuck at Agile 101.
  - the rule-based approach to agile (do this, don't do that).
  - a necessary first step towards becoming agile, but it's only a first step.
- The next step is to embrace change and respond respectively.
  - the entire organization, both delivery teams and management.
  - embrace the process changes required to respond rapidly.
  - engage in the collaboration required between development and operations.
  - embrace an adaptive, exploratory mindset.

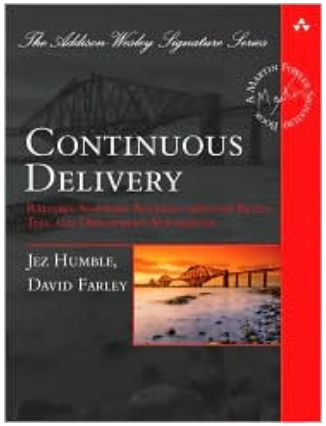
References

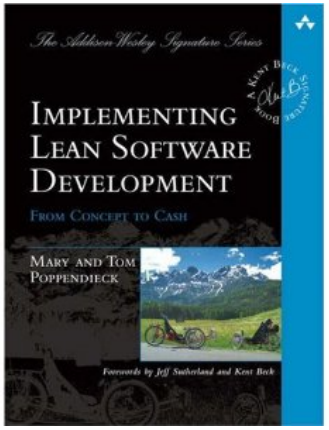
- http://continuousdelivery.com/ (Jez Humble)
- http://pragprog.com/titles/mnee/release-it (Michael T. Nygard)
- http://martinfowler.com/articles/continuousIntegration.html
- http://nvie.com/posts/a-successful-git-branching-model/
- http://progit.org/book/

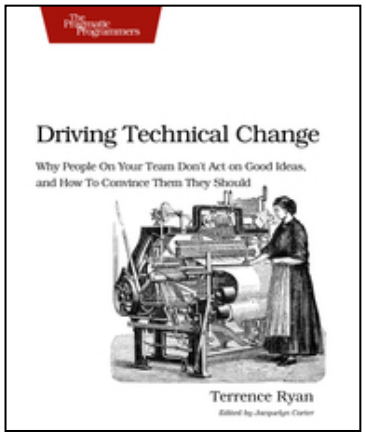
License & Copyright stuff

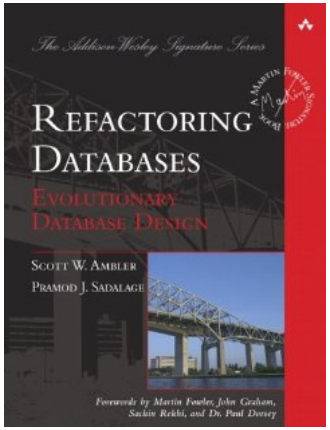
- Images used are either fair use (book covers, Star Wars), public domain or Creative Commons BY-SA licensed.
-

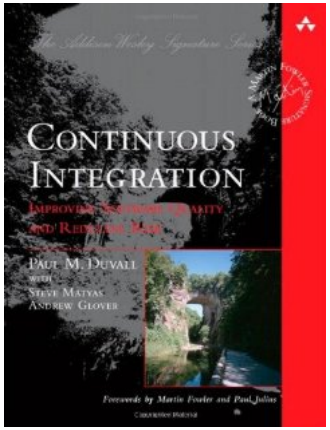
Recommended reading

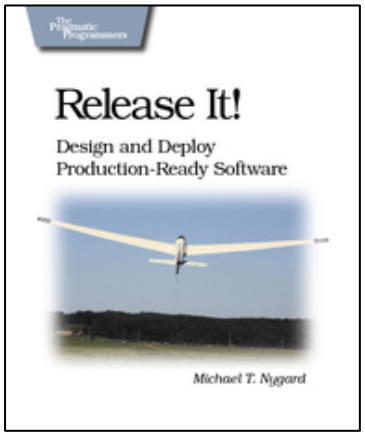


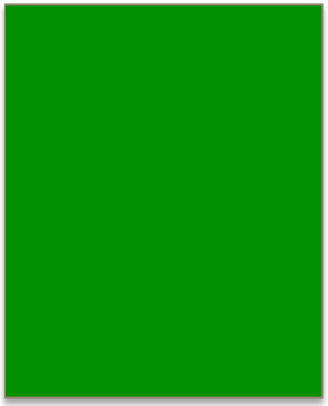
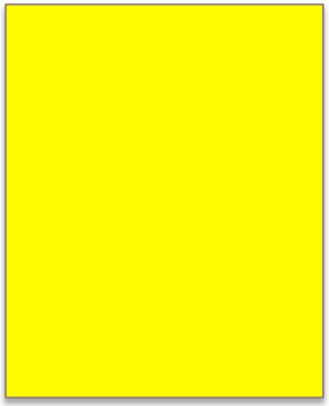











speakerrate SpeakerRate
@steinim @olecr have a great talk today. Remember to ask attendees to give you feedback at http://spkr8.com/t/7621
3 hours ago ☆ Favorite ↻ Retweet ↩ Reply

BEKK

Stein Inge Morisbak
Developer, Technology
+47 909 64 372
stein.inge.morisbak@BEKK.no
@steinim

Ole Christian Rynning
Creative leader, Technology
+47 982 19 347
ole.chr.rynning@BEKK.no
@olecr

BEKK CONSULTING AS
SKUR 39, VIPPETANGEN. P.O. BOX 134 SENTRUM, 0102 OSLO, NORWAY. WWW.BEKK.NO

http://open.bekk.no/