

สยามชั้นนำคิท  
SIAM CHANNON KIT



**Shu Ha Ri**



**SEAL**



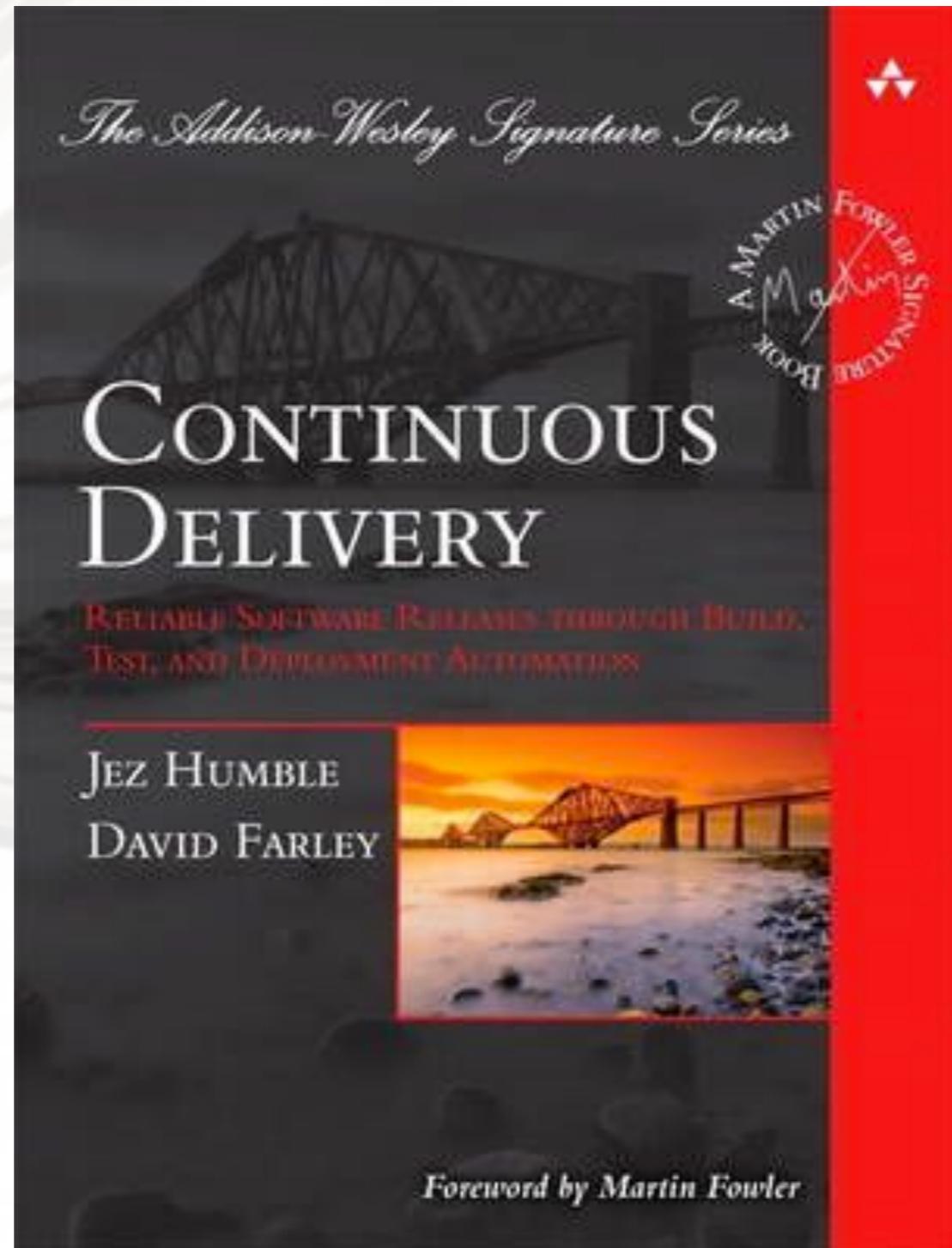
**We Love Bug**

## **Continuous Delivery**

Reliable Software Releases Through Build , Test and Deployment Automation



# Continuous Delivery



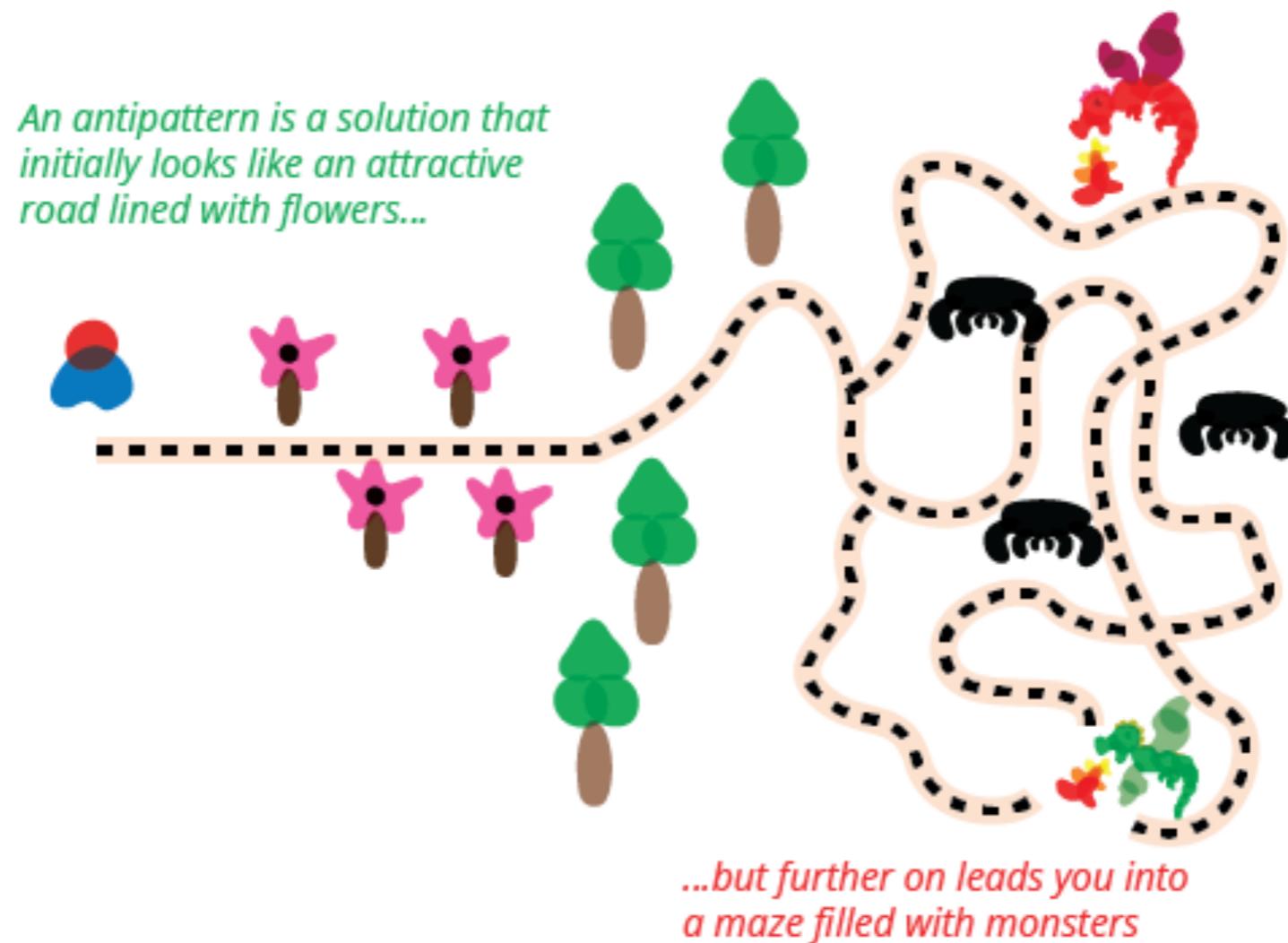
Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# The Problem of Delivering Software



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

# Some Common Release Antipatterns



“An antipattern is just like a pattern, except that instead of a solution it gives something that looks superficially like a solution but isn't one.”

-- Andrew Koenig

Source: <https://martinfowler.com/bliki/AntiPattern.html>



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.

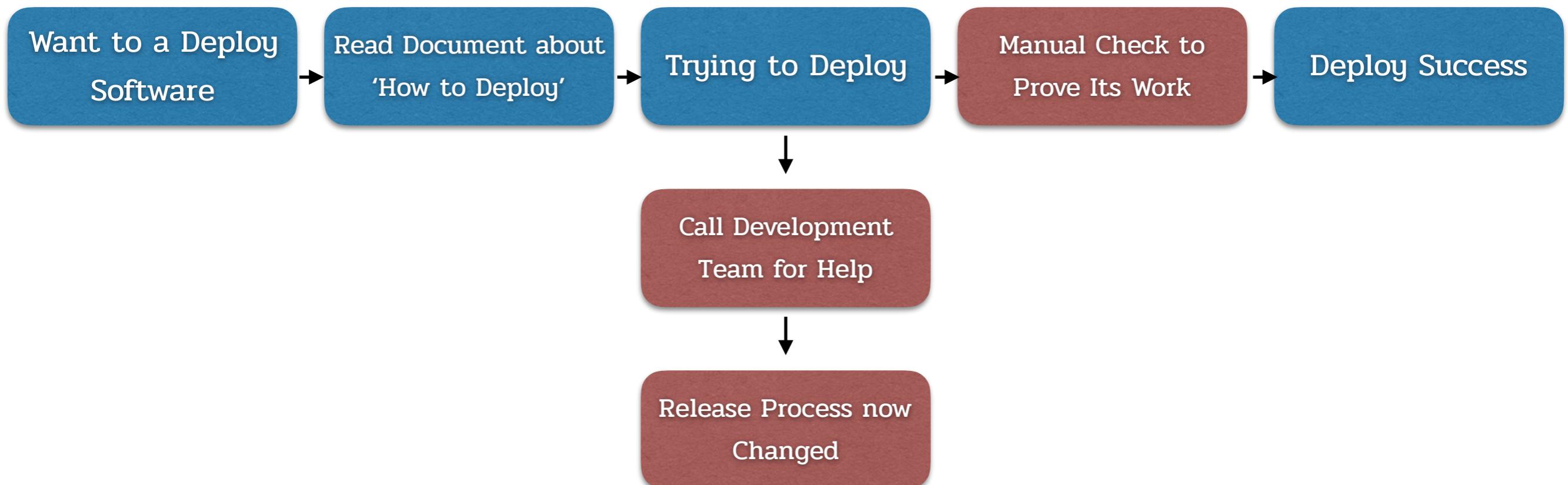
# Some Common Release Antipatterns

- Deploying Software Manually
- Deploying to a Production-like Environment Only after Development Is Complete
- Manual Configuration Management of Production Environments



# Antipatterns : Deploying Software Manually

expected time : 15 mins



actual time : > 1 hours



# Antipatterns : Deploying Software Manually

## Automated Deployment

Detect errors fast

Repeatable

Fast



Reduce documentation

No human errors

Cheap

Just press the 'Deploy' button



Share — copy and redistribute the material in any medium or format.

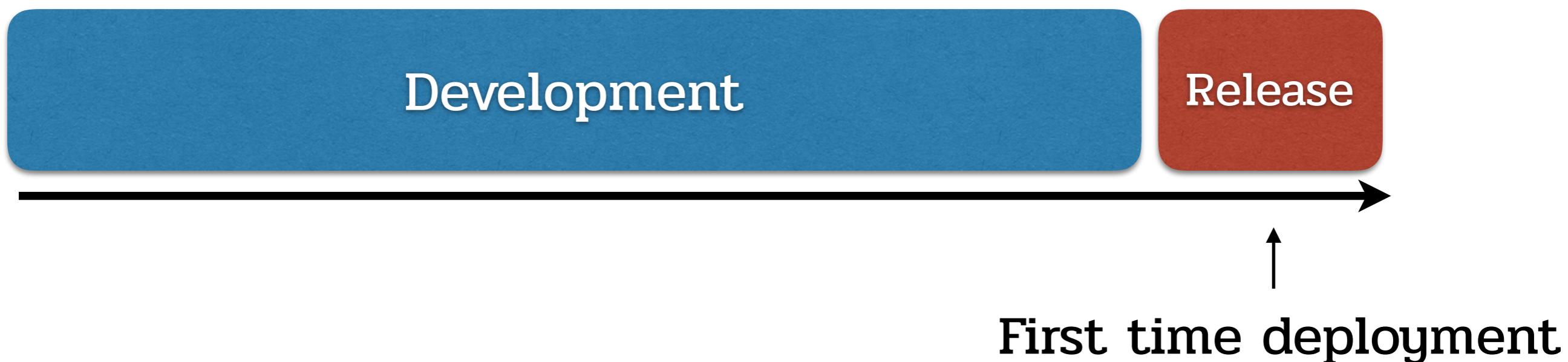
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

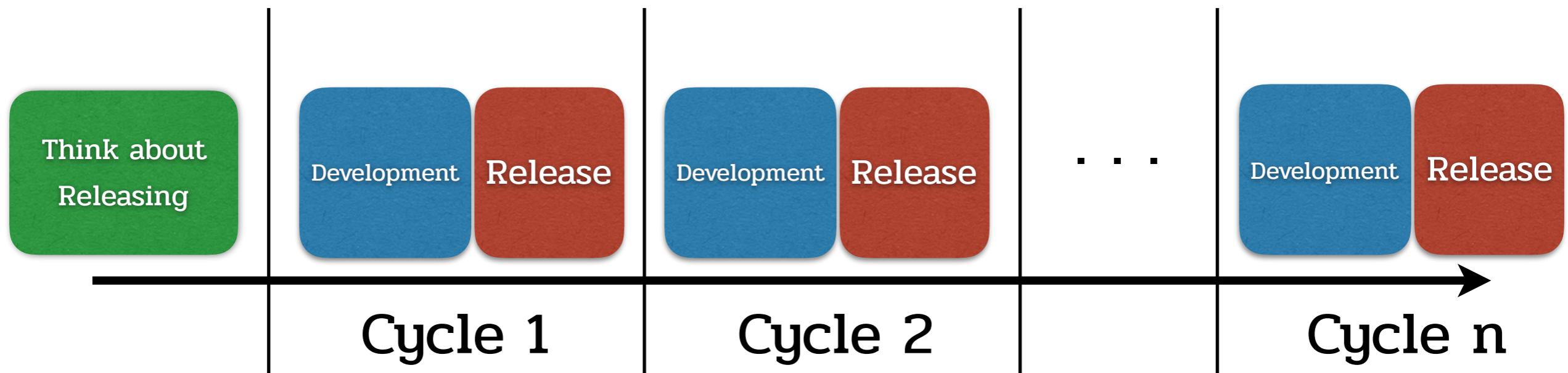
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

## **Antipatterns : Deploying to a Production-like Environment Only after Development Is Complete**

In this pattern, the first time the software is deployed to a production-like environment is once most of the development work is done—at least, “done” as defined by the development team.



# Antipatterns : Deploying to a Production-like Environment Only after Development Is Complete



# Antipatterns : Manual Configuration Management of Production Environments

> Maunal Config

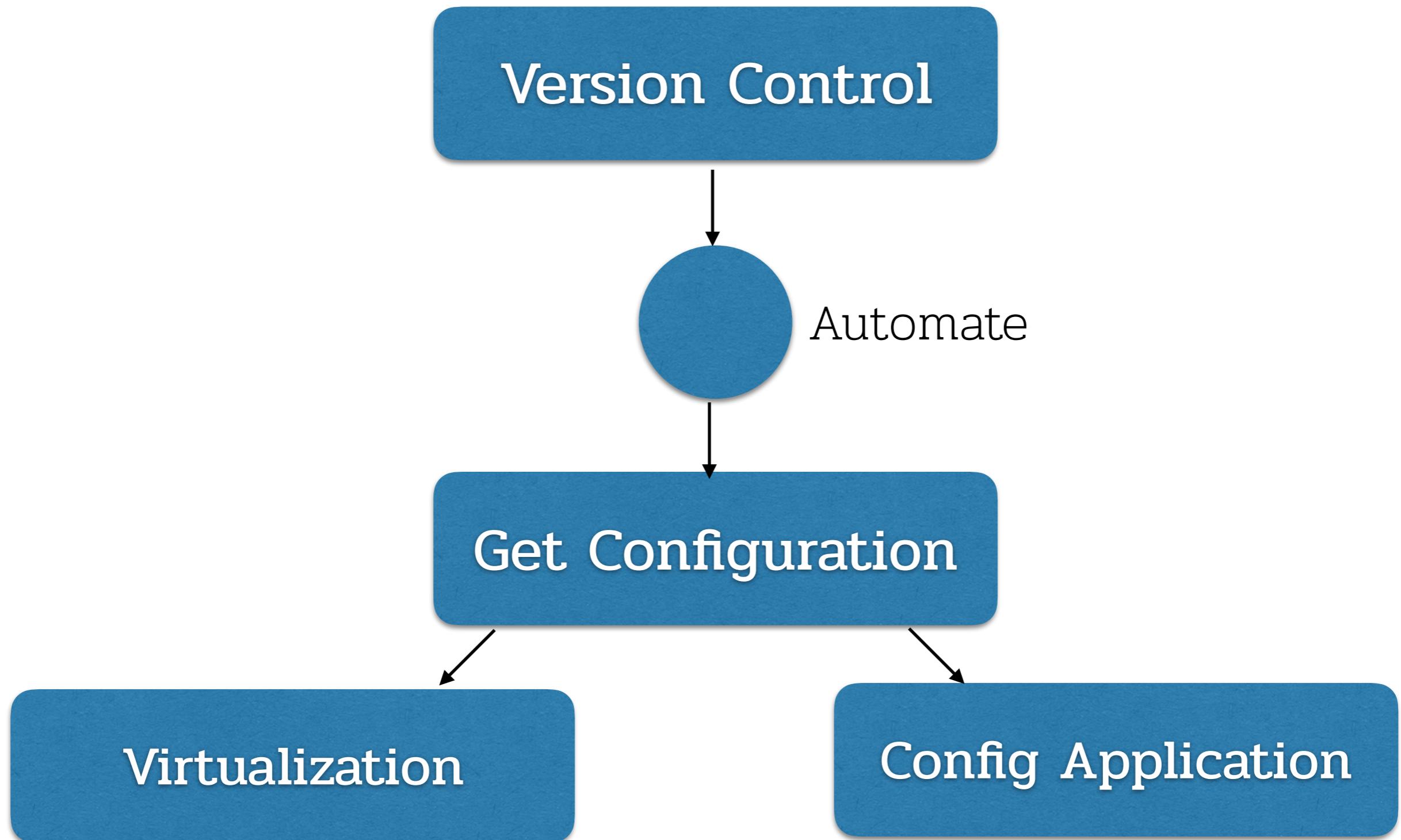
Staging fine but production fail

Get different of some environments

Unable rollback the config



# Antipatterns : Manual Configuration Management of Production Environments

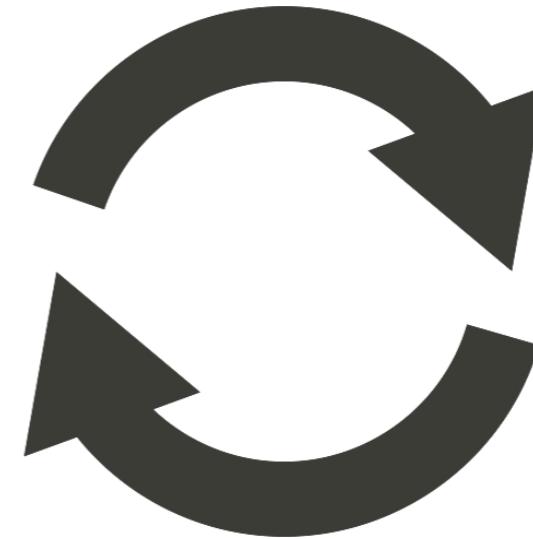


# How do we achieve our goal

“low cycle time and high quality”



Automate



Frequency Feedback

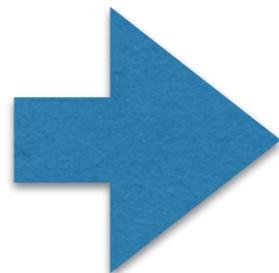
# How to achieve the Feedback Criteria

- Every Change Should Trigger the Feedback Process
- The Feedback Must Be Received as Soon as Possible
- The Delivery Team Must Receive Feedback and Then Act on It



# What Are The Benefits ?

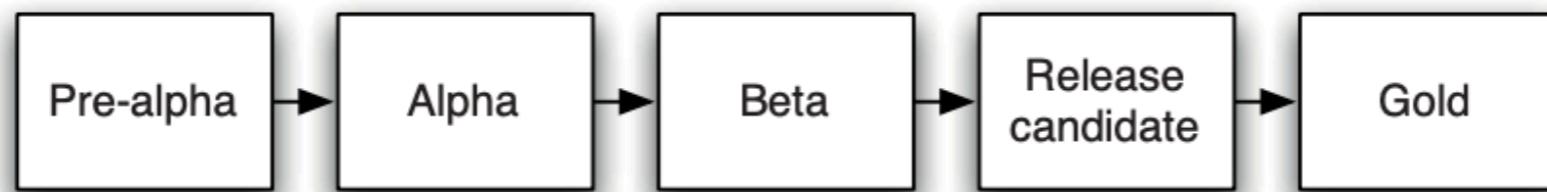
Create release process  
repeatable,  
reliable  
and predictable



- Empowering Team
- Reducing Errors
- Lower Stress
- Deployment Flexibility



# The Release Candidate

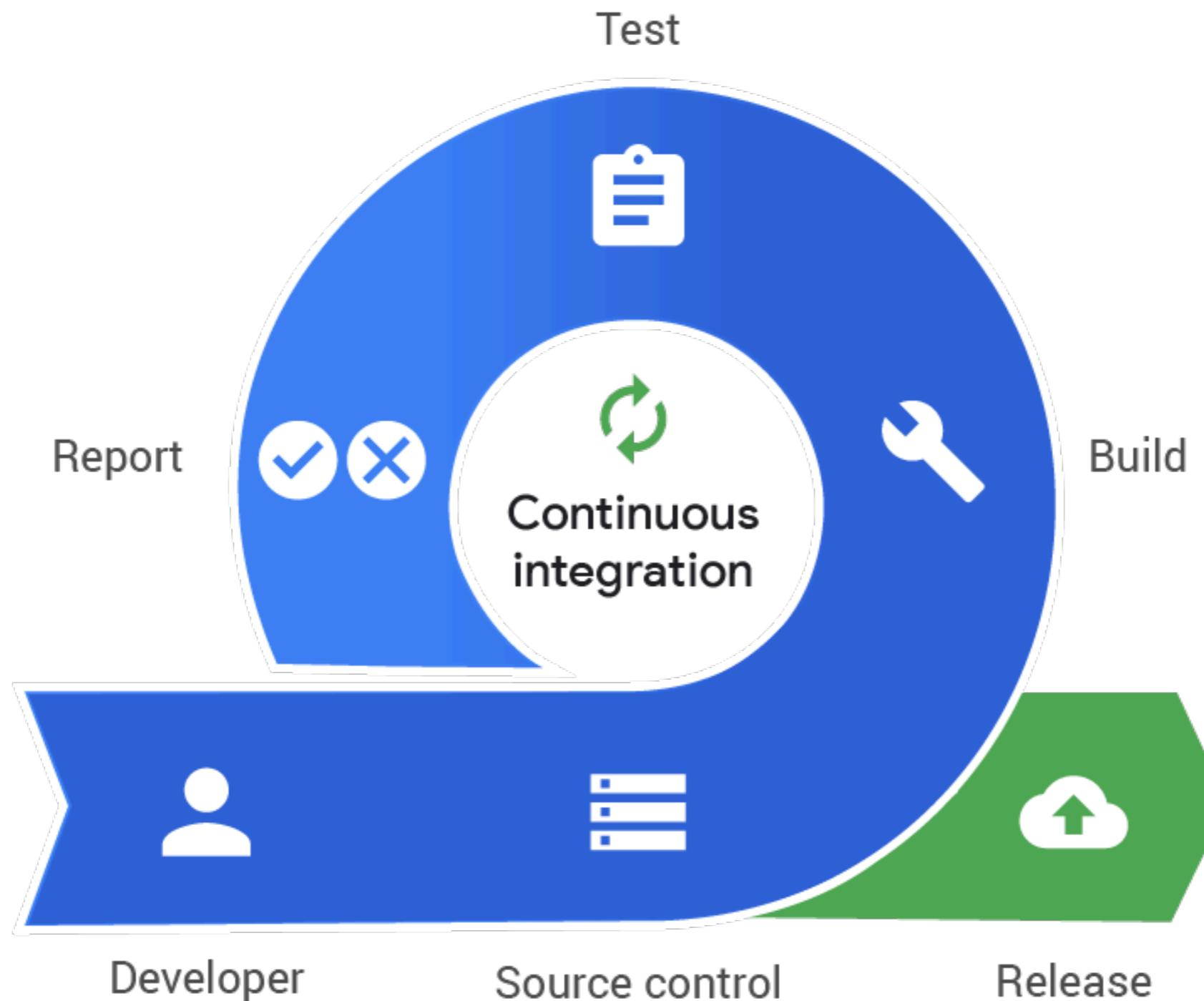


**Figure 1.2** *Traditional view of release candidates*

“A change to your code may or may not be  
releasable.”



# Every Check-in Leads to a Potential Release



# Principles of Software Delivery

- Create a Repeatable, Reliable Process for Releasing Software
- Automate Almost Everything
- Keep Everything in Version Control
- If It Hurts, Do It More Frequently, and Bring the Pain Forward
- Build Quality In
- Done Means Released
- Everybody Is Responsible for the Delivery Process
- Continuous Improvement



# Configuration Management

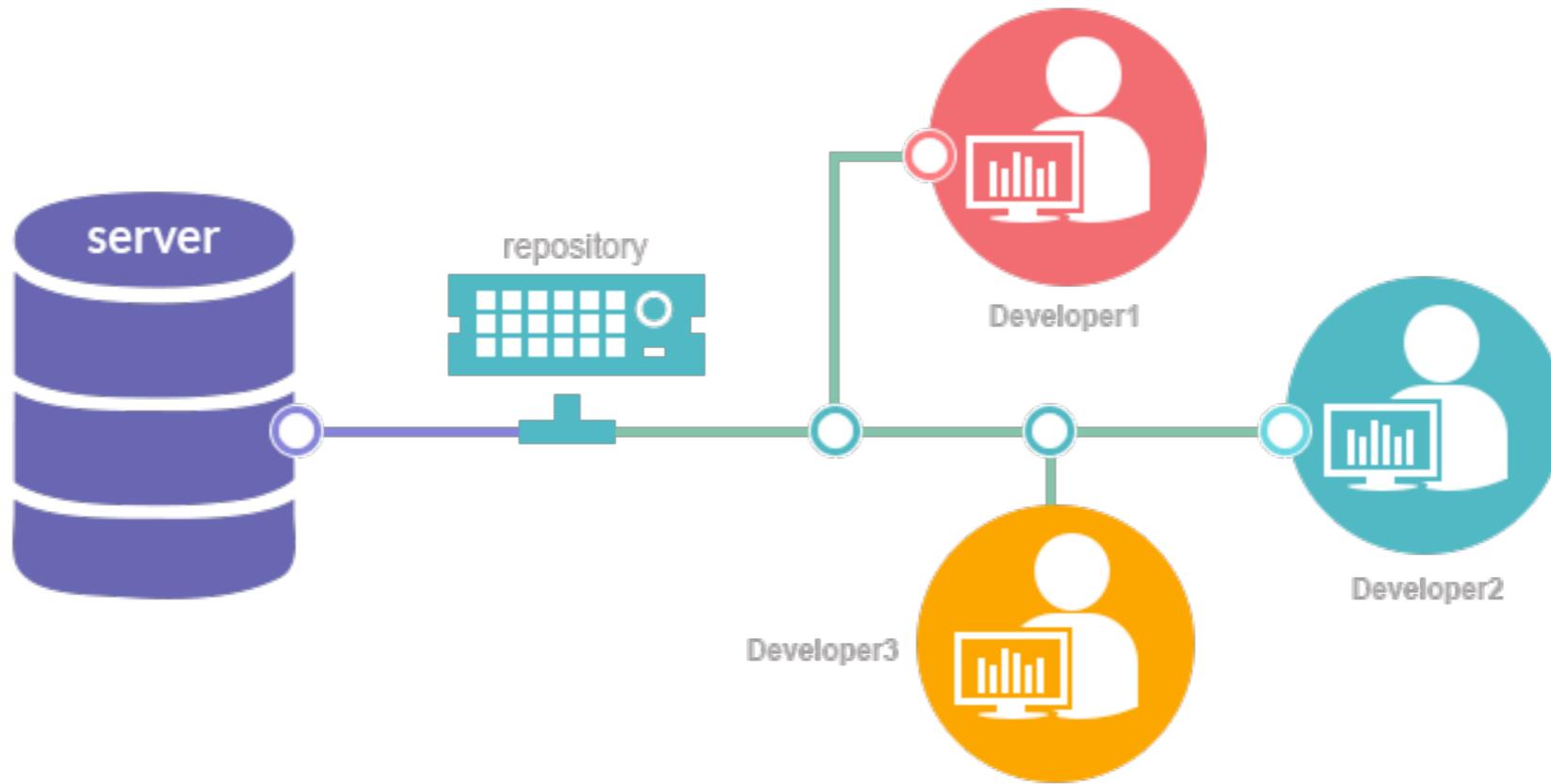


# Good configuration management can answer the quest below

- Can I exactly reproduce any of my environment with version configuration?
- Can I easily make change and deploy it to my environment?
- Can I trace back to see exactly what the change was, who made it and when?
- Can I satisfy all of the compliance regulations that I am subject to?
- It is easy for every member of the team to get the information that to make a change?
- Does the strategy get in the way of efficient delivery, leading to increased cycle time and reduced feedback?



# Using Version Control



- Keep Absolutely Everything in Version Control
- Check In Regularly to Trunk
- Use Meaningful Commit Messages



# Managing Dependencies

- **Managing External Libraries**

- Keep copies libraries on repository that version suited with our application

- **Managing Component**

- Split smallest of application into component for efficient development process



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Managing Software Configuration

- Configuration and Flexibility
- Types of Configuration
- Managing Application Configuration
- Managing Configuration across Applications
- Principles of Managing Application Configuration



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Managing Your Environments

- Tools to Manage Environments
  - ex. Puppet , CfEngine
  - Virtualization
- Managing the Change Process
  - Any change can be break
  - Test process like code frequent



# Continuous Integration

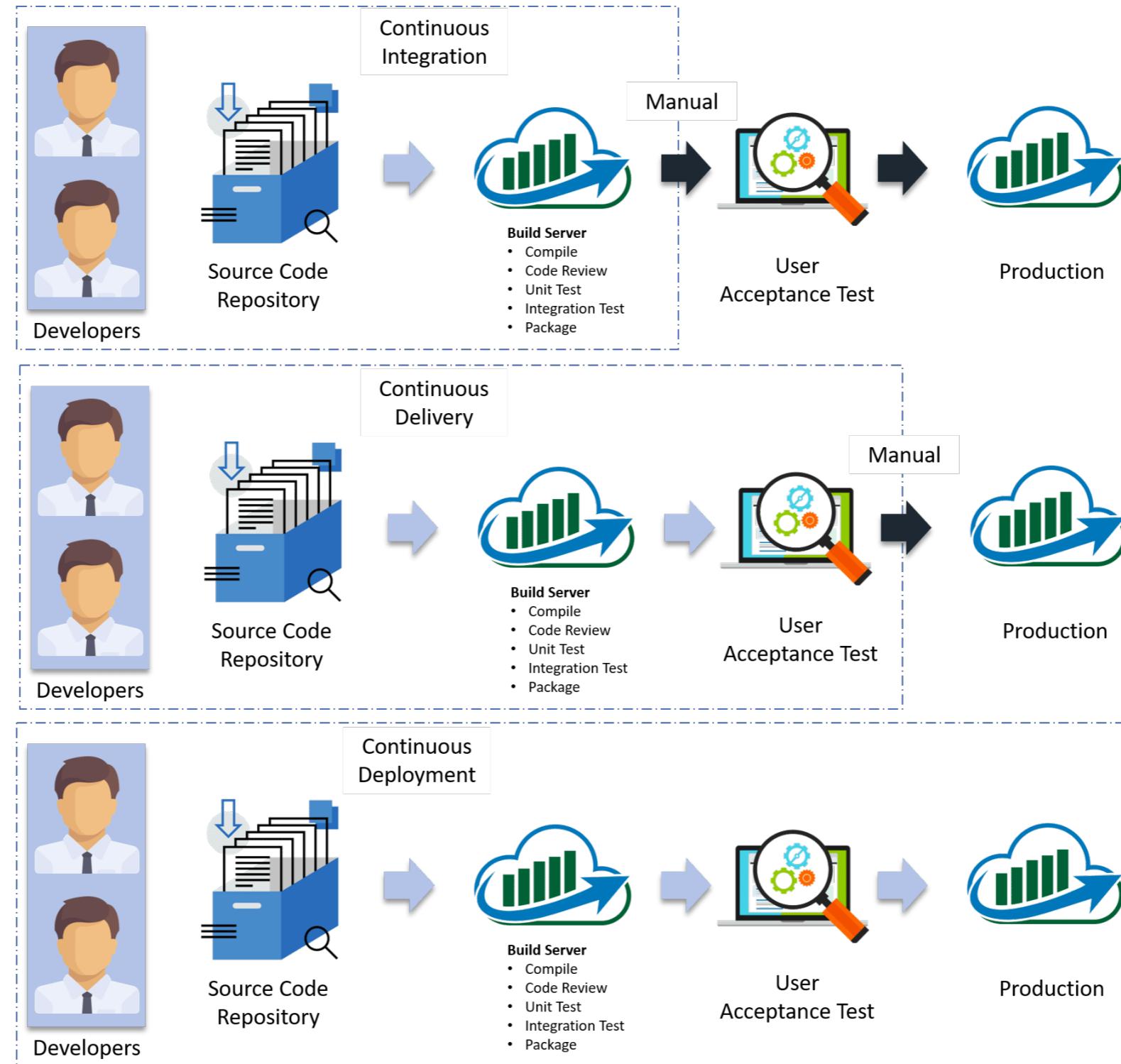


Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

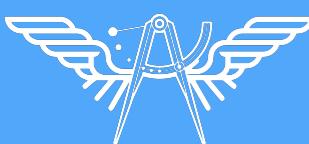
# Continuous Integration



# Continuous Integration / Continuous Delivery / Continuous Deployment



<https://dzone.com/articles/building-a-continuous-delivery-pipeline-using-jenk>



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# What You Need Before You Start

1. Version Control
2. An Automate Build
3. Agreement of the Team



# Implementing Continuous Integration

- A Basic Continuous Integration System
- Prerequisites for Continuous Integration
- Using Continuous Integration Software
- Essential Practices



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# A Basic Continuous Integration System

1. Work with rest of team to fix the build failure
2. Update your code from repository after your task finished and tested
- 3. Run build and test on your machine to confirm its still work**
- 4 If local build and test passes , check your code into version control**
5. Wait for your CI tool to run the build with your changes.
6. If it fails, stop what you're doing and fix the problem immediately –go to step 3.
7. If the build passes, rejoice and move on to your next task.



# Prerequisites for Continuous Integration

- Check In Regularly
- Create a Comprehensive Automated Test Suite
- Keep the Build and Test Process Short
- Managing Your Development Workspace



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Essential Practices

- Don't Check In on a Broken Build
- Always Run All Commit Tests Locally before Committing
- Wait for Commit Tests to Pass before Moving On
- Never Go Home on a Broken Build
- Always Be Prepared to Revert to the Provision Revision
- Time-Box Fixing before Reverting
- Don't Comment Out Failing Tests
- Take Responsibility the Result form Your Changes
- Test-Driven Development



Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Suggested Practices

- Extreme Programming Development Practices
  - TDD , Share Code Ownership , Refactor
- Failing a Build for Architectural Breaches
- Failing the Build for Slow Tests
- Failing the Build for Warnings and Code Style Breaches

\*\* Failing Build need to have strategy for closing the gap \*\*



# Implementing a Testing Strategy



What does

**CONT. DEL.** say about

# TEST STRATEGY

66

TESTING IS A CROSSFUNCTIONAL ACTIVITY THAT INVOLVES THE WHOLE TEAM, AND SHOULD BE DONE CONTINUOUSLY FROM THE BEGINNING OF THE PROJECT.



HOW DO I KNOW WHEN I'M DONE?



ANSWERS

DID I GET WHAT I WANTED?

? NOT MUCH INFORMATION REGARDING THIS TYPE OF TESTS IN THE BOOK.

INTEGRATION TEST - TEST THAT ENSURE THAT EACH INDEPENDENT PART OF YOUR APPLICATION WORKS CORRECTLY WITH THE SERVICES IT DEPENDS ON.



WILL FORM PART OF YOUR REGRESSION TEST SUITE

UNIT TEST  
COMPONENT TEST  
DEPLOYMENT TEST

BUSINESS FACING		MANUAL
PROGRAMMING	AUTOMATED	MANUAL
SUPPORT	<ul style="list-style-type: none"> <li>• FUNCTIONAL ACCEPTANCE TESTS</li> </ul>	<ul style="list-style-type: none"> <li>• SHOWCASES</li> <li>• USABILITY TESTING</li> <li>• EXPLORATORY TESTING</li> </ul>
PROGRAMMING	<ul style="list-style-type: none"> <li>• UNIT TESTS</li> <li>• INTEGRATION TESTS</li> <li>• SYSTEM TESTS</li> </ul>	<ul style="list-style-type: none"> <li>• NONE FUNCTIONAL ACCEPTANCE TESTS (CAPACITY, SECURITY...)</li> </ul>
TECHNOLOGY FACING	AUTOMATED	MANUAL/AUTOMATED

YOUR DEPLOYMENT PIPELINE SHOULD HAVE ALL THESE FOUR TYPE OF TESTS.

ANY PLAN THAT DEFERS TESTING TO THE END OF THE PROJECT IS **BROKEN**.



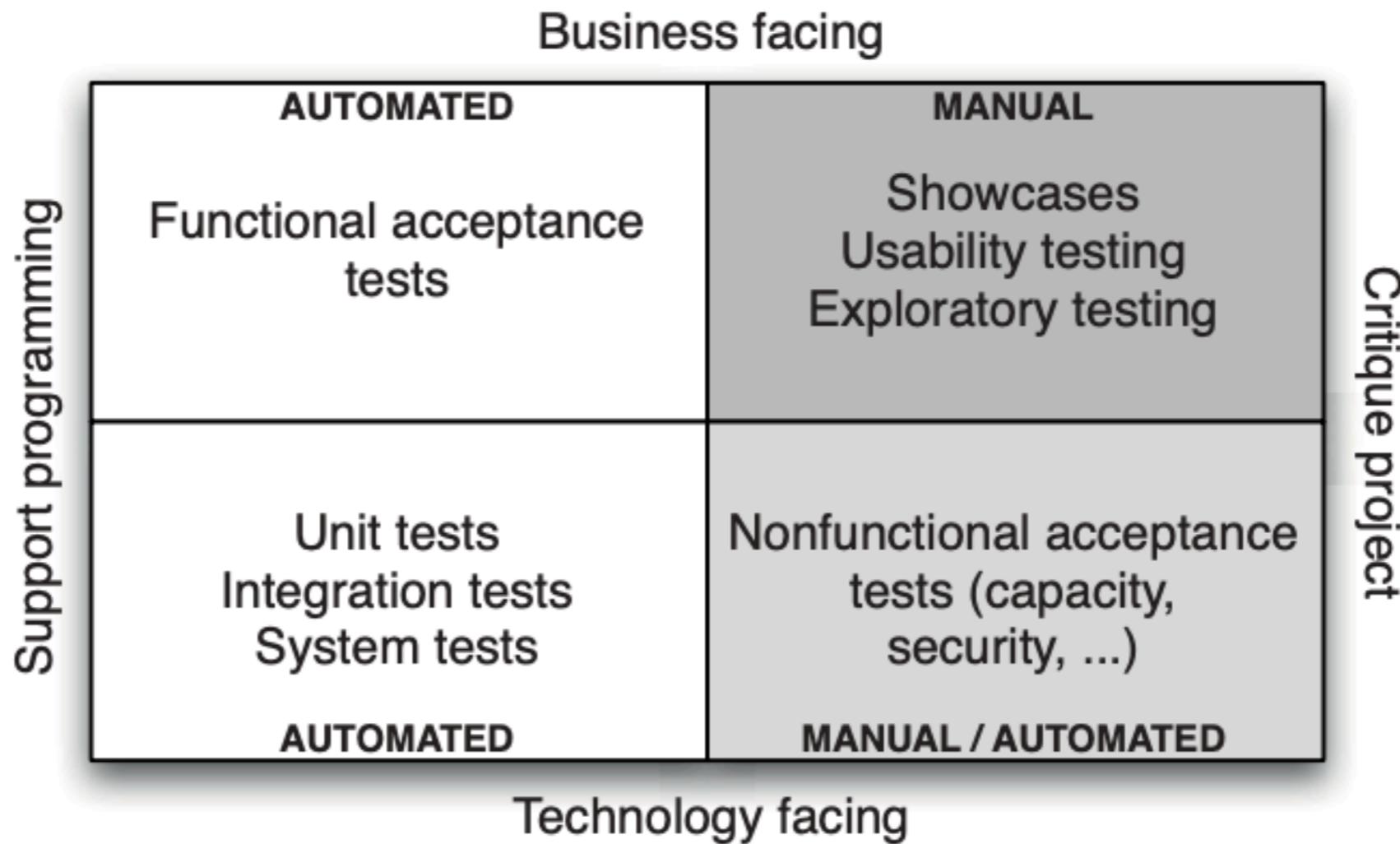
Source: Nhan Ngo, a QA engineer at Spotify,



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Type of Tests



**Figure 4.1** *Testing quadrant diagram, due to Brian Marick, based on ideas that were “in the air” at the time*



# Real-Life Situations and Strategies

## New Project

- Important to start writing automate acceptance tests from the very beginning
- Choose technology , Testing tools
- Strict the process



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Real-Life Situations and Strategies

## Mid Project

- Automate testing the most common , most important and high value use cases
- Identify with customer where is real business value
- Specify and clearly the objective of you tests



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Real-Life Situations and Strategies

## From Legacy System To Automation Deployment

- First create **automated build** process (Continuous Integration - XP)
- First create **automated deploy** process (Continuous Integration - XP)
- Create **automate test suite** from documents or member of team (Continuous Integration - XP); ใช้ Agile Testing Quadrant-Brian Merick และ Automation Test Pyramid-Mike Cohn
- **Automated deployment** with **automate test suite** to save a lot of effort on manual testing (ให้รู้ผลเร็วขึ้น และข้อผิดพลาดจากคน)
- Explain about value of automate regression testing to unwilling customer
- Identify high-value uses of features
- Test the code that you change



# Anatomy of the Deployment Pipeline



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

# Basic Deployment Pipeline

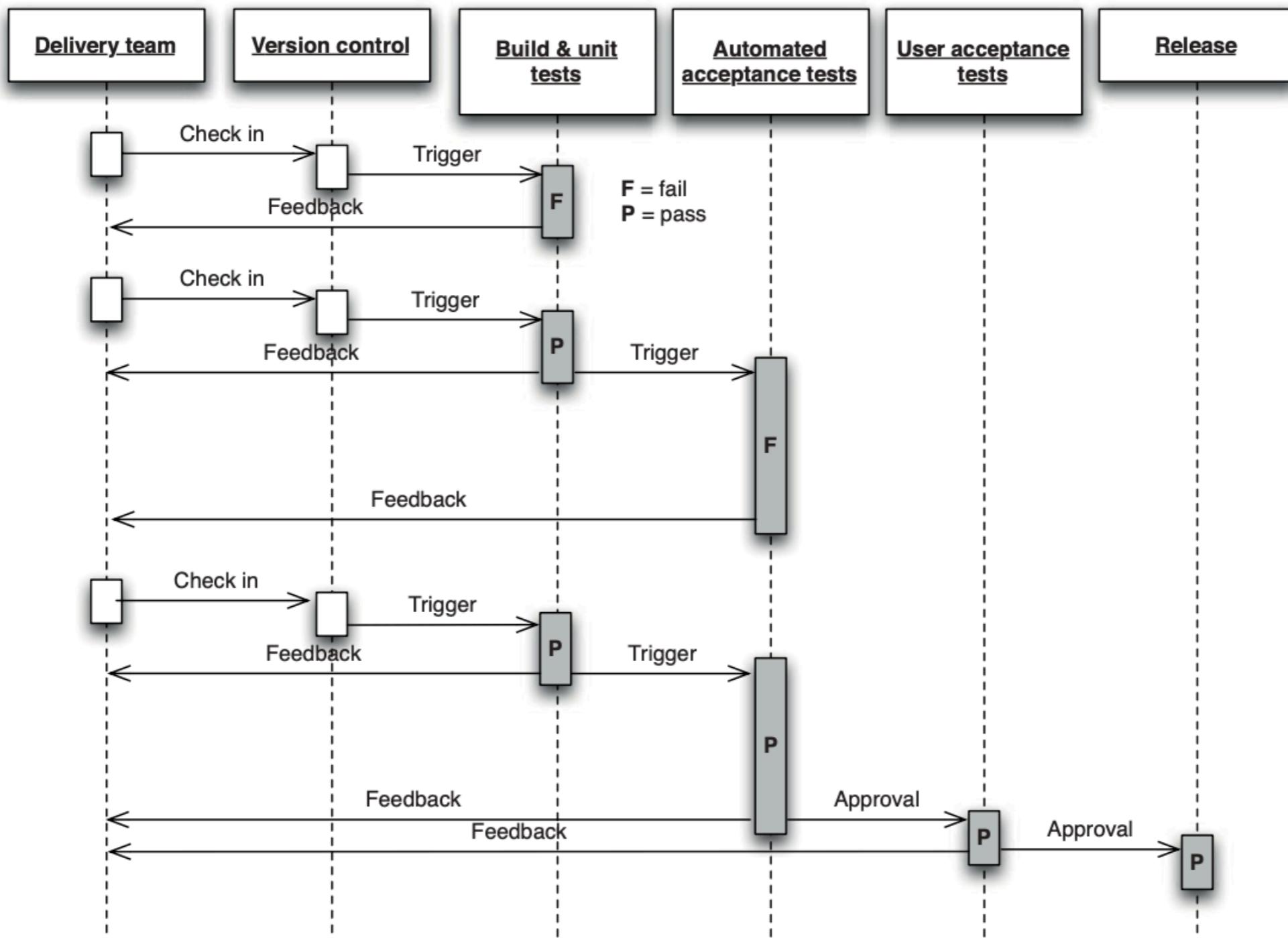


Figure 5.2 *Changes moving through the deployment pipeline*



Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Basic Deployment Pipeline

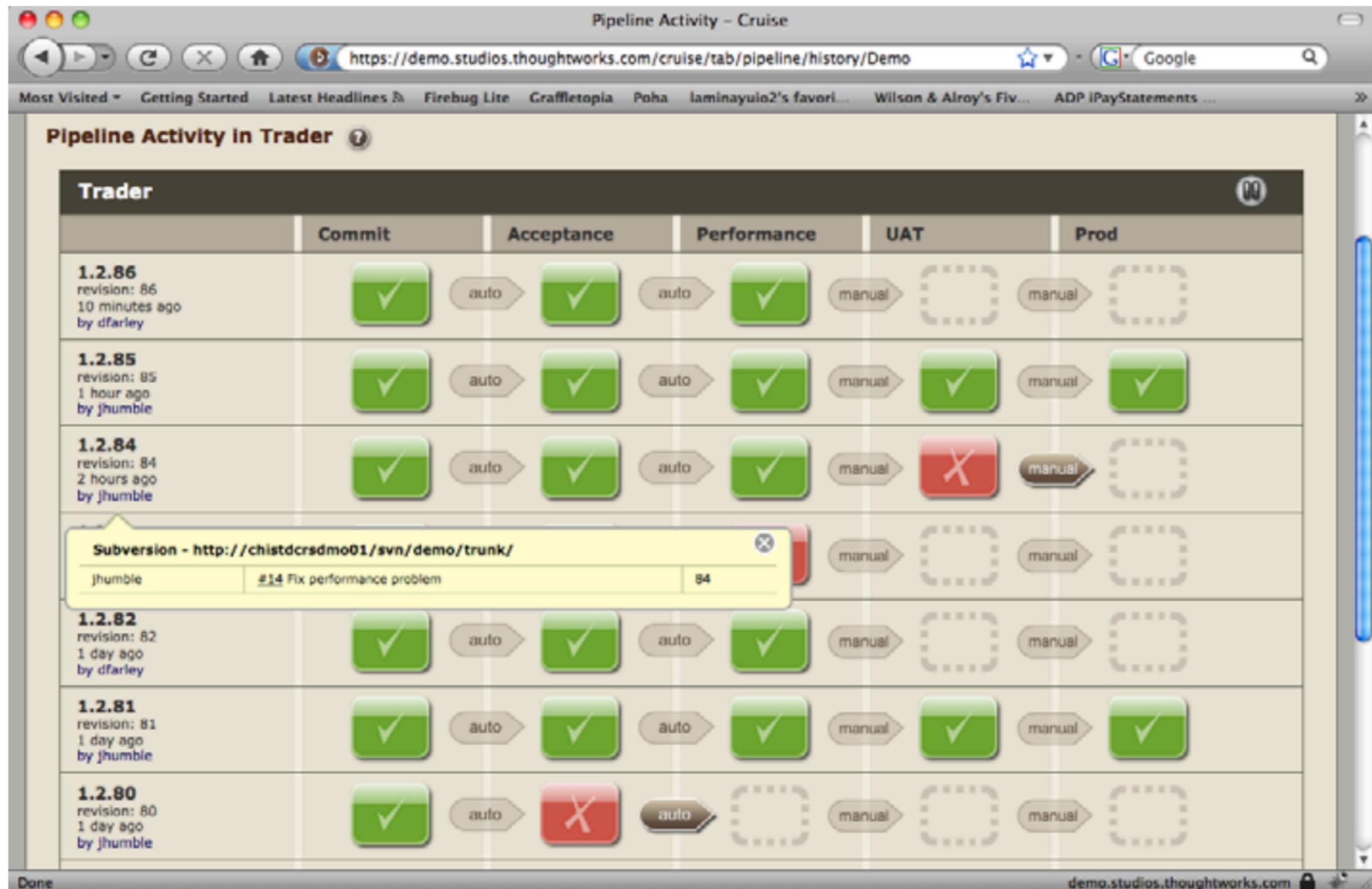


Figure 5.5 Go showing which changes have passed which stages

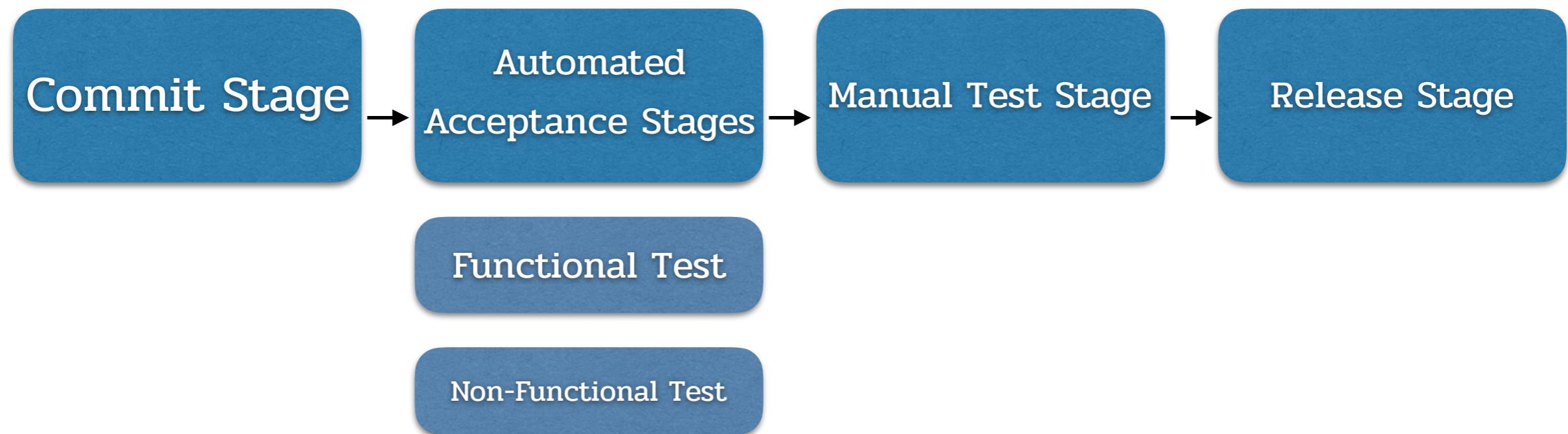


Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Stage of Testing in Releasing Process



# Stage of Testing in Releasing Process

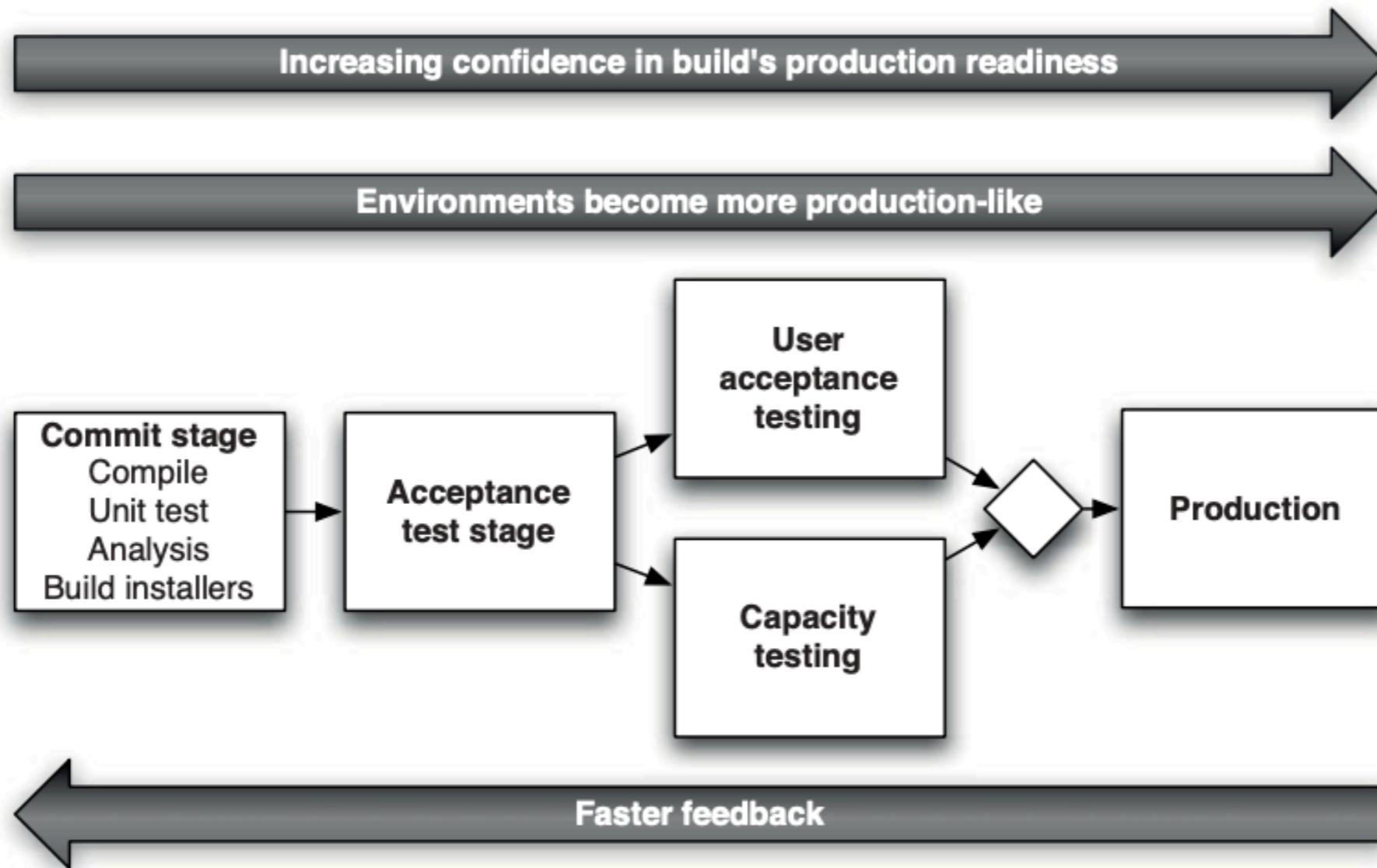


Figure 5.3 *Trade-offs in the deployment pipeline*

# Basic Deployment Pipeline

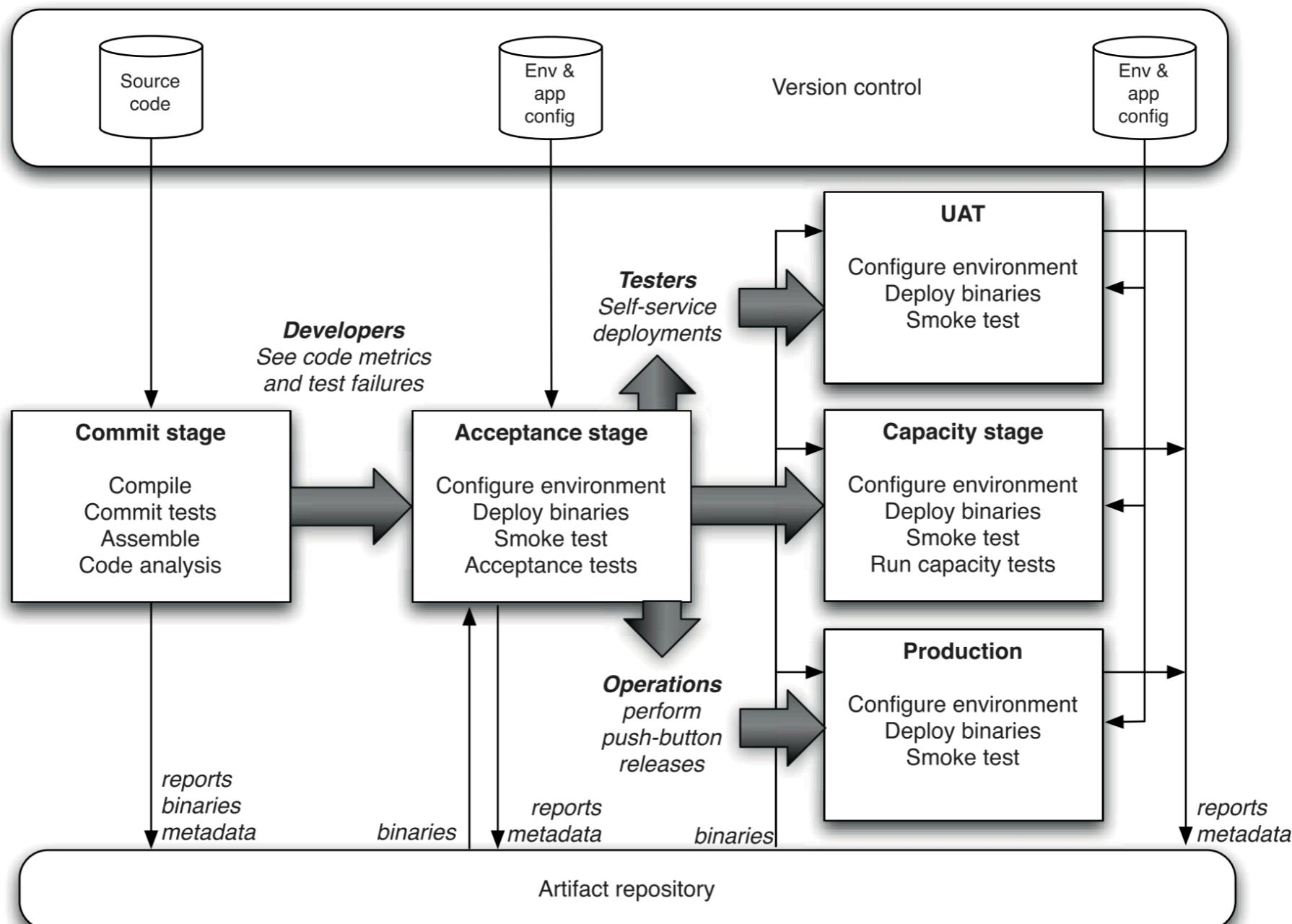


Figure 5.4 Basic deployment pipeline



# Basic Deployment Pipeline

The screenshot shows a web browser window titled "Deployment Page". The address bar displays the URL "http://tfvqa01". The main content area is titled "Deployment Page" and contains a table titled "Available Releases". The table has columns for "Del", "Iteration", "Release", "Build number", "Commit timestamp", and "Actions". The "Actions" column includes buttons for "Deploy", "Re-Deploy", "Stop Release", and "Browse". The table lists four releases:

Del	Iteration	Release	Build number	Commit timestamp	Actions
X	14	4055	build.480	Fri, 02 May 2008 09:09:41	<a href="#">Deploy</a>
X	14	4028	build.479	Wed, 30 Apr 2008 15:35:59	<a href="#">Re-Deploy</a> <a href="#">Stop Release</a> <a href="#">Browse</a>
X	13	3972	build.469	Fri, 25 Apr 2008 16:43:30	<a href="#">Deploy</a>
X	12	3775	build.442	Fri, 11 Apr 2008 17:28:35	<a href="#">Deploy</a>

At the bottom left of the browser window, there is a "Done" button. At the bottom right, there are several small icons.

Figure 5.7 Example deployment page



# Deployment Pipeline Practices

- Only Build Your Binaries Once
- Deploy the Same Way to Every Environment
- Smoke-Test Your Deployments
- Deploy Into a Copy of Production
- Each Change Should Propagate through Pipeline

Instantly

- If any Part of the Pipeline Fails , Stop the Line



Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Topic size 48

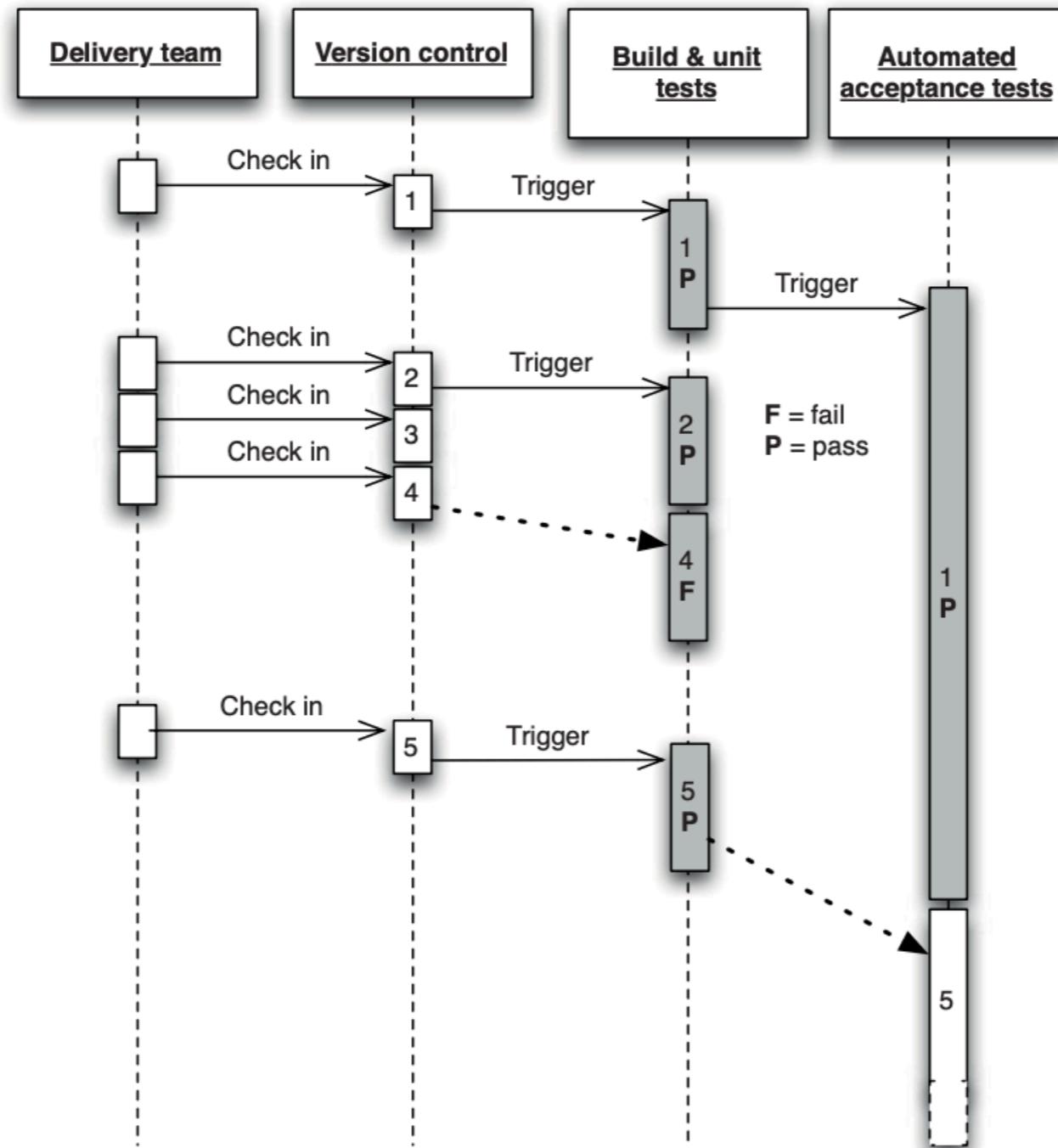


Figure 5.6 *Scheduling stages in a pipeline*



# Basic Deployment Pipeline

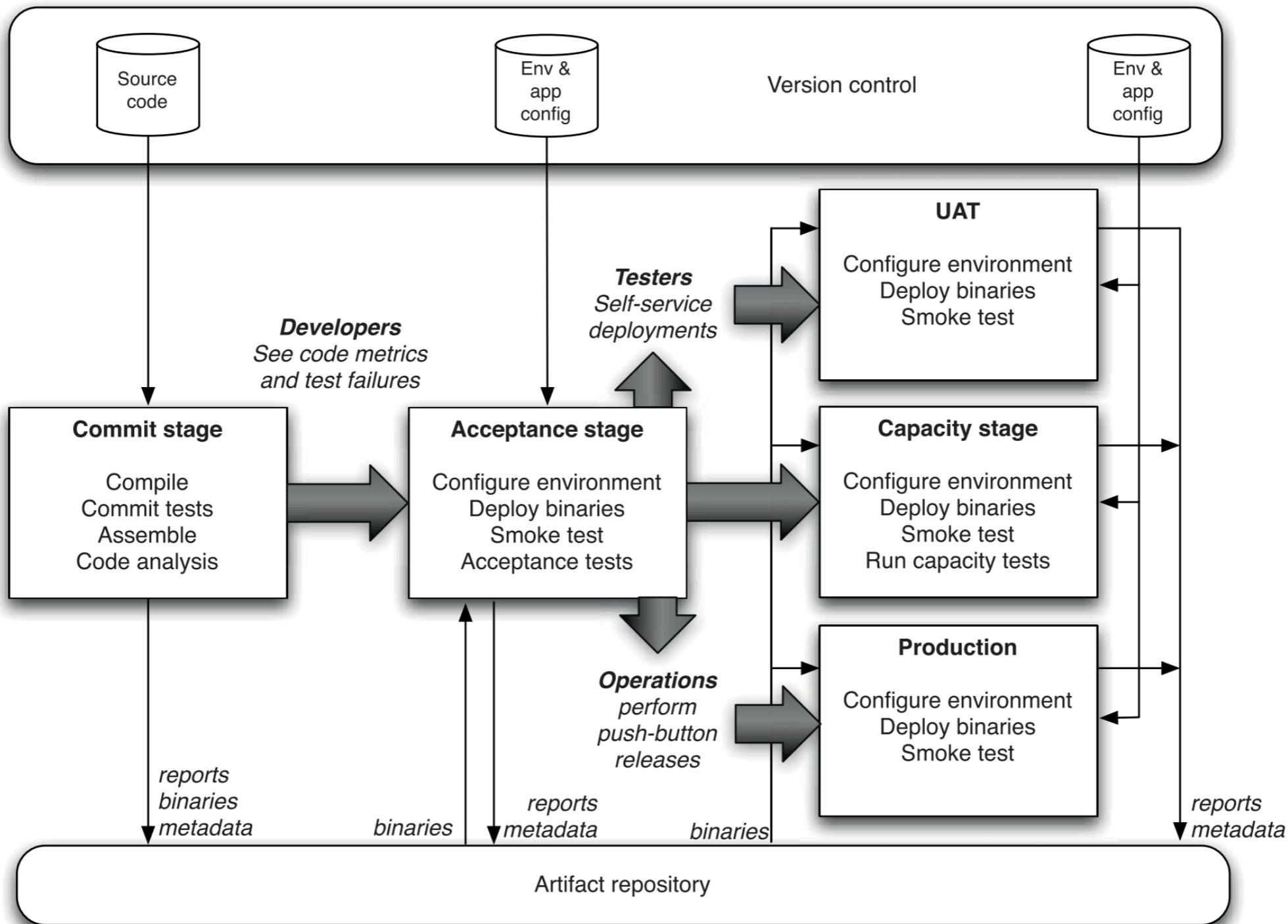


Figure 5.4 Basic deployment pipeline



# Preparing to Release

- Automating Deployment and Release
- Backing Out Changes
- Building on Success



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Implementing a Deployment Pipeline

1. Model your value stream and create a walking skeleton
2. Automating the Build and Deployment Process
3. Automating the Unit Tests and Code Analysis
4. Automating Acceptance Test
5. Automating Release
6. Evolving Your Pipeline



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Metrics

Feedback is heart of any software delivery process , use metrics to improve not only the cycle time.



# Metrics

- Automated test coverage
- Properties of the codebase such as the amount of duplication, cyclomatic complexity, efferent and afferent coupling, style problems, and so on
- Number of defects
- Velocity
- Number of commits to the version control system per day
- Number of builds per day
- Number of build failures per day
- Duration of build, including automated tests



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

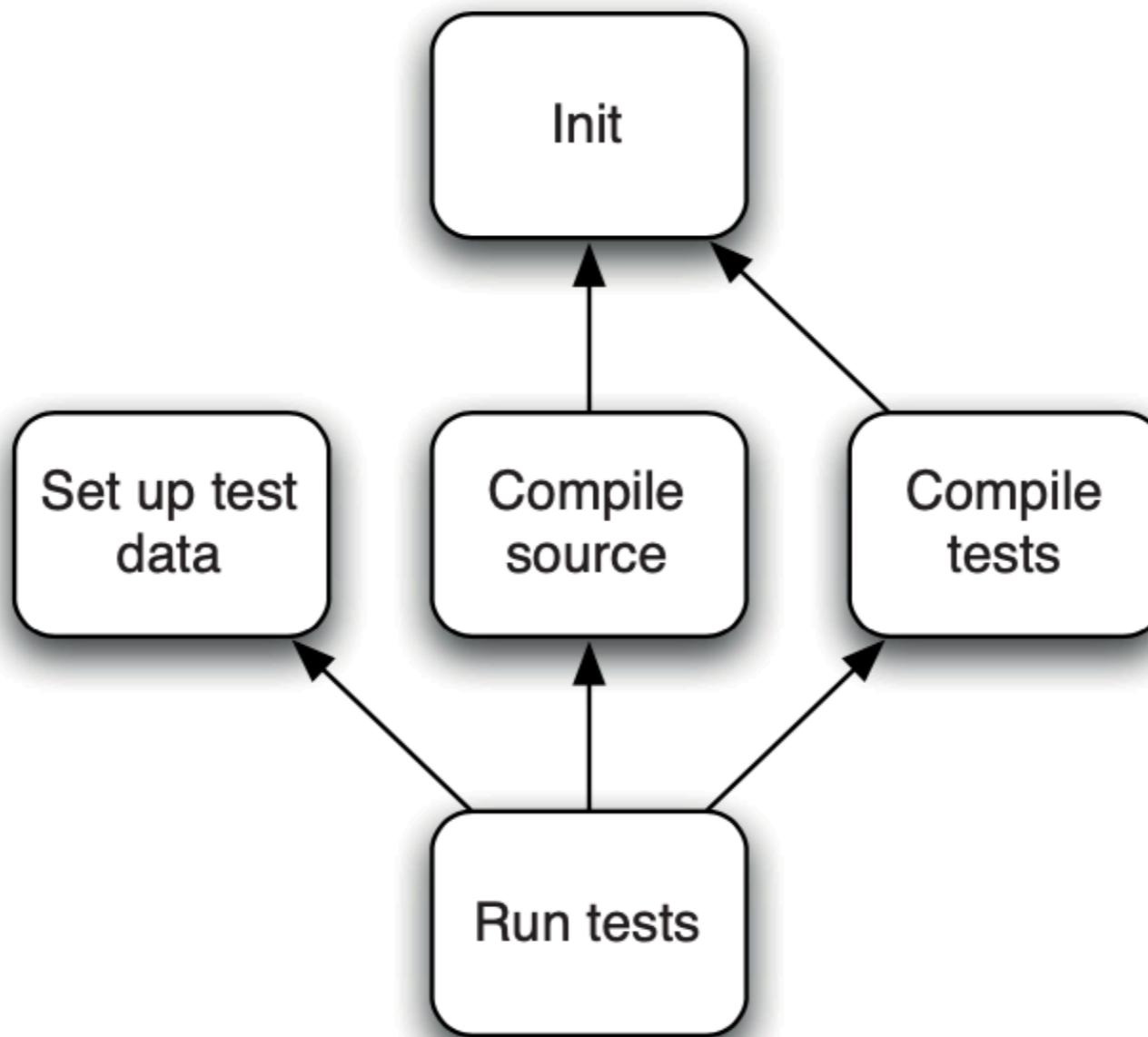
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Build and Deployment Scripting



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

# An Overview of Build Tools



**Figure 6.1** *A simple build dependency network*



# Principles and Practices of Build and Deployment Scripting

- Create a Script for Each Stage in Your Deployment Pipeline
- Use an Appropriate Technology to Deploy Your Application
- Use the Same Scripts to Deploy to Every Environment
- Use Your Operating System's Packing Tools
- Ensure the Deployment Process Is Idempotent
- Evolve Your Deployment System Incrementally



# Project Structure for Applications That Target the JVM

- Project Layout
- Managing Source Code
- Managing Tests
- Managing Build Output
- Managing Libraries



# Project Structure for Applications That Target the JVM

```
/[project-name]
  README.txt
  LICENSE.txt
  /src
    /main
      /java      Java source code for your project
      /scala     If you use other languages, they go at the same level
      /resources Resources for your project
      /filters   Resource filter files
      /assembly  Assembly descriptors
      /config    Configuration files
      /webapp    Web application resources
    /test
      /java      Test sources
      /resources Test resources
      /filters   Test resource filters
      /site      Source for your project website
      /doc       Any other documentation
  /lib
    /runtime   Libraries your project needs at run time
    /test      Libraries required to run tests
    /build     Libraries required to build your project
```



Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Deploying and Testing Layers

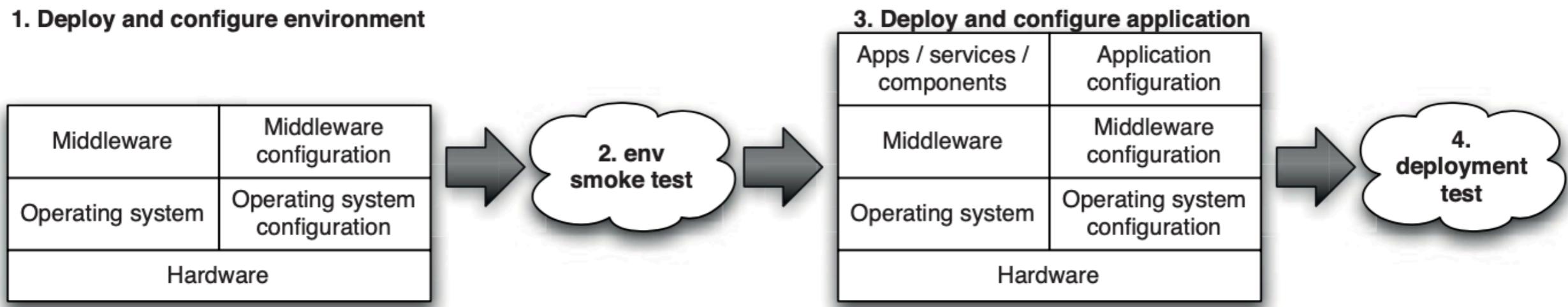


Figure 6.3 *Deployment testing layers*



# Tips and Tricks

- Always Use Relative Paths
- Eliminate Manual Steps
- Build In Traceability from Binaries to Version Control
- Test Target Should Not Fail The Build
- Constrain Your Application with Integrated Smoke Tests
- .NET Tips a Tricks ( Clean before Build )



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# The Commit Stage



# The Commit Stage

- Compiling Code and Run Test
- Create the Binary File for Later Stages
- Performing any Analysis
- Creating any Other Artifacts
- Provide Rapid Feedback



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# The Commit Stage

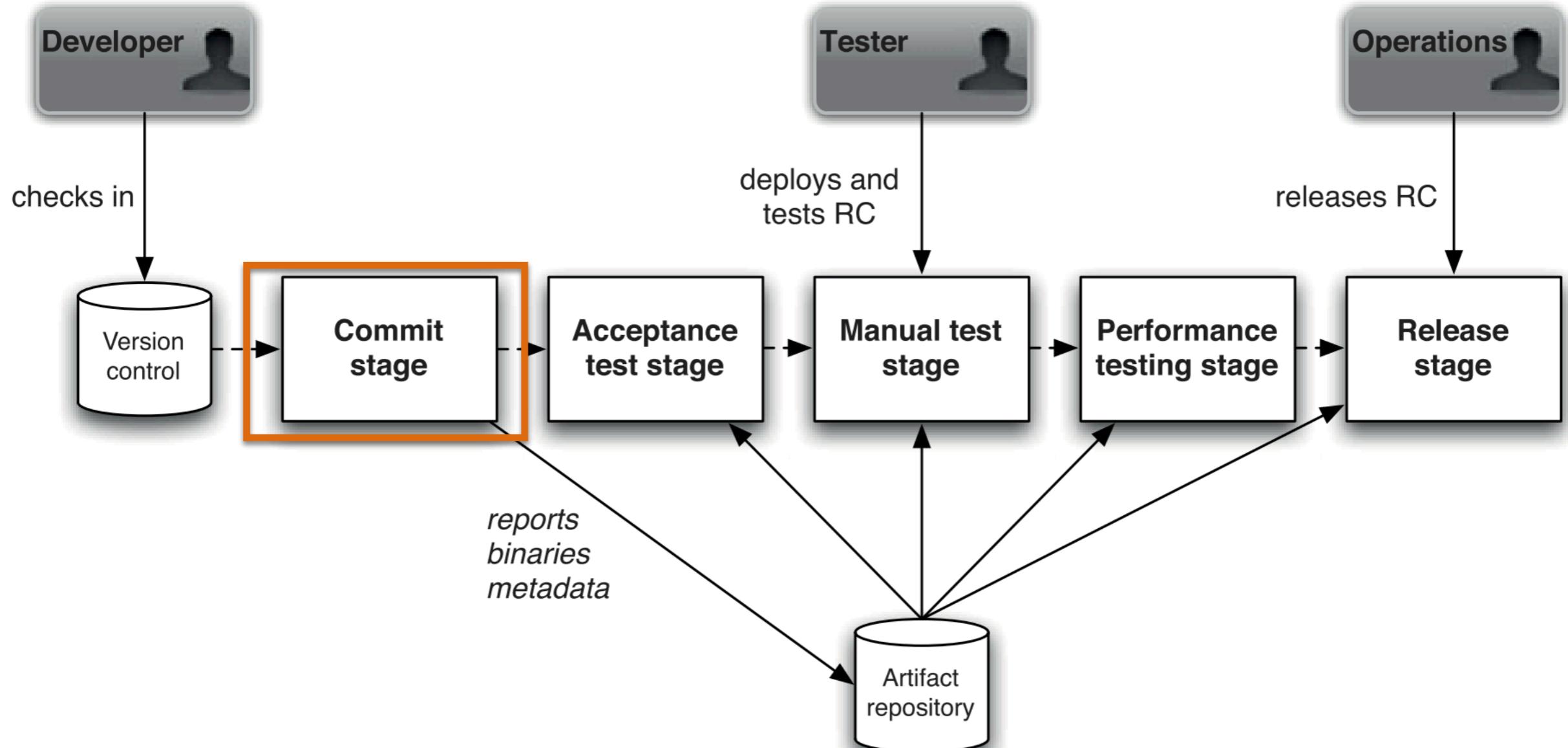


Figure 7.2 *The role of the artifact repository*



# The Commit Stage

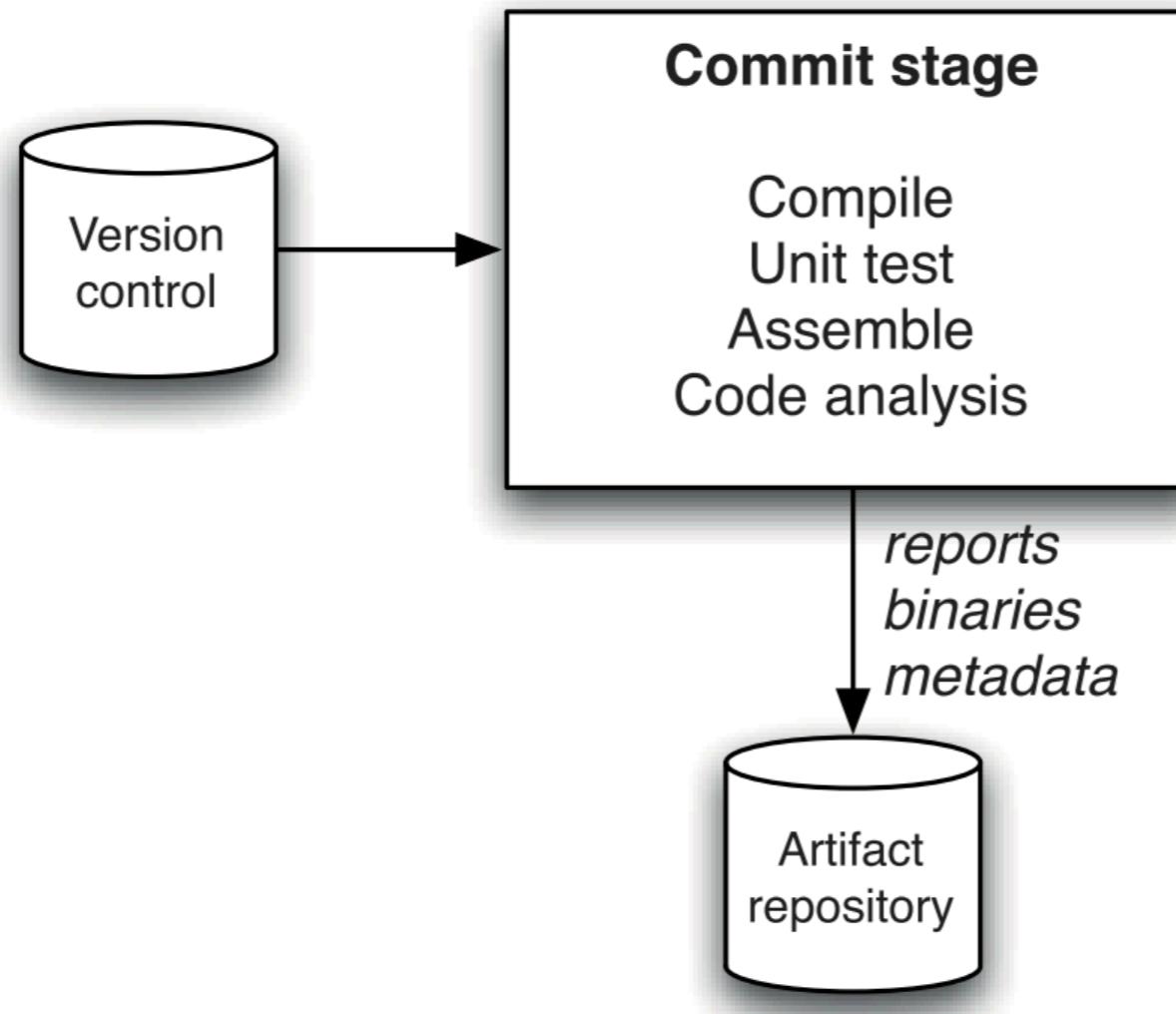


Figure 7.1 *The commit stage*



# Commit Stage Principles and Practices

- What Should Break the Commit Stage ?
- Tend the Commit Stage Carefully
- Give Developers Ownership
- Use a Build Master for Very Large Teams



# The Artifact Repository

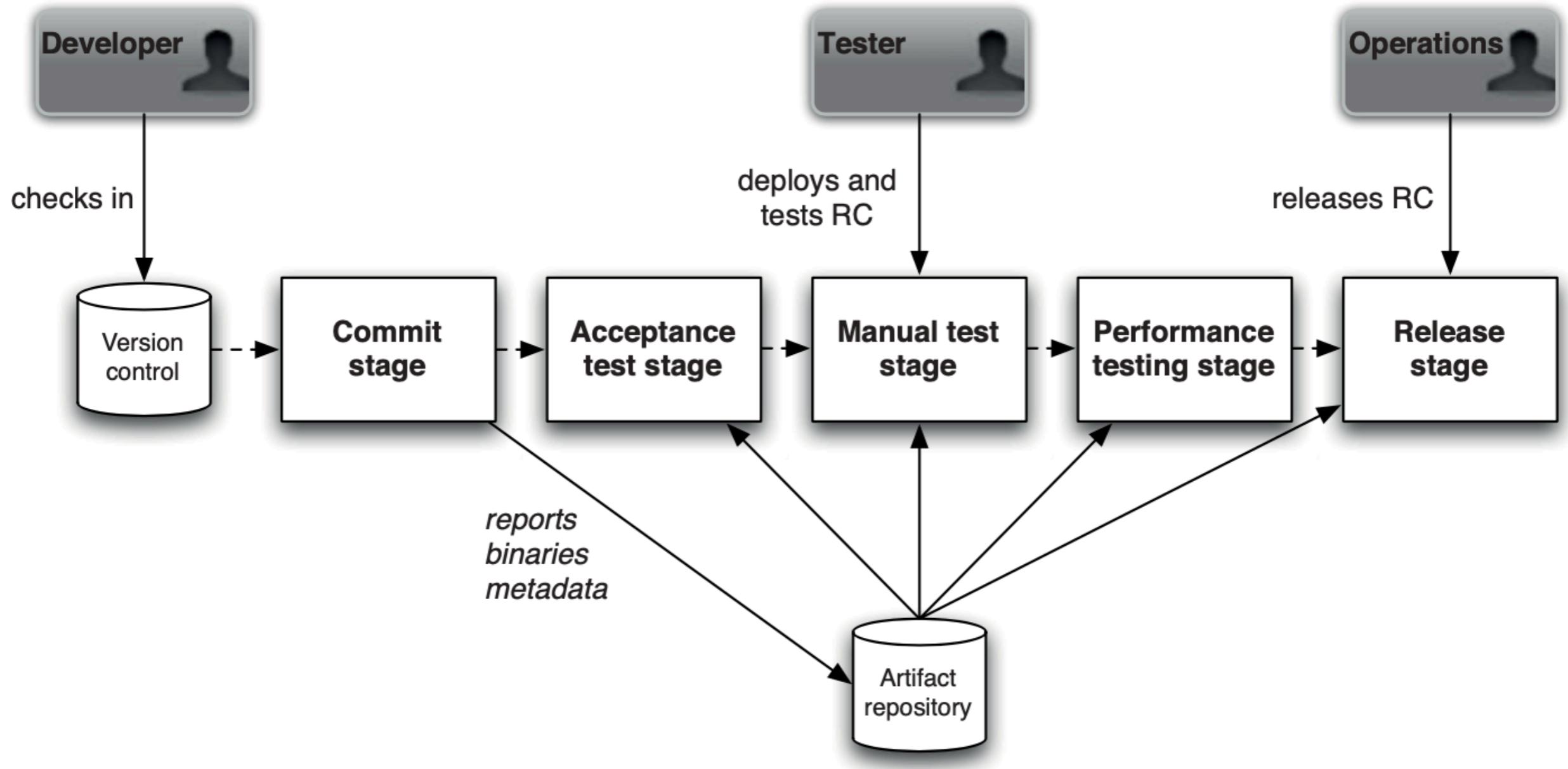
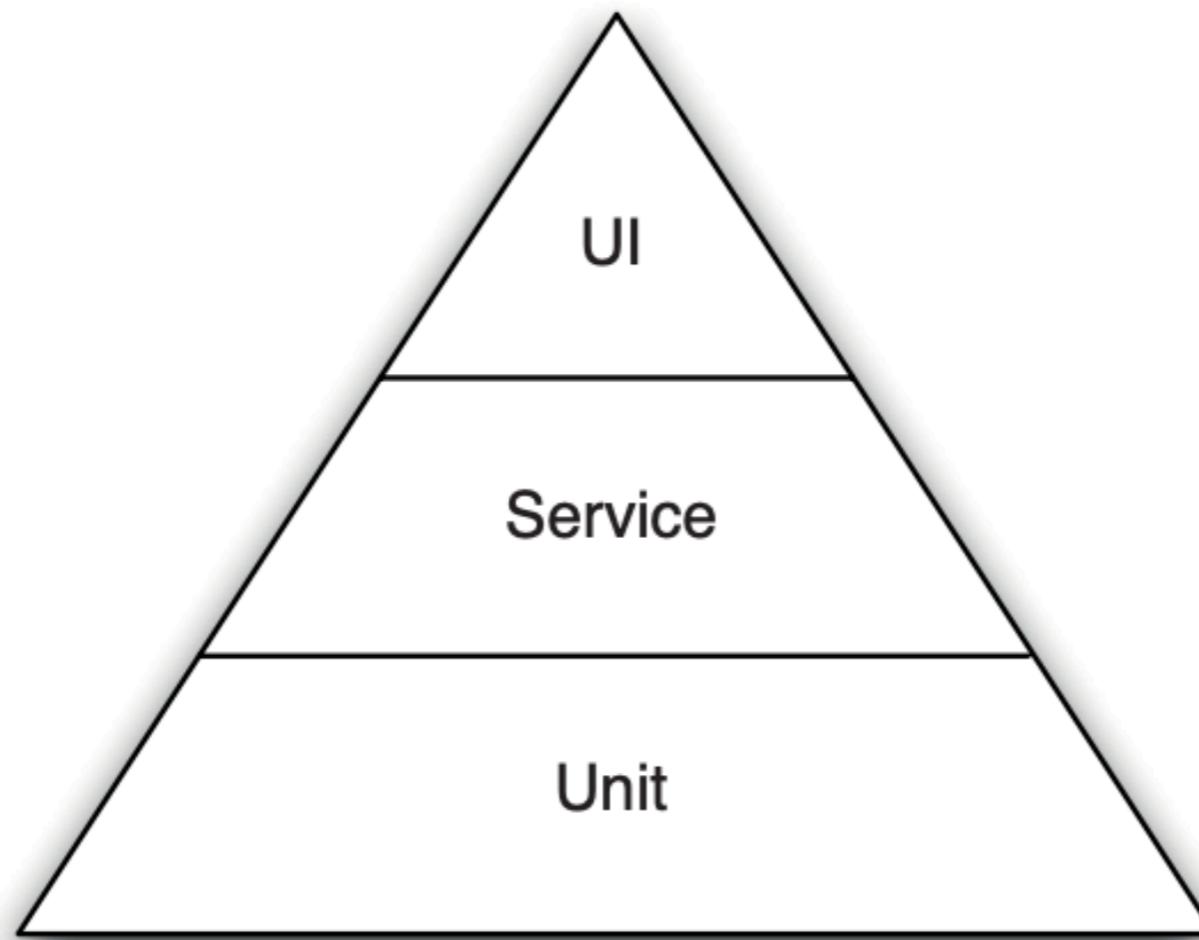


Figure 7.2 *The role of the artifact repository*



# Test Automation Pyramid



**Figure 7.3** *Test automation pyramid (Cohn, 2009, Chapter 15)*



# Commit Test Suite Principles and Practices

- Avoid the User Interface
- Use Dependency Injection
- Avoid the Database
- Avoid Asynchrony in Unit Tests
- Using Test Doubles
- Minimizing Stage in Tests
- Faking Time
- Brute Force



# Automate Acceptance Testing



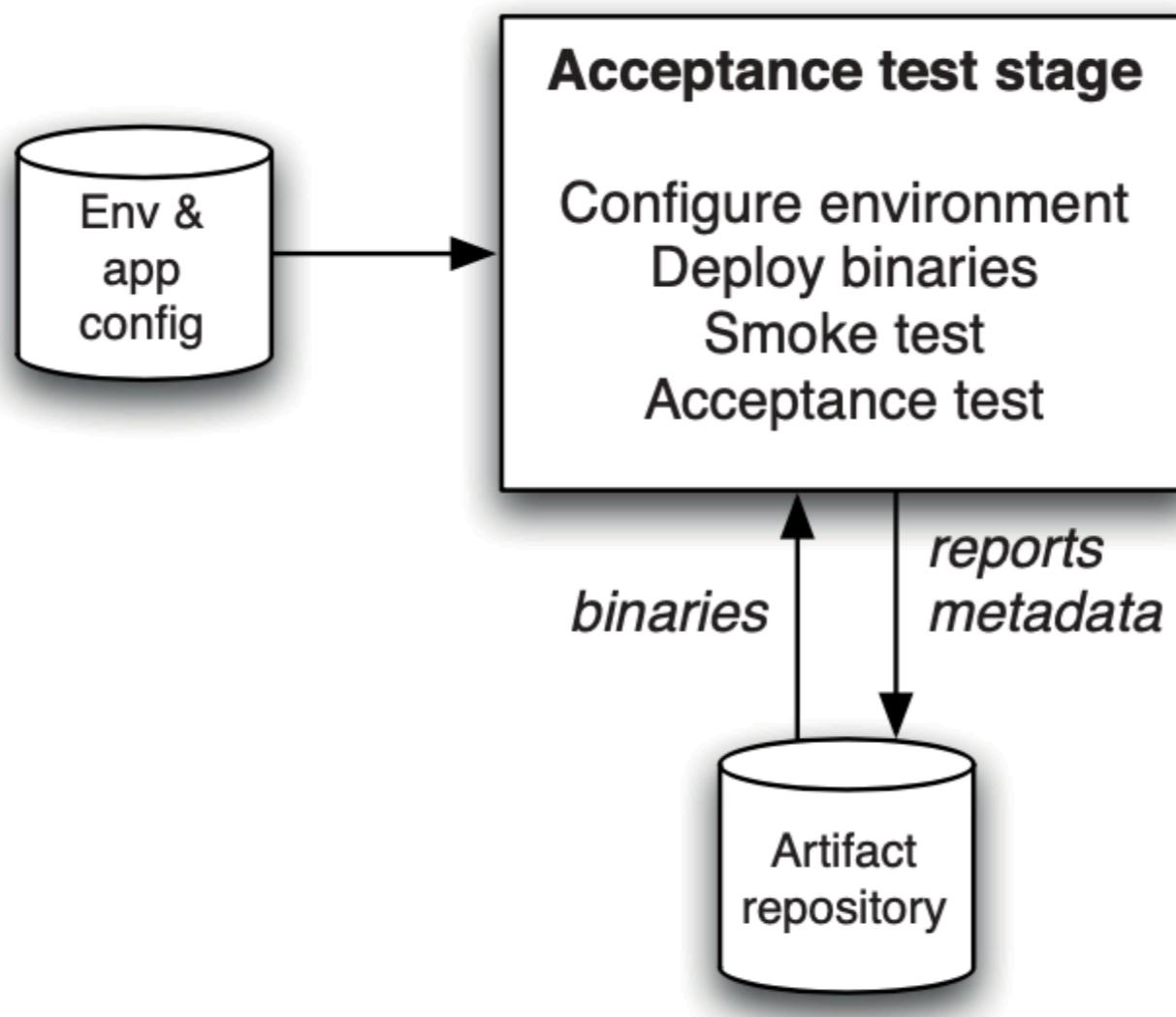
Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

# Acceptance Testing

“Once you have automate acceptance test in place, you are testing the **business acceptance criteria** of your application, that is, **validating** that it provides **users** with valuable **functionality**.”



# Acceptance Test Stage



**Figure 8.1** *The acceptance test stage*



# How to Create Maintainable Acceptance Test Suites

Follow the INVEST  
guidelines for good  
user stories!



# How to Create Maintainable Acceptance Test Suites

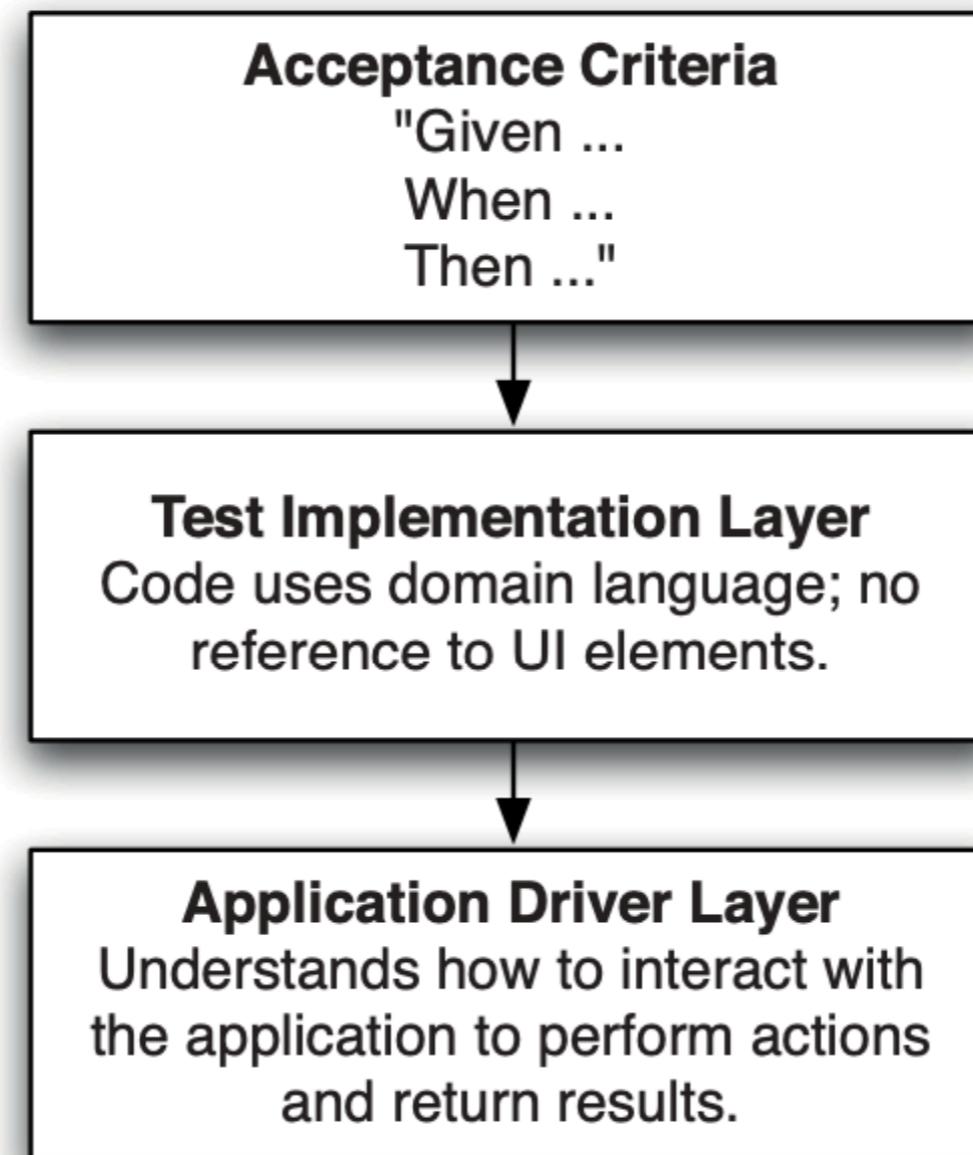


Figure 8.2 *Layers in acceptance tests*



# **Creating Acceptance Tests**

- 1. Tester and BA working with customer to define acceptance criteria for requirements**
- 2. Tester and BA working with developer to create automated acceptance tests**
- 3. Tester and BA Perform manual test  
( Exploratory , Acceptance , Showcase )**



# Acceptance Criteria as Executable Specifications

Feature: Placing an order

Scenario: User order should debit account correctly

Given there is an instrument called bond

And there is a user called Dave with 50 dollars in his account

When I log in as Dave

And I select the instrument bond

And I place an order to buy 4 at 10 dollars each

And the order is successful

Then I have 10 dollars left in my account



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Implementing Acceptance Tests

- Minimize the states in acceptance tests
- Process boundaries , encapsulation and testing
- Managing asynchrony and timeouts
- Using test double
- Keeping acceptance tests green
- Deployment test
- Refactor common task



# Testing Nonfunctional Requirements



# Performance and Throughput

**Performance** is a measure of the time taken to process a single transaction, and can be measured either in isolation or under load.

**Throughput** is the number of transactions a system can process in a given timespan. It is always limited by some bottleneck in the system (**Capacity**)



# Programming for Capacity

The point of the capacity testing stage is to tell us whether we have a problem, so that we can go on to fix it. Don't guess



# Strategy is Address Capacity Problems

1. Decide upon an architecture for your application
2. Understand and use patterns and avoid antipatterns
3. Keep team working within architecture but can apply appropriate
4. Pay attention to data structures and algorithms
5. Be extremely careful about threading
6. Establish automate test for capacity
7. Use profiling for attempt to fix problems
8. Wherever you can use real-world capacity measures



Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Measuring Capacity

Scalability testing.

Longevity testing.

Throughput testing.

Load testing.



Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Automating Capacity Testing

During development, what we need is the ability to assert that our application will achieve the capacity required by the customer.

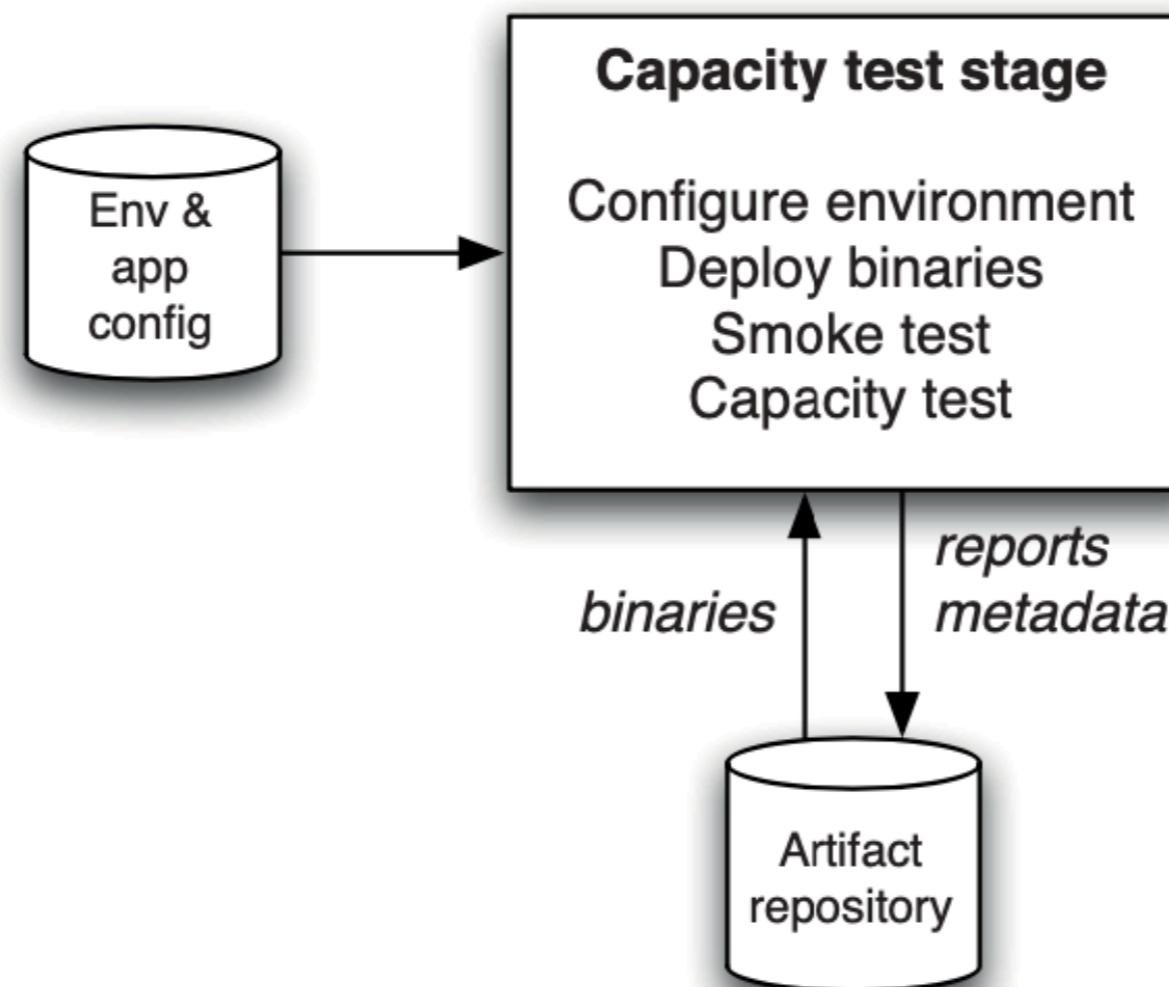


Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Automating Capacity Testing



**Figure 9.7** *The capacity test stage of the deployment pipeline*



# Additional Benefits of a Capacity Test System

- Detecting and debugging memory leaks
- Tuning application configuration parameters
- Tuning third-party application configuration, such as operating system, application server, and database configuration
- Measuring the scalability of the application over different hardware configurations



# Deploying and Releasing Applications



# Creating a Release Strategy

The point of their discussions should be working out a common understanding concerning the deployment and maintenance of the application throughout its lifecycle.



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

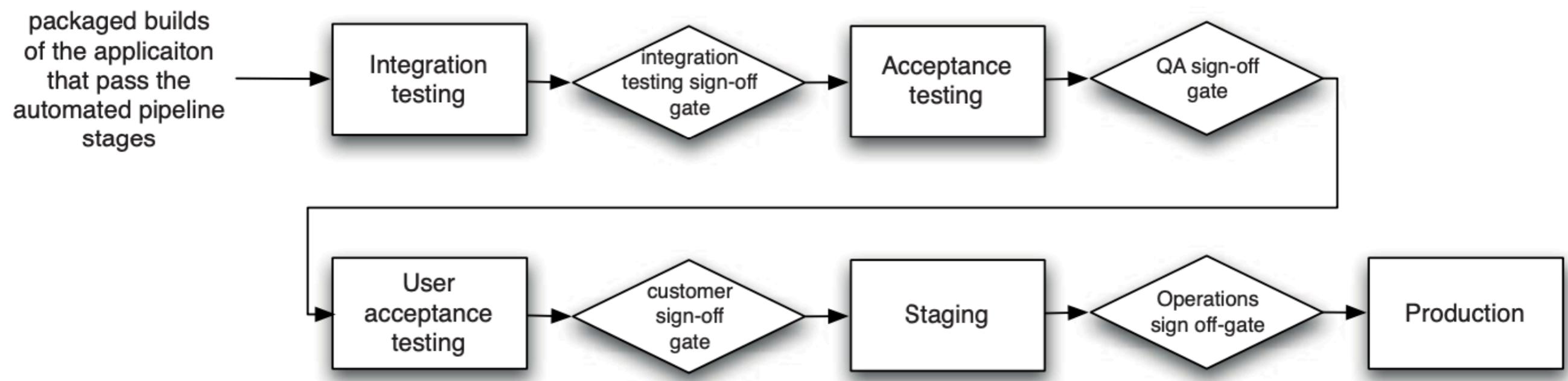
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Creating a Release Strategy

- Parties in charge of deployment in environment and release
- Plan for implement deployment pipeline
- Requirements for monitoring the application
- Description of the integration
- A disaster recovery plan
- The steps about back out the deployment



# Modeling Your Release Process and Promoting Builds

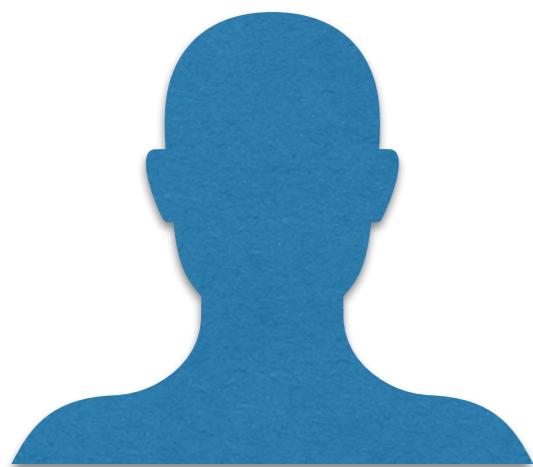


**Figure 10.1** *An example test and release process diagram*



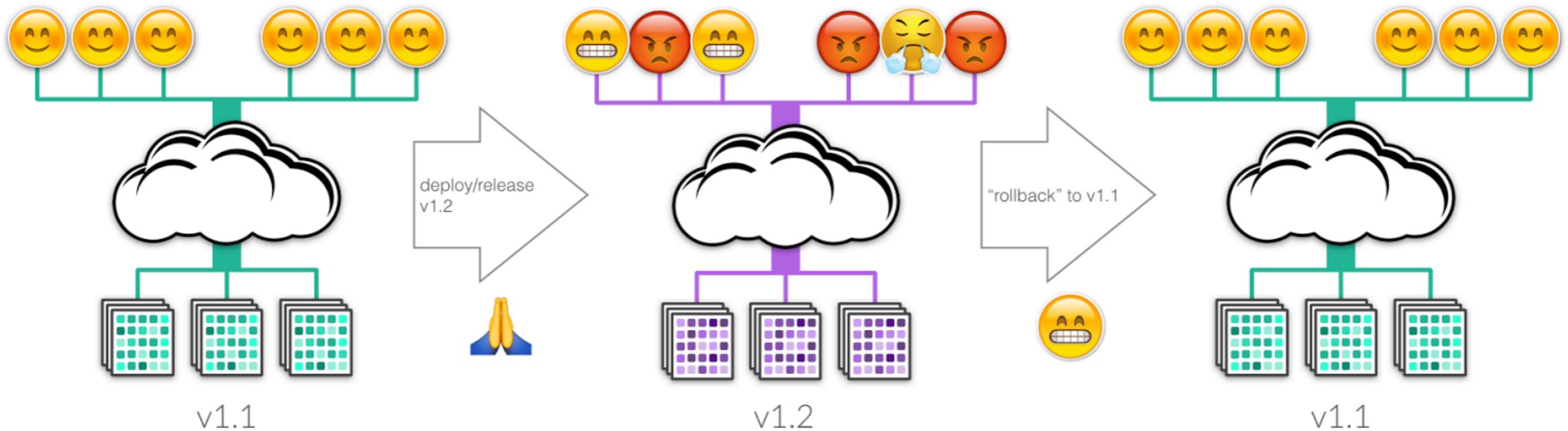
# Deployments to Staging Environments

“You should have started to put your staging environment together at the beginning of your project.”

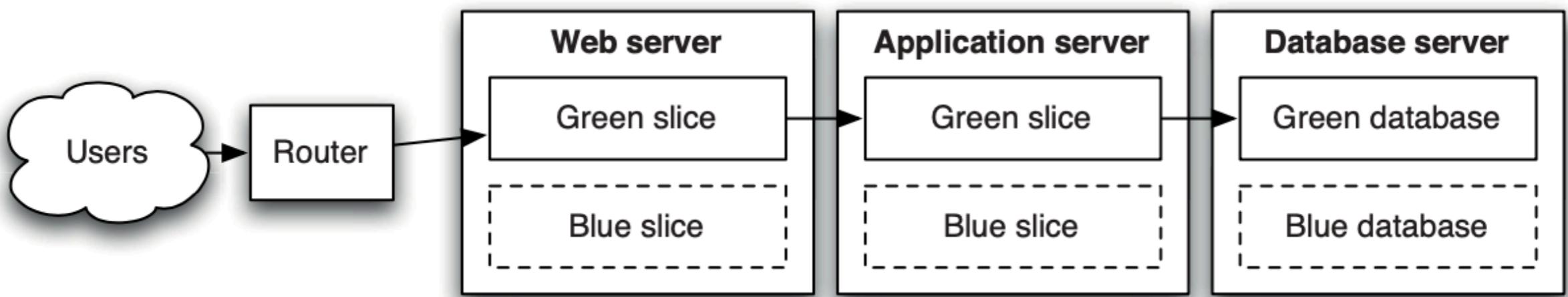


# Rolling Back Deployments

“It is essential to be able to roll back a deployment in case it goes wrong. Debugging problems in a running production environment is almost certain to result in late nights, mistakes with unfortunate consequences, and angry users.”



# Blue-Green Deployments



**Figure 10.2** *Blue-green deployments*



# Canary Releasing

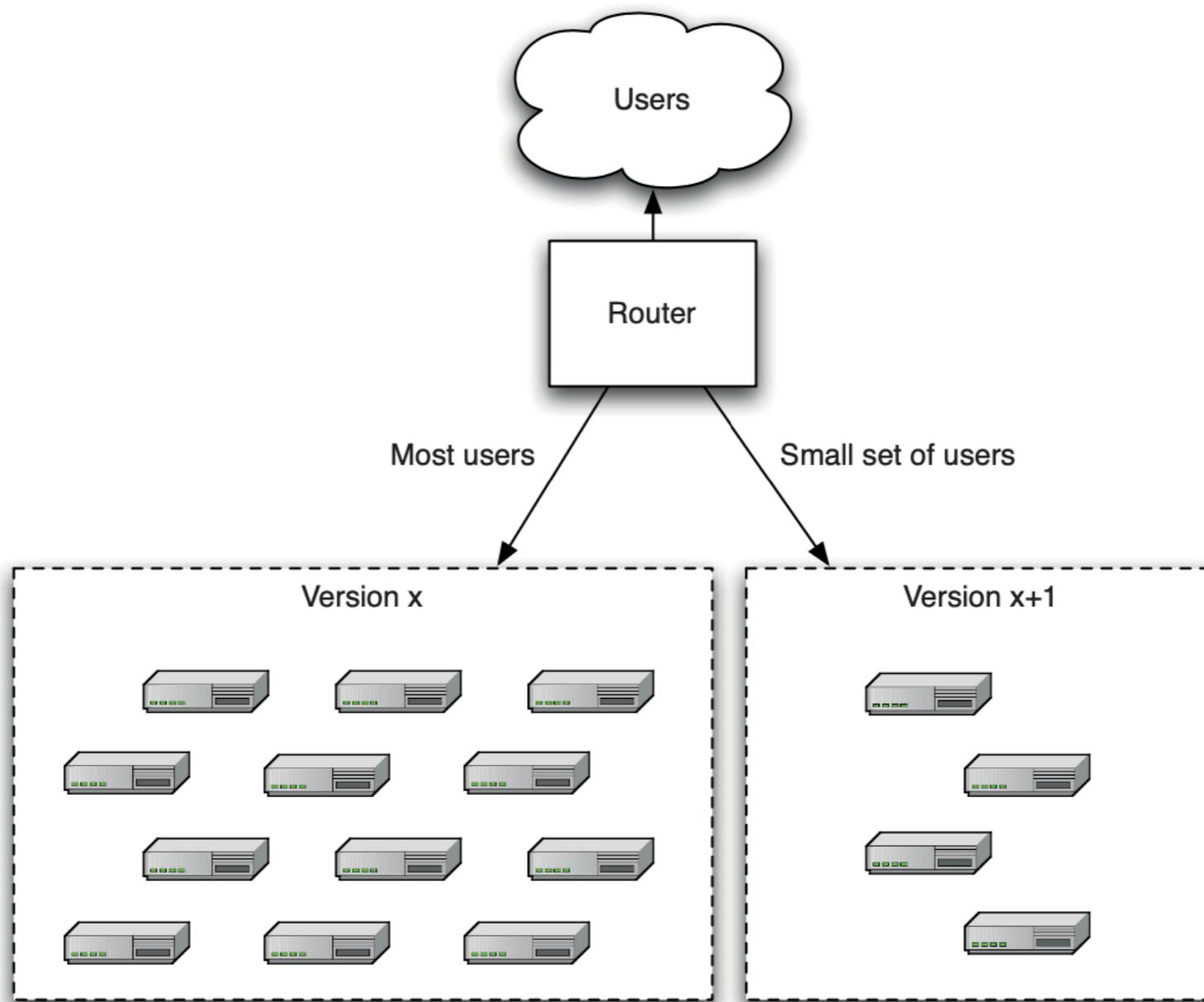
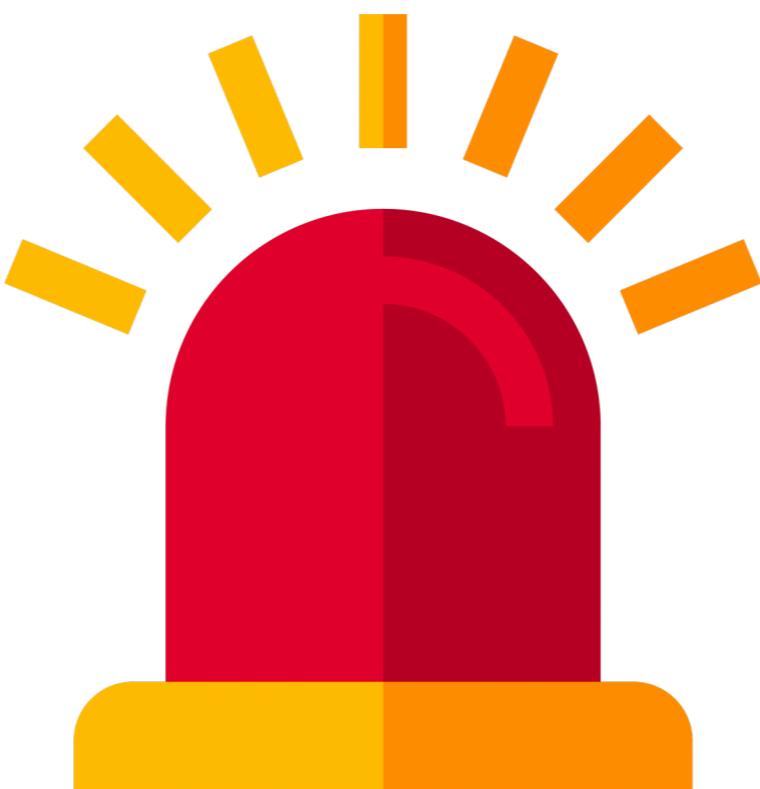


Figure 10.3 *Canary releasing*



# Emergency Fixes

“Emergency fixes have to go through the same build, deploy, test, and release process as any other change.”



# Managing Infrastructure and Environments



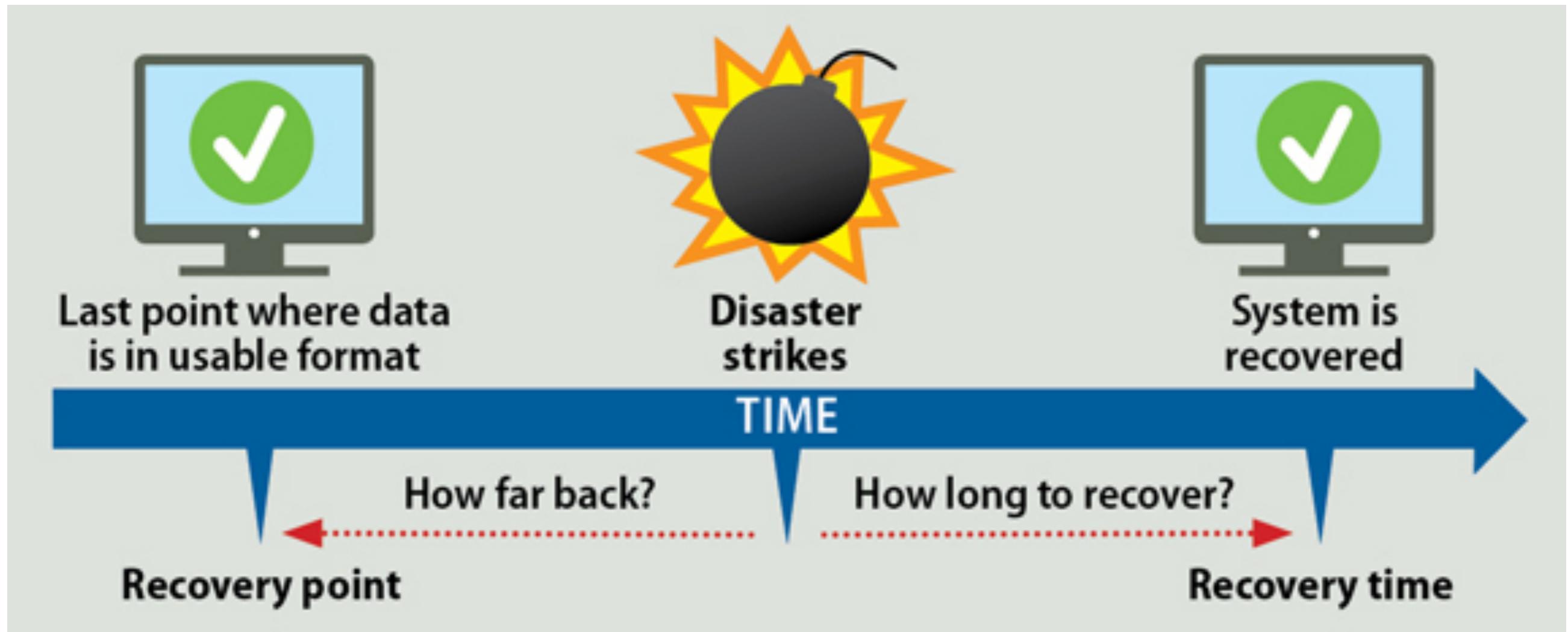
Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

# Understanding the Needs of the Operation Team

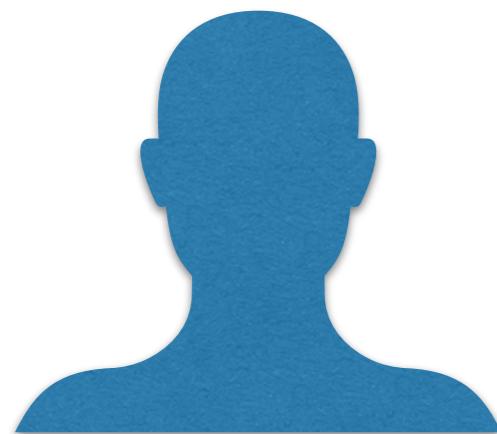
- Documentation and Auditing
- Alters for Abnormal Events
- IT Service Continuity Planning
- Use the Technology the Operations Team Is Familiar With



# Recovery Point Objective & Recovery Time Objective



# Modeling and Managing Infrastructure



- How will we provision our infrastructure?
- How will we deploy and configure the various bits of software that form part of our infrastructure?
- How do we manage our infrastructure once it is provisioned and configured?



# Virtualization

“Virtualization provides a way to amplify the benefits of the techniques, already described in this chapter, for automating the provisioning of servers and environments”



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Virtualization

## Benefits :

- **Fast Response to Change Requirement**

Can be provisioned new machine in seconds , no cost

- **Consolidation**

More efficient at team service and hardware

- **Standardizing Hardware**

No longer distinct hardware configuration

- **Easier-To-Maintain Baselines**

Application stack maintain in a image



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Monitoring Infrastructure and Applications

- To find out how much revenue
- Detecting something goes wrong
- Historical data for planning



# Collecting Data

Hardware : Voltages , Temperatures

Operating System : Memory Usage , Disk Space , I/O

Middleware : Database Connection Pool

Application : Number of Transactions



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Creating Dashboards

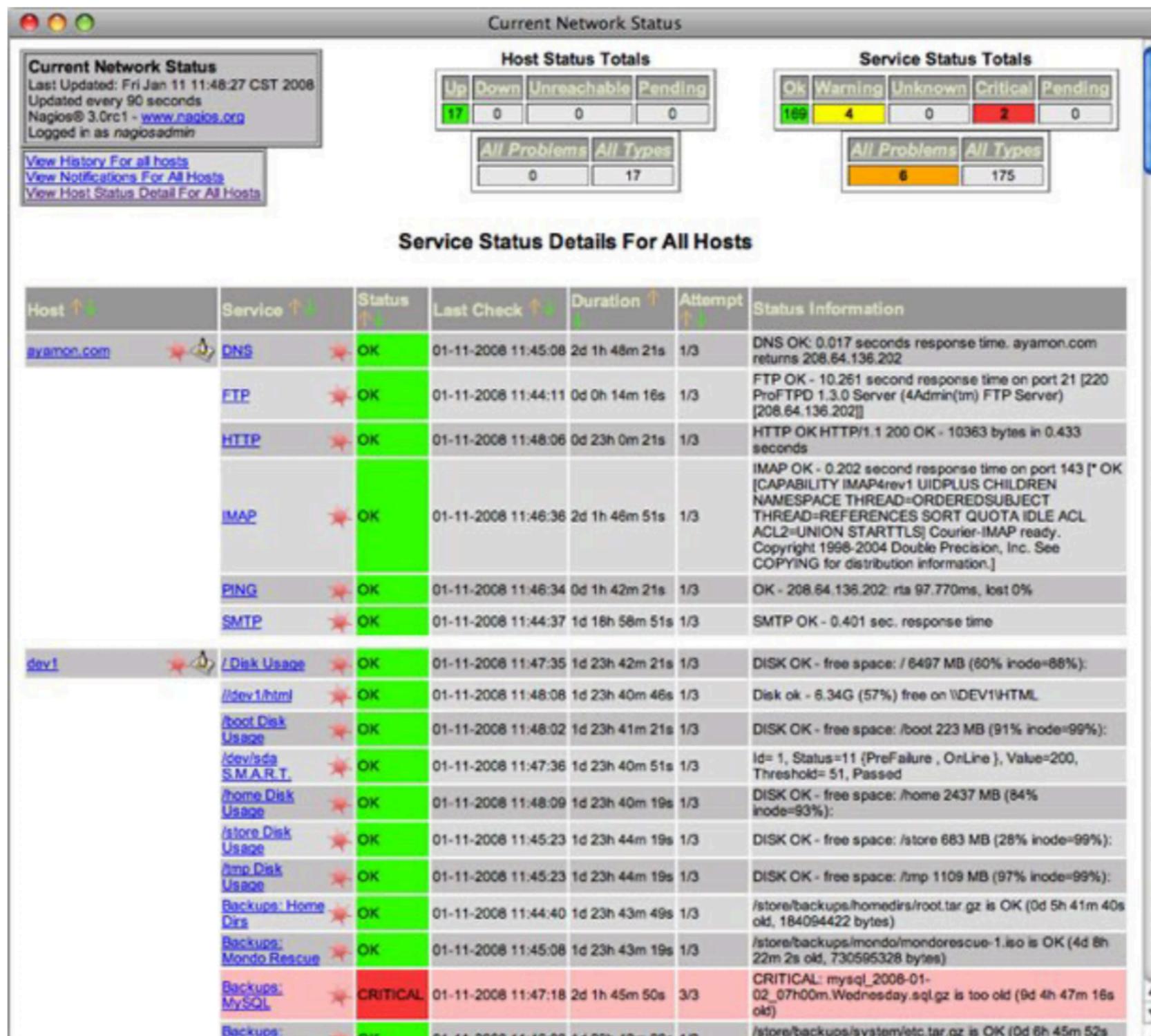


Figure 11.10 Nagios screenshot



Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material.

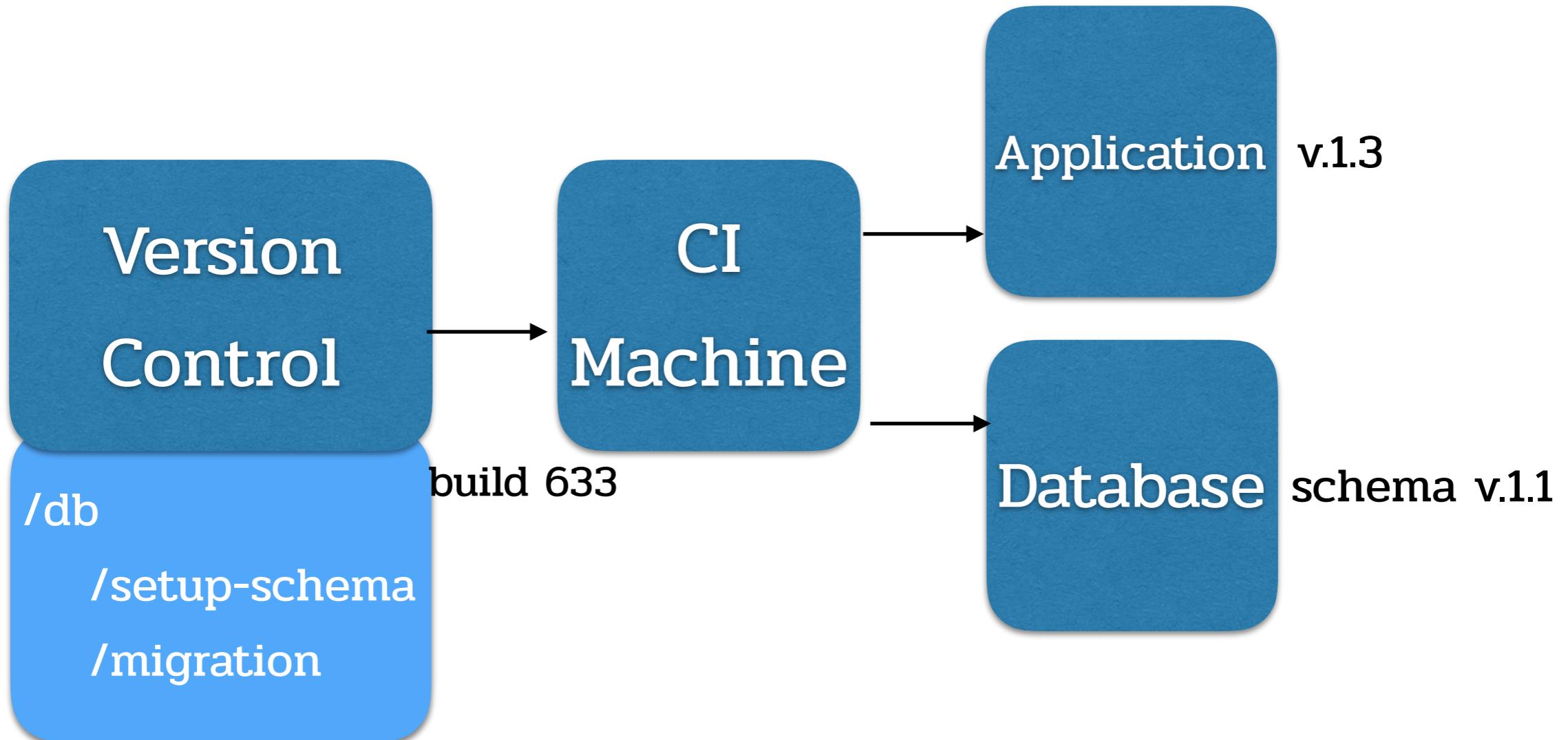
NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Managing Data



# Database Scripting



# Incremental Change: Versioning Your Database

## 1\_create\_initial\_database.sql

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

## 2\_add\_customer\_date\_of\_birth.sql

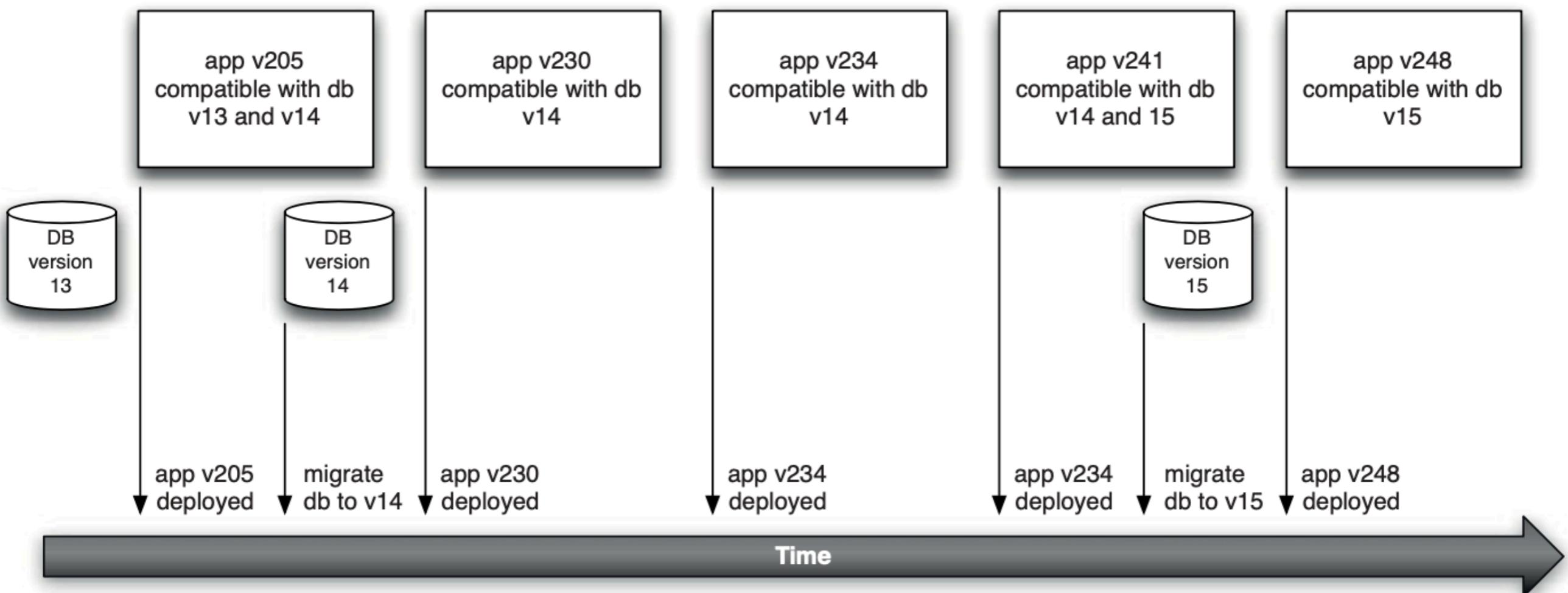
```
ALTER TABLE customer ADD COLUMN dateofbirth DATETIME;

--//@UNDO

ALTER TABLE customer DROP COLUMN dateofbirth;
```



# Decoupling Application Deployment from Database Migration



**Figure 12.1** *Decoupling database migration from application deployment*

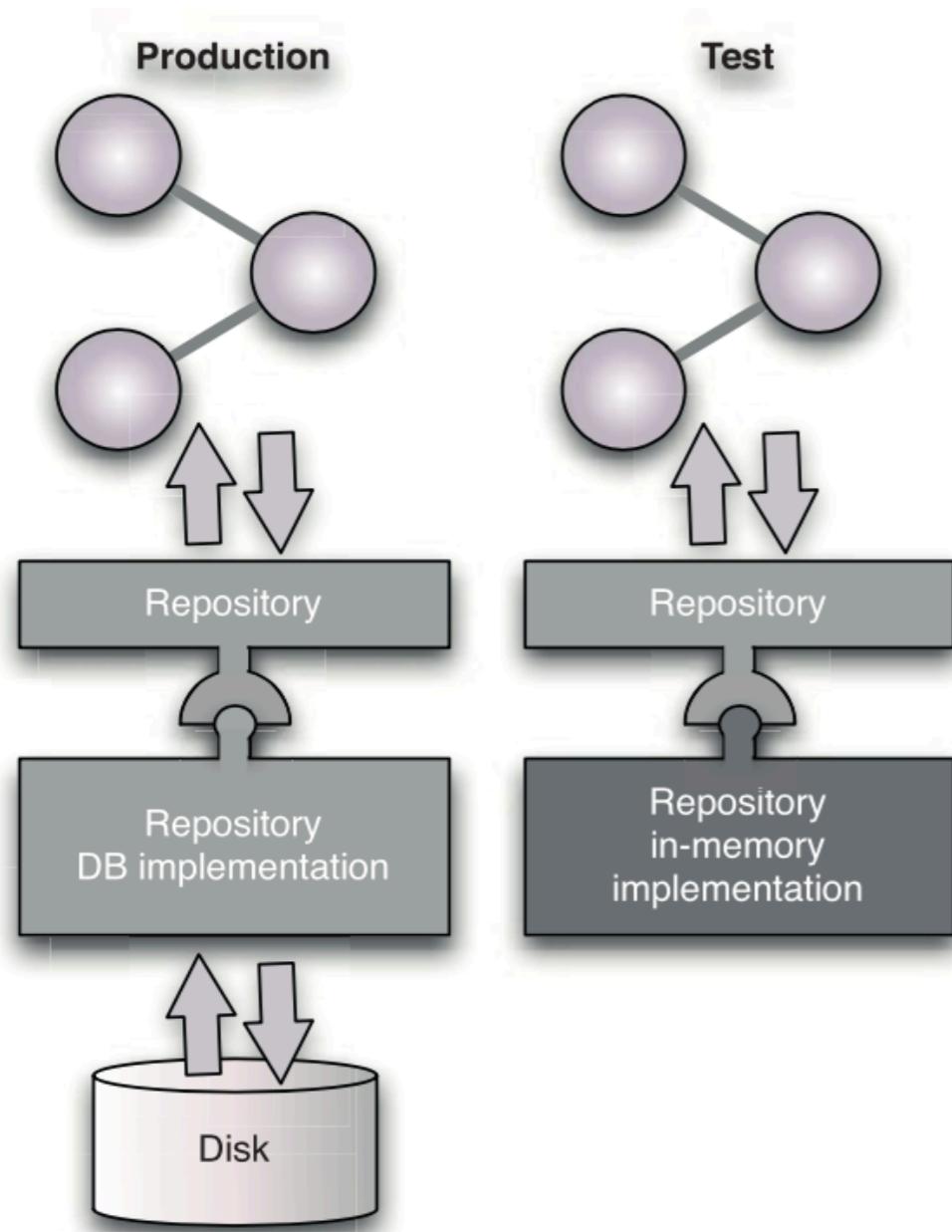


# Managing Test Data

- Faking the Database for Unit Test
- Managing the Coupling between Tests and Data
- Test Isolation
- Setup and Tear Down
- Coherent Test Scenarios



# Managing Test Data : Faking the Database for Unit Test



- Don't run test against a real database
- Replace with a test double
- Still possible to use fake database

Figure 12.2 Abstracting database access



Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Data Management and the Deployment Pipeline

- **Data in Commit Stage Tests**
  - Use test double , Avoid elaborate data setup
- **Data in Acceptance Tests**
  - Test-specific , Test reference , Application Reference
- **Data in Capacity Tests**
  - Scale acceptance test with data
- **Data in Other The Stages**
  - Specific data for specific objective



# Managing Components and Dependencies



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

# Keeping Your Application Releaseable

- Hide new functionality until it is finished.
- Make all changes incrementally as a series of small changes, each of which is releasable.
- Use branch by abstraction to make large-scale changes to the codebase.
- Use components to decouple parts of your application that change at different rates.



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Dependencies : Libraries and Components

## Libraries

- Out of control
- Rarely update

## Components

- Organization controlled
- Frequently update



Share — copy and redistribute the material in any medium or format.

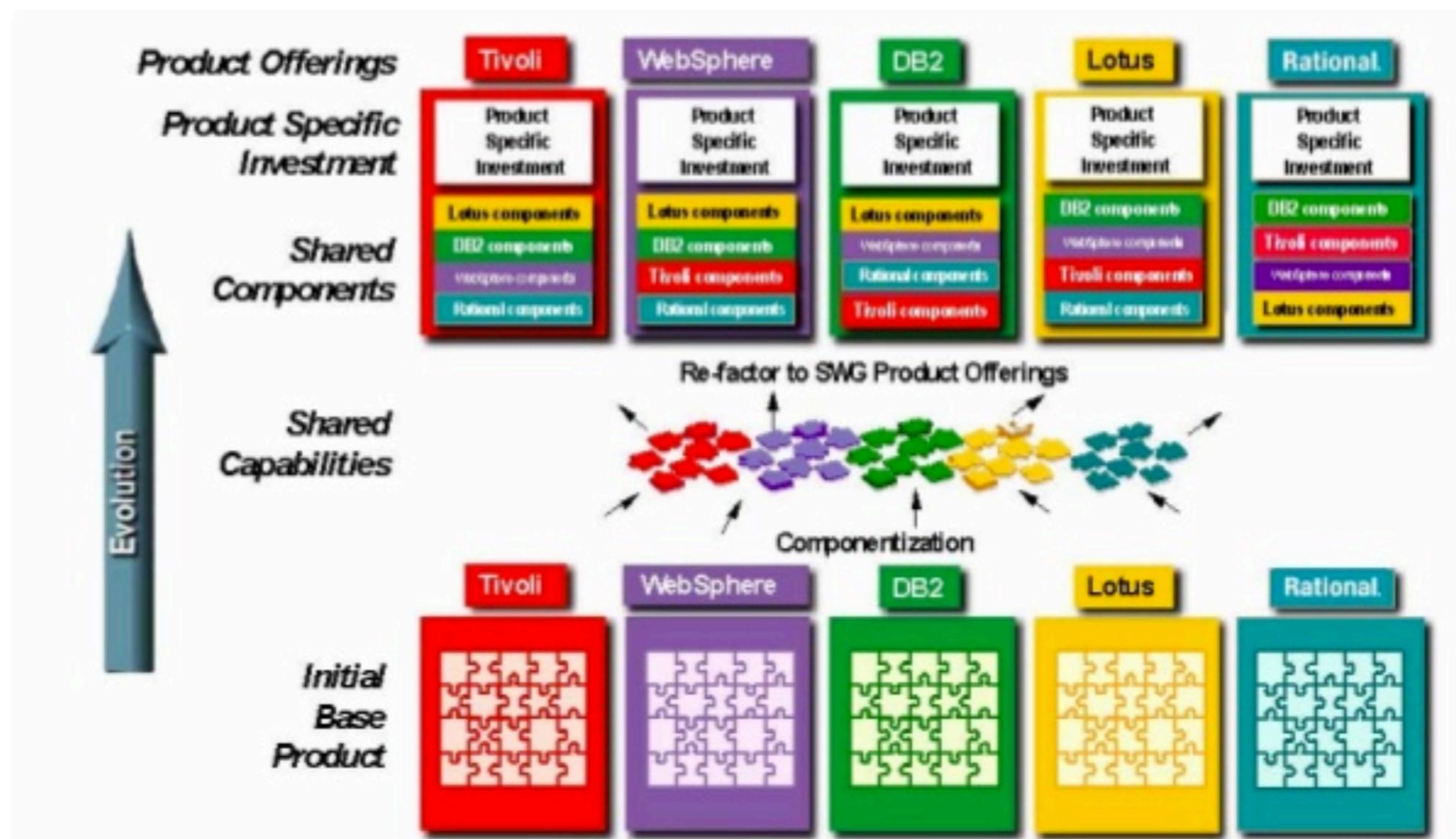
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# How to Divide a Codebase into Components

## Componentization: Sharing capabilities



IBM

11

Source: The Benefits of an Open Service Oriented Architecture in the Enterprise - Craig Hayman, IBM Software Group



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

112

# The Integration Pipeline

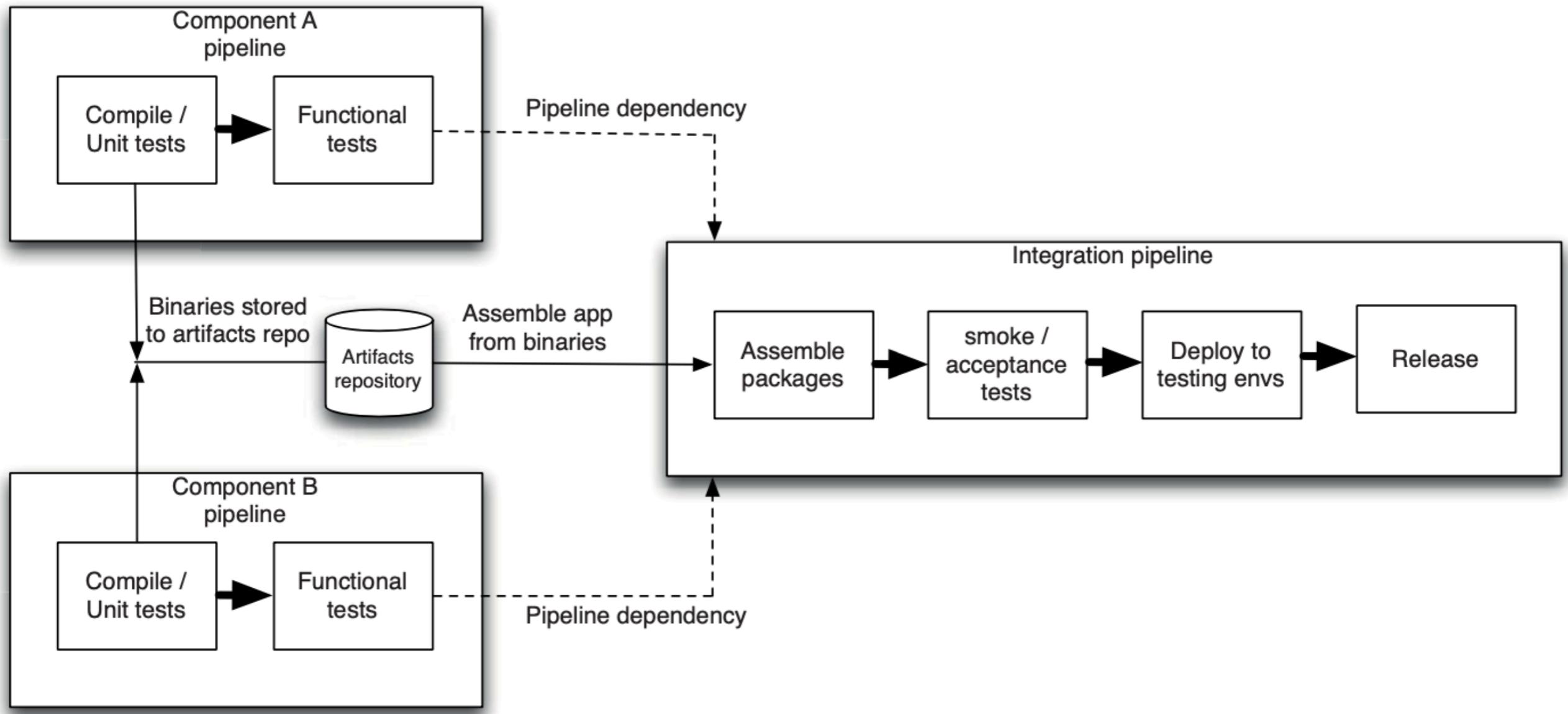


Figure 13.1 *Integration pipeline*



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Building Dependency Graphs

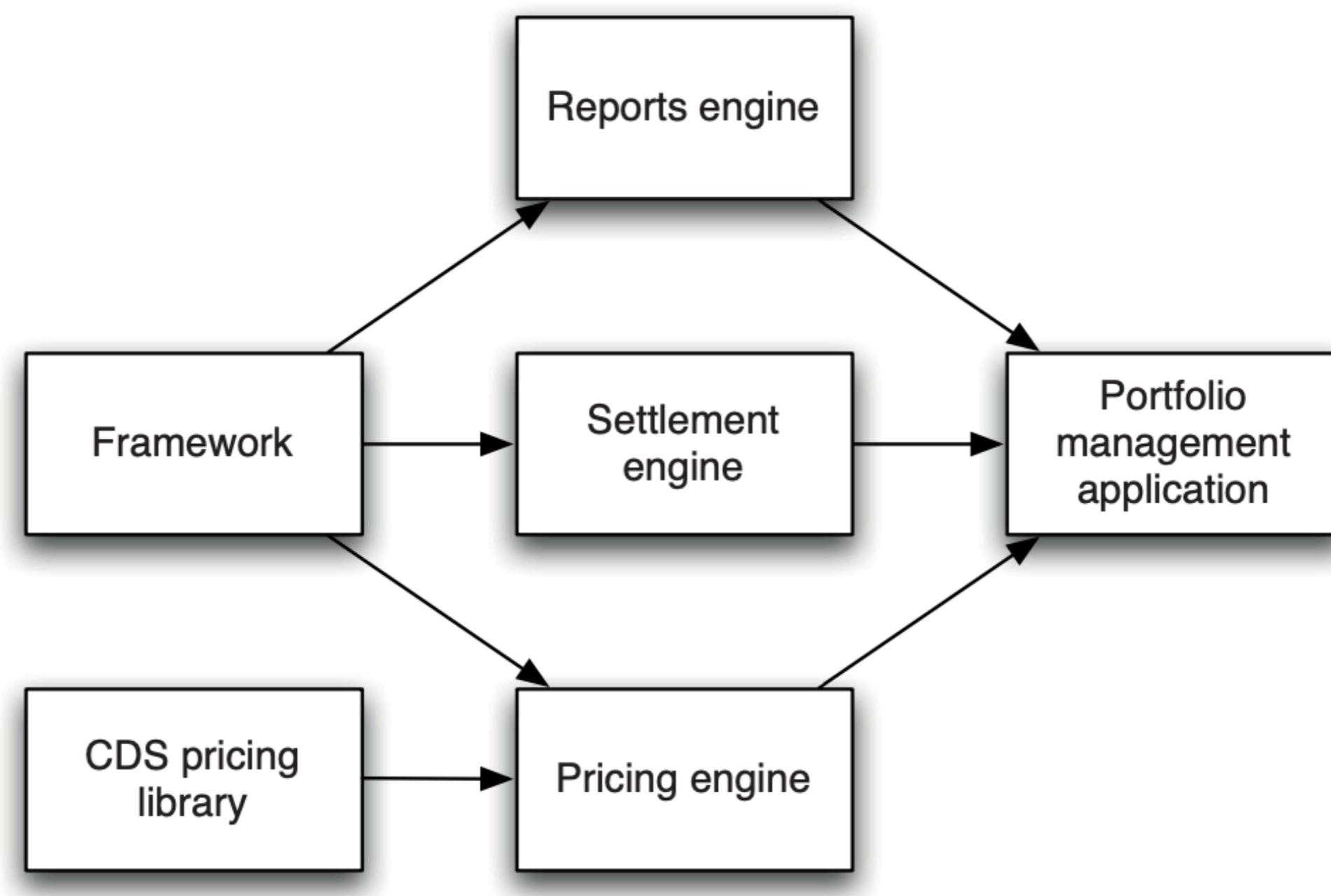


Figure 13.2 *A dependency graph*



# Pipelining Dependency Graphs

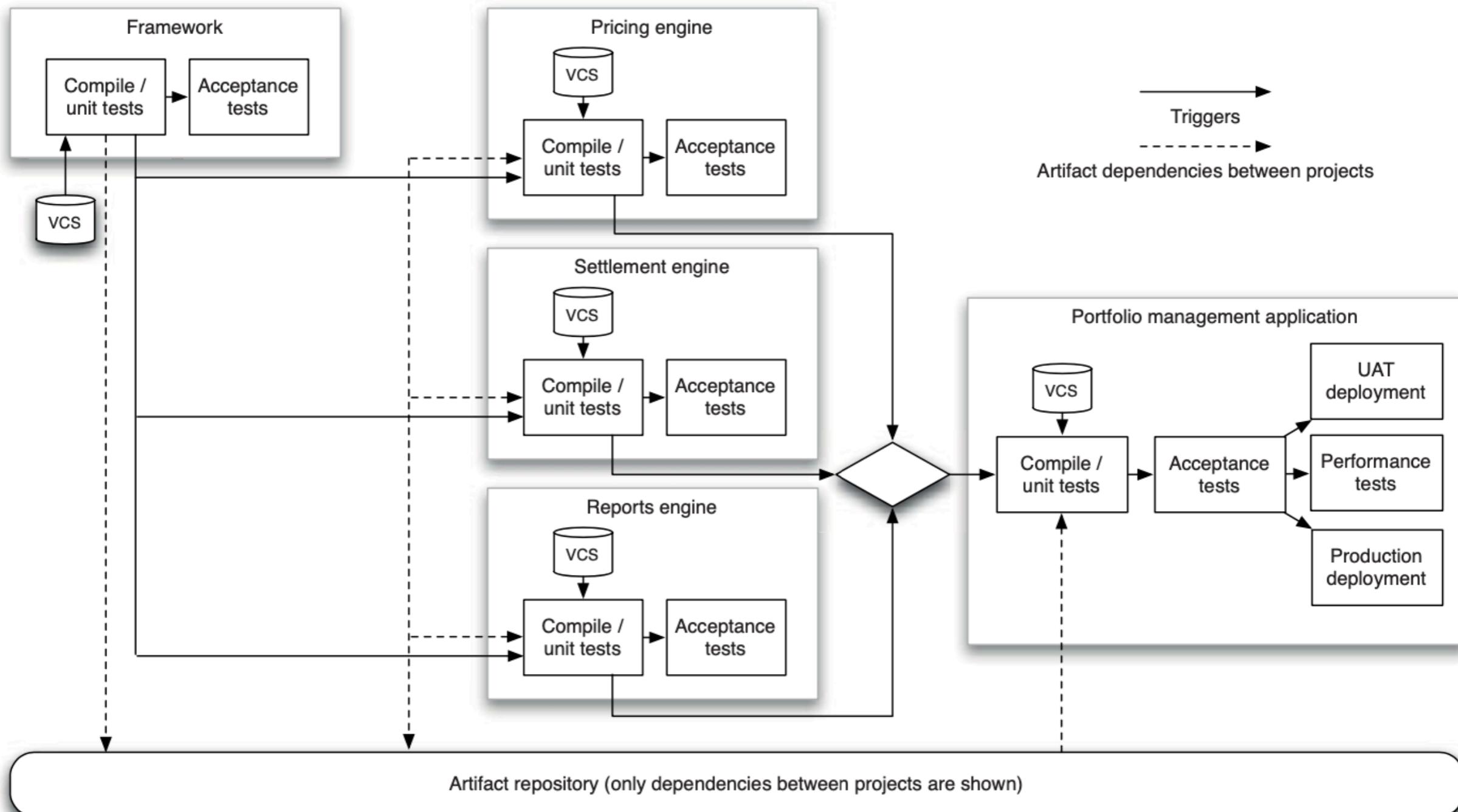


Figure 13.3 *Component pipeline*



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Pipelining Dependency Graphs

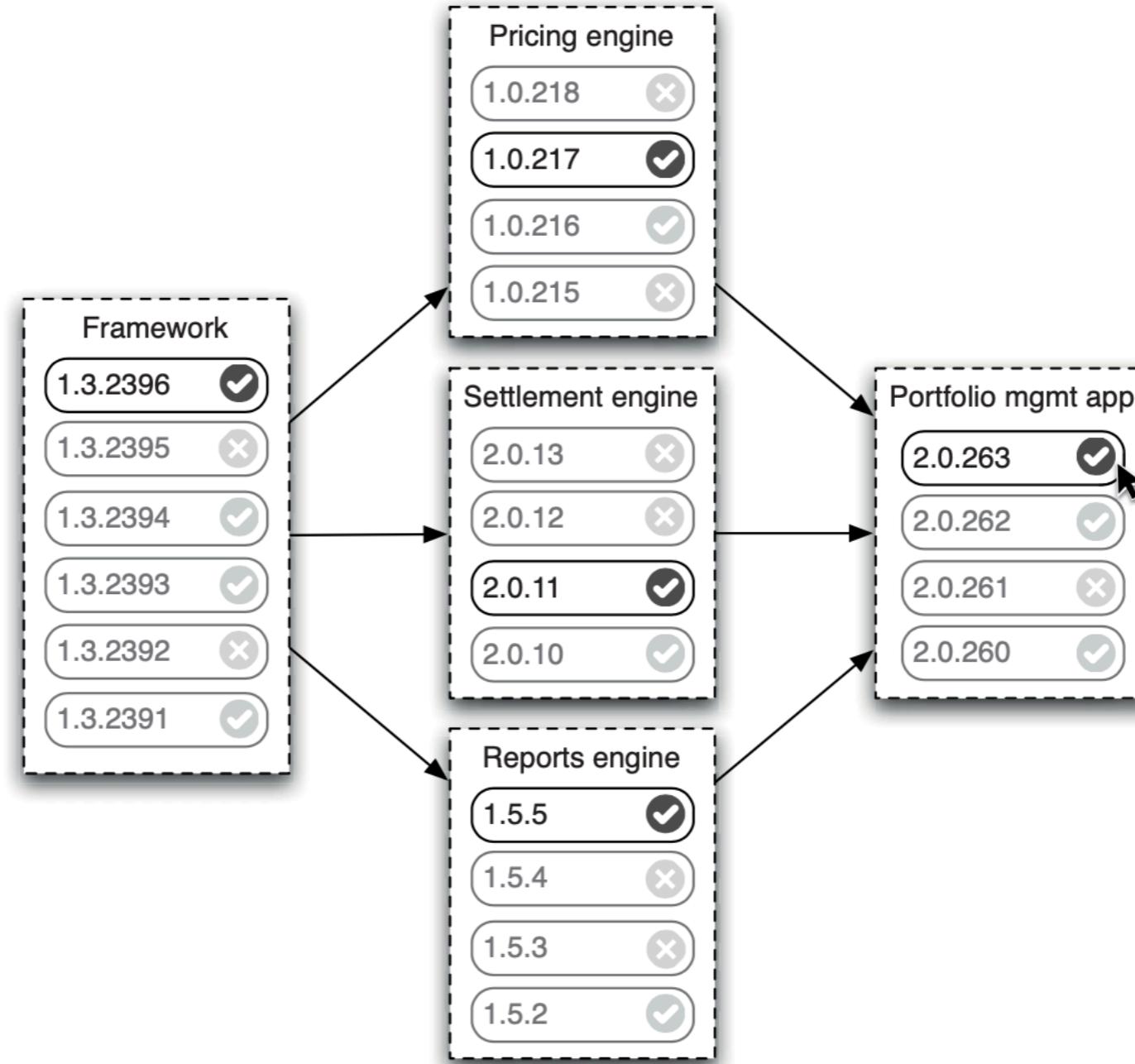


Figure 13.4 Visualizing upstream dependencies



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Pipelining Dependency Graphs

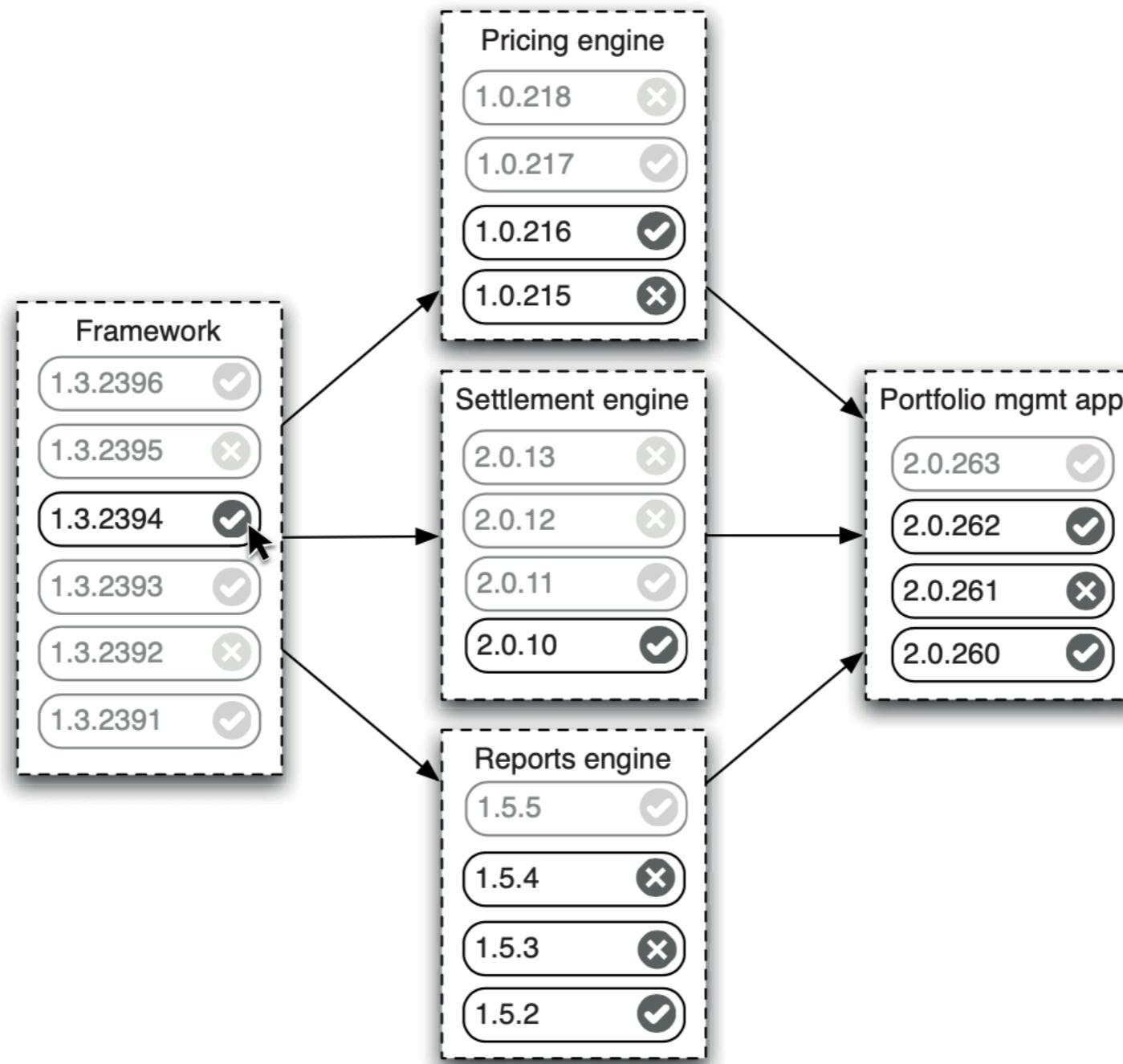


Figure 13.5 Visualizing downstream dependencies



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Pipelining Dependency Graphs

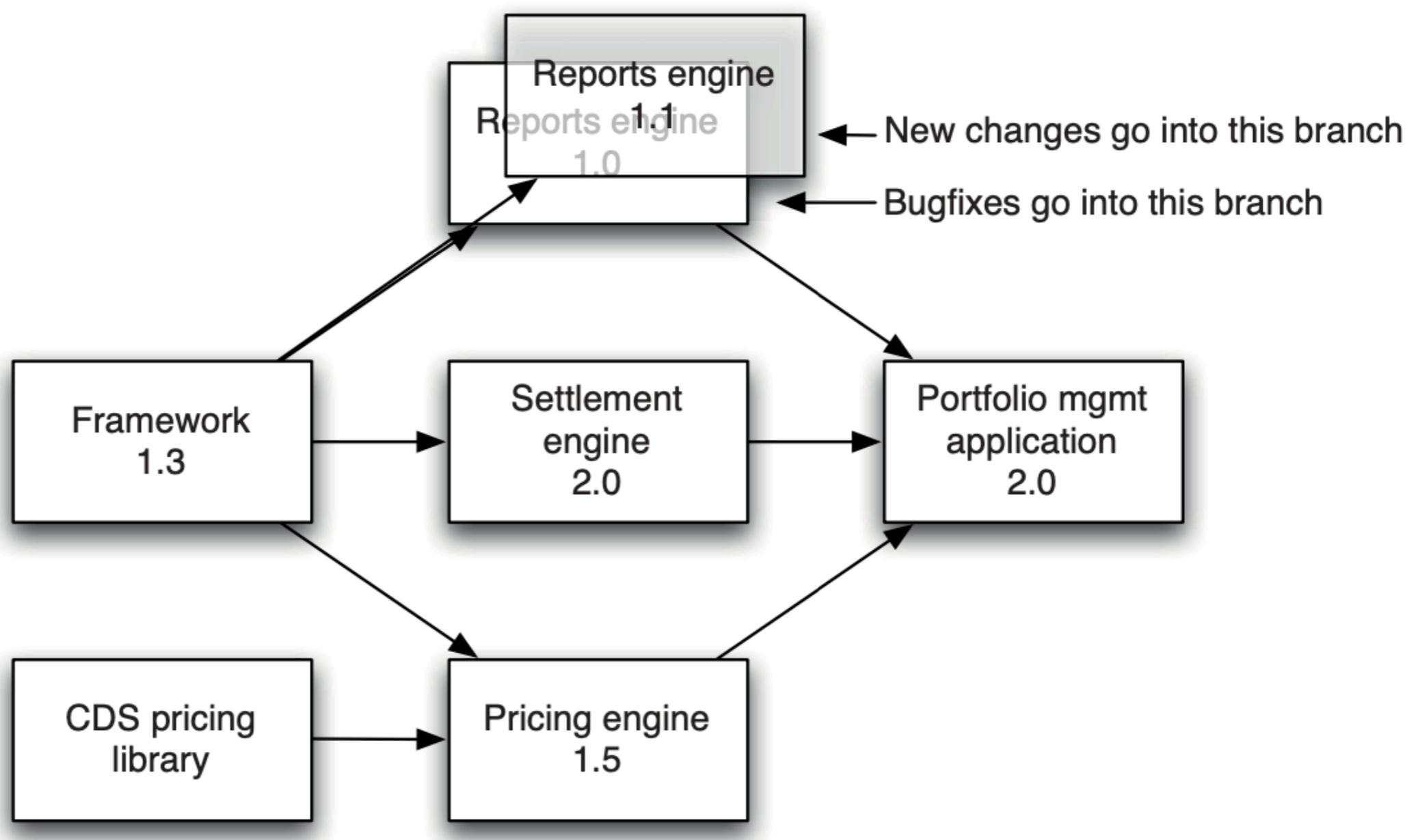


Figure 13.6 *Branching components*



# When Should We Trigger Builds? : Cautious Optimism

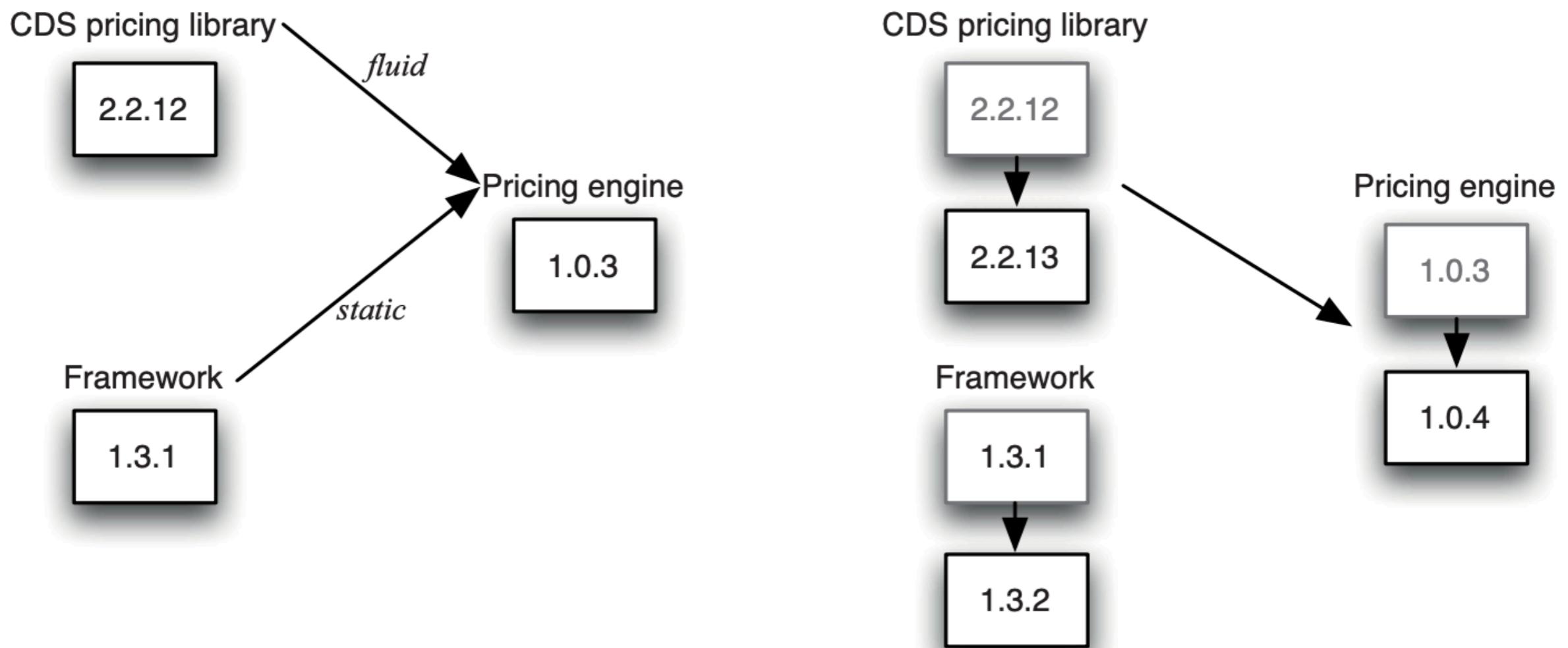


Figure 13.7 *Cautious optimism triggering*



# When Should We Trigger Builds? : Circular Dependencies

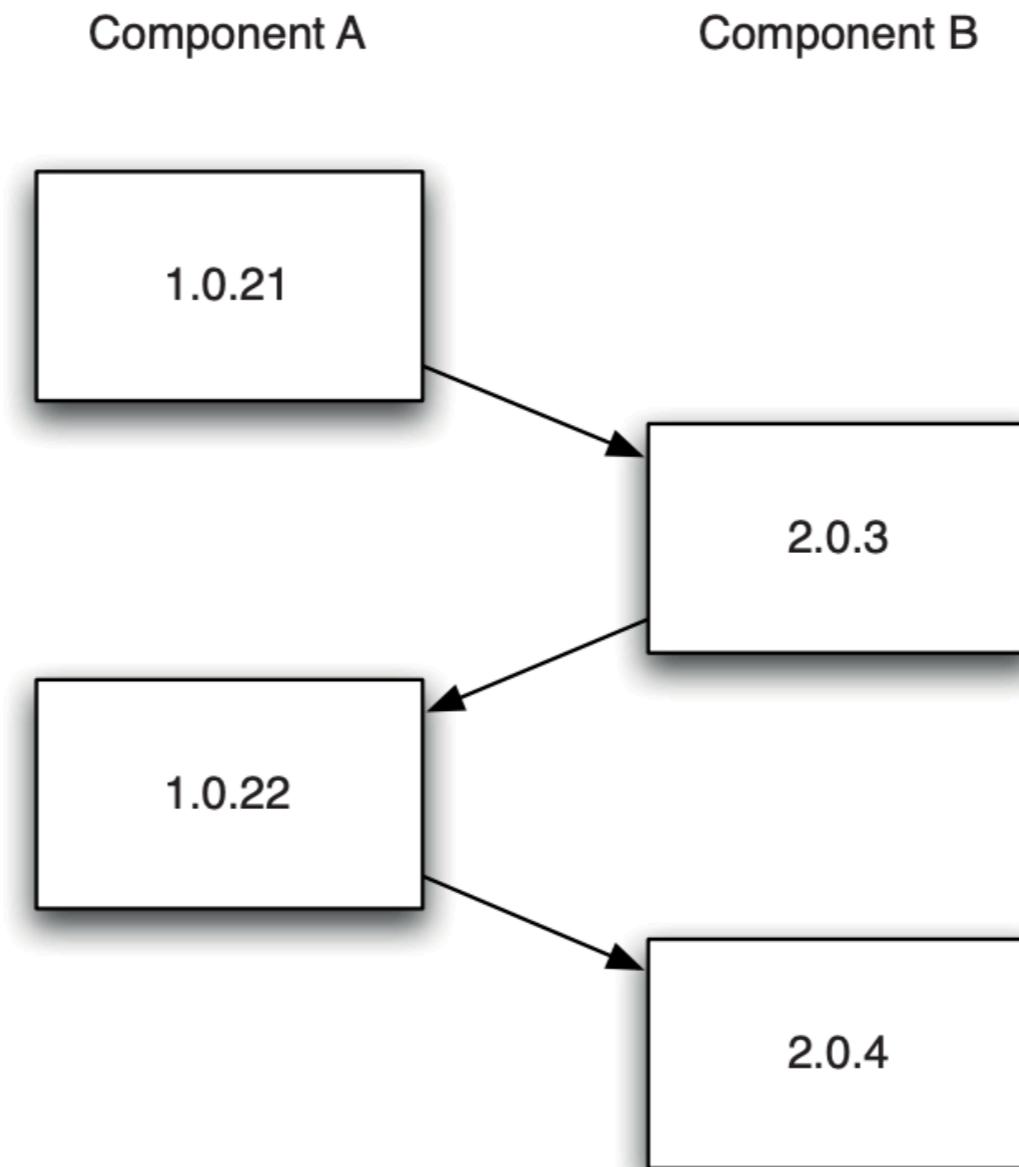


Figure 13.8 *Circular dependency build ladder*



Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Advanced Version Control



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

# Branching and Merging

It's acceptable when we branching when . . .

- Releasing
- Refactoring
- Creating the large change



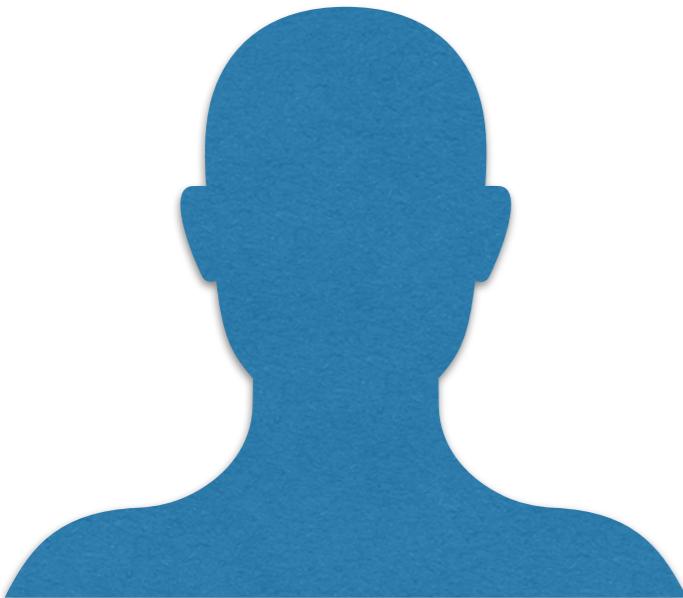
Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

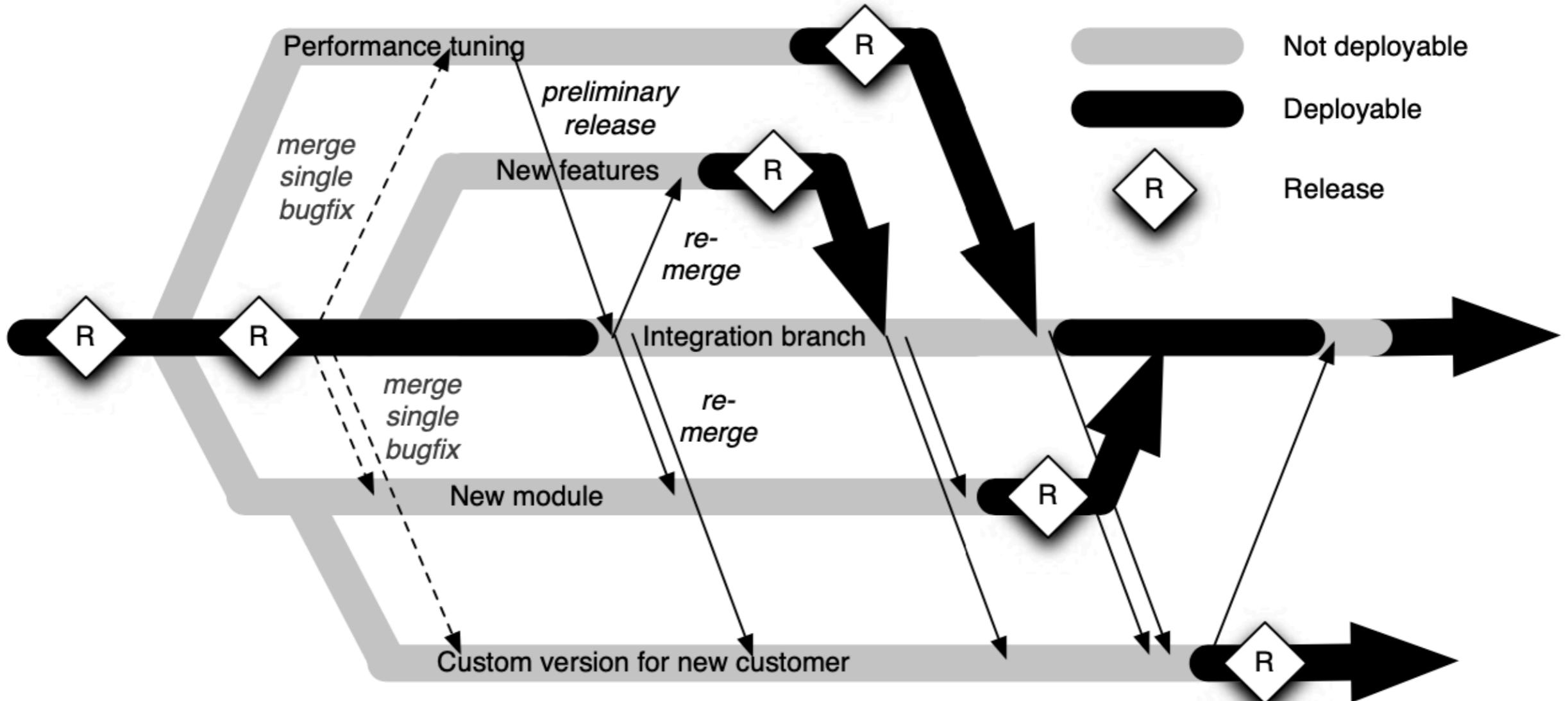
This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Branching and Merging

“We are branching for delay the pain of merging and conflict and got extremely pain in the end”



# Branches , Steams and Continuous Integration



**Figure 14.1** *A typical example of poorly controlled branching*



# Branch for Release

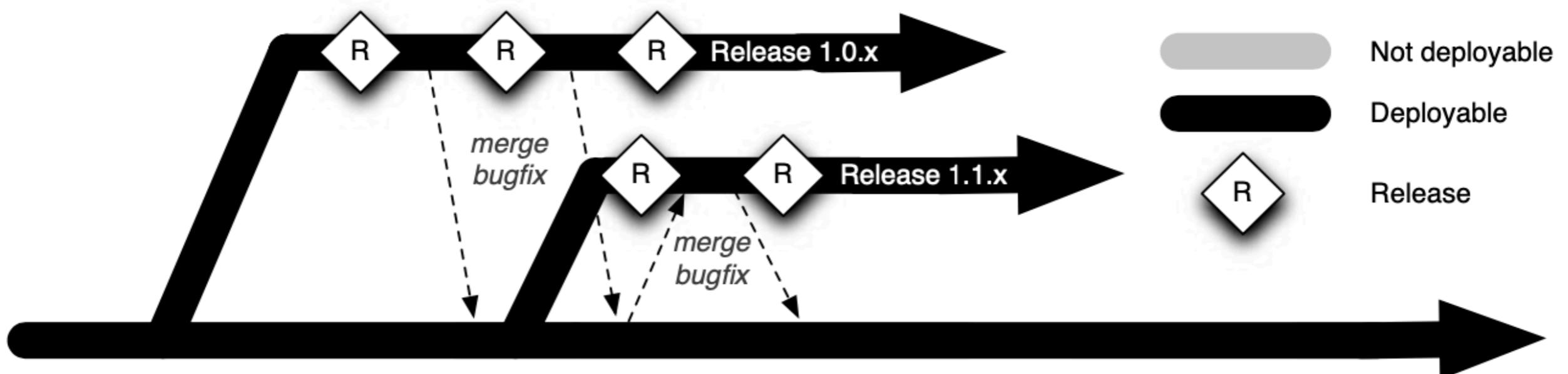
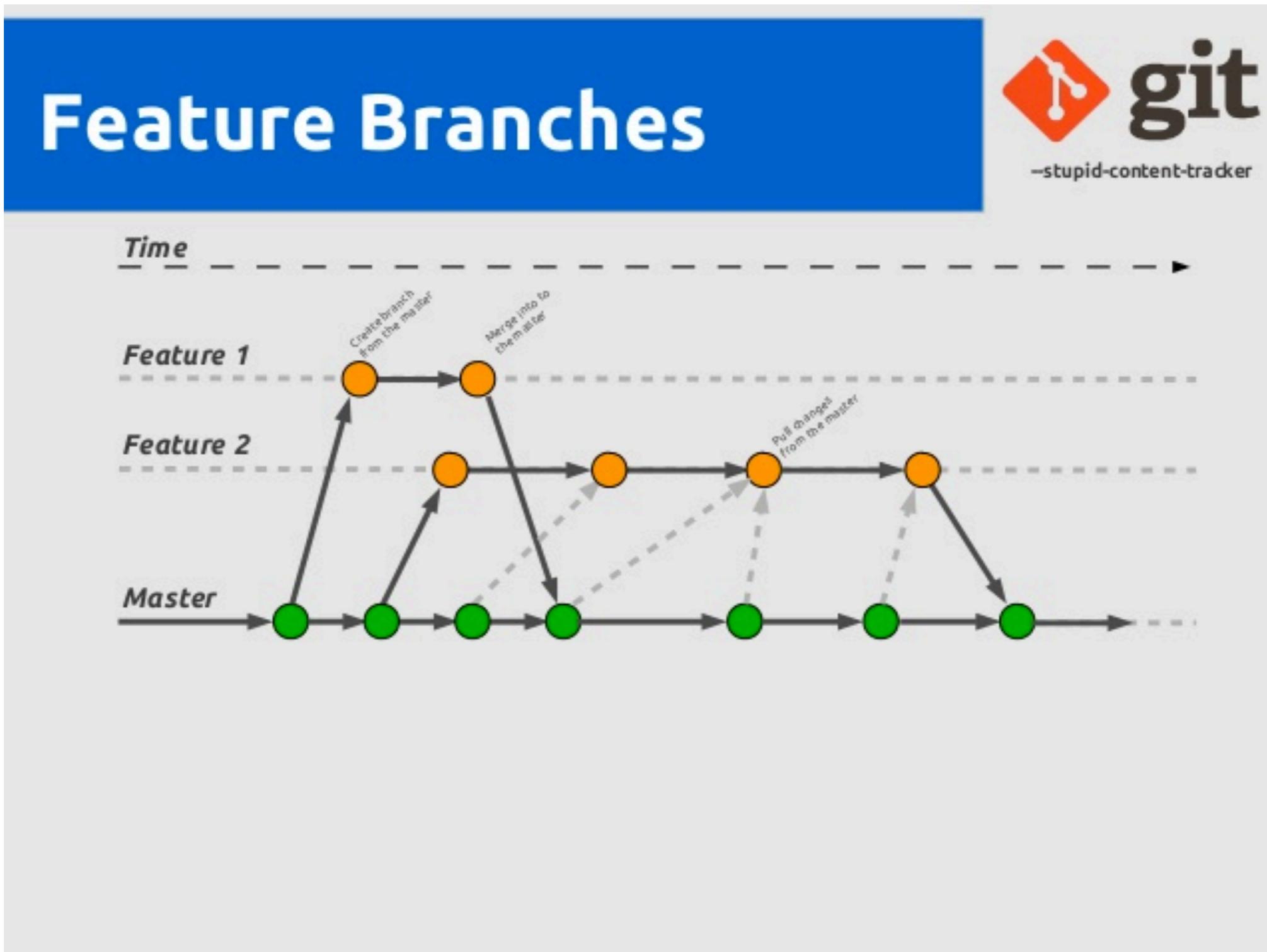


Figure 14.2 *Release branching strategy*



# Branch by Feature



Source: Git Branching Model : Harun Yardımcı, Software & Product Development Manager @ eBay Turkey



Share — copy and redistribute the material in any medium or format.

Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Branch by Team

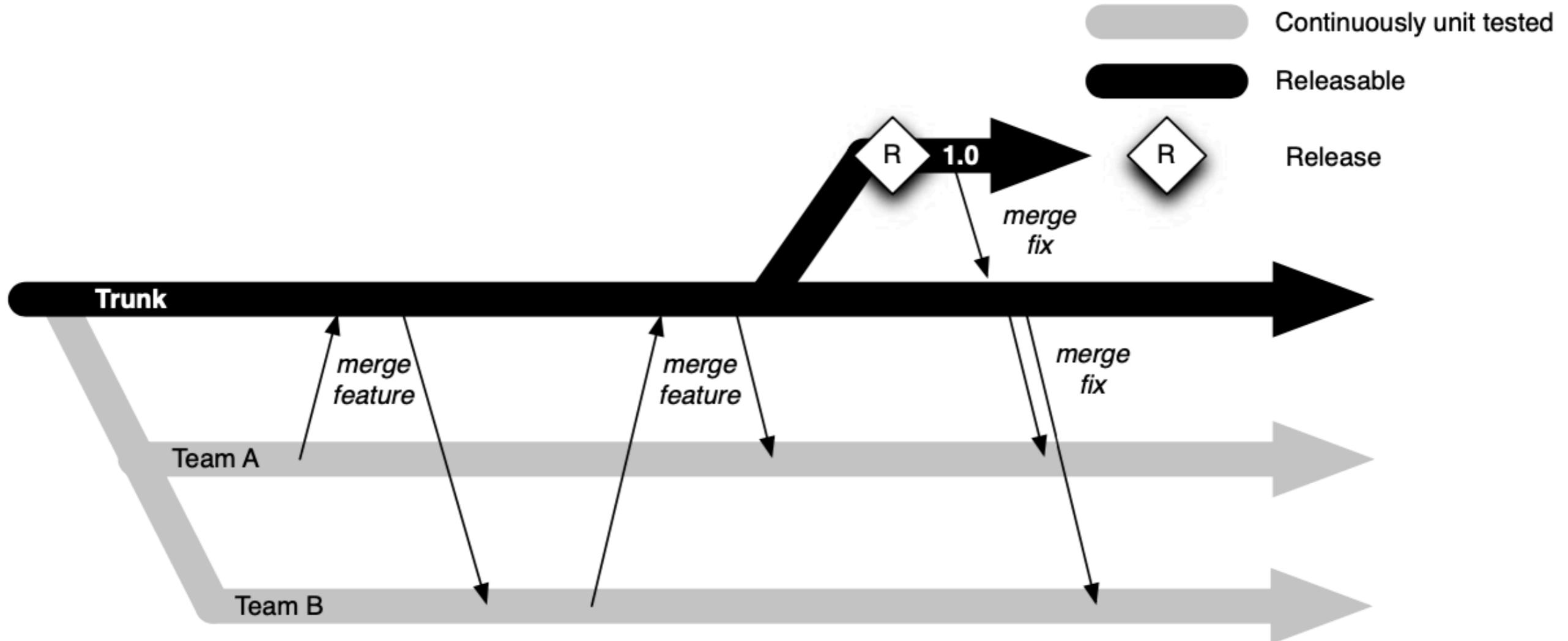


Figure 14.7 *Branch by team*



# Managing Continuous Delivery



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.  
NonCommercial — You may not use the material for commercial purposes.  
This work is licensed under a [Creative Commons Attribution-NonCommercial 4.0 International License](#).

# A Maturity Model for Configuration and Release Management

Practice	Build management and continuous integration	Environments and deployment	Release management and compliance	Testing	Data management	Configuration management
<b>Level 3 - Optimizing:</b> Focus on process improvement	Teams regularly meet to discuss integration problems and resolve them with automation, faster feedback, and better visibility.	All environments managed effectively. Provisioning fully automated. Virtualization used if applicable.	Operations and delivery teams regularly collaborate to manage risks and reduce cycle time.	Production rollbacks rare. Defects found and fixed immediately.	Release to release feedback loop of database performance and deployment process.	Regular validation that CM policy supports effective collaboration, rapid development, and auditable change management processes.
<b>Level 2 - Quantitatively managed:</b> Process measured and controlled	Build metrics gathered, made visible, and acted on. Builds are not left broken.	Orchestrated deployments managed. Release and rollback processes tested.	Environment and application health monitored and proactively managed. Cycle time monitored.	Quality metrics and trends tracked. Non functional requirements defined and measured.	Database upgrades and rollbacks tested with every deployment. Database performance monitored and optimized.	Developers check in to mainline at least once a day. Branching only used for releases.
<b>Level 1 - Consistent:</b> Automated processes applied across whole application lifecycle	Automated build and test cycle every time a change is committed. Dependencies managed. Re-use of scripts and tools.	Fully automated, self-service push-button process for deploying software. Same process to deploy to every environment.	Change management and approvals processes defined and enforced. Regulatory and compliance conditions met.	Automated unit and acceptance tests, the latter written with testers. Testing part of development process.	Database changes performed automatically as part of deployment process.	Libraries and dependencies managed. Version control usage policies determined by change management process.
<b>Level 0 – Repeatable:</b> Process documented and partly automated	Regular automated build and testing. Any build can be re-created from source control using automated process.	Automated deployment to some environments. Creation of new environments is cheap. All configuration externalized / versioned	Painful and infrequent, but reliable, releases. Limited traceability from requirements to release.	Automated tests written as part of story development.	Changes to databases done with automated scripts versioned with application.	Version control in use for everything required to recreate software: source code, configuration, build and deploy scripts, data migrations.
<b>Level -1 – Regressive:</b> processes unrepeatable, poorly controlled, and reactive	Manual processes for building software. No management of artifacts and reports.	Manual process for deploying software. Environment-specific binaries. Environments provisioned manually.	Infrequent and unreliable releases.	Manual testing after development.	Data migrations unversioned and performed manually.	Version control either not used, or check-ins happen infrequently.

Figure 15.1 *Maturity model*

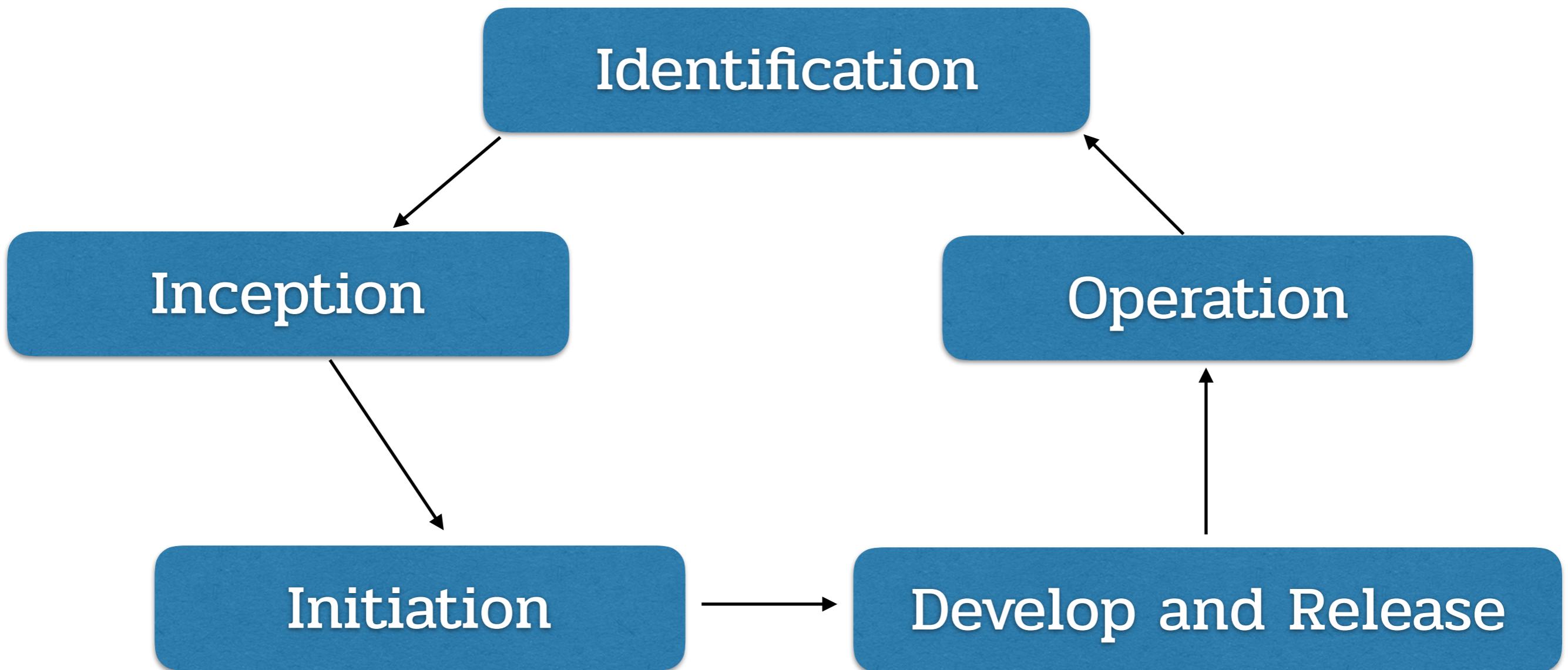


Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Project Lifecycle



# Project Lifecycle

## Identification

- Business Objective
- Requirement Gathering
- Business Cases



# Project Lifecycle

Inception

- Release Strategy
- Testing Strategy
- Description Development

Lifecycle

- Deliverables living Documents



# Project Lifecycle

## Initiation

- Project's Infrastructure
- Setting Up Project
- Setup Working Agreement
- Try to Working with Tools

and Architecture



# Project Lifecycle

Develop and Release

- Make Your Software Always Working
- Showcase Testing
- Deploy Every Iterations



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Project Lifecycle

Operation

- First Release is not Last
- Series of Maintenance
- Discover user needs



# Common Delivery Problems

- Infrequent or Buggy Deployments
- Poor Application Quality
- Poorly Managed Continuous Integration
- Poor Configuration Management



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

# Compliance and Auditing

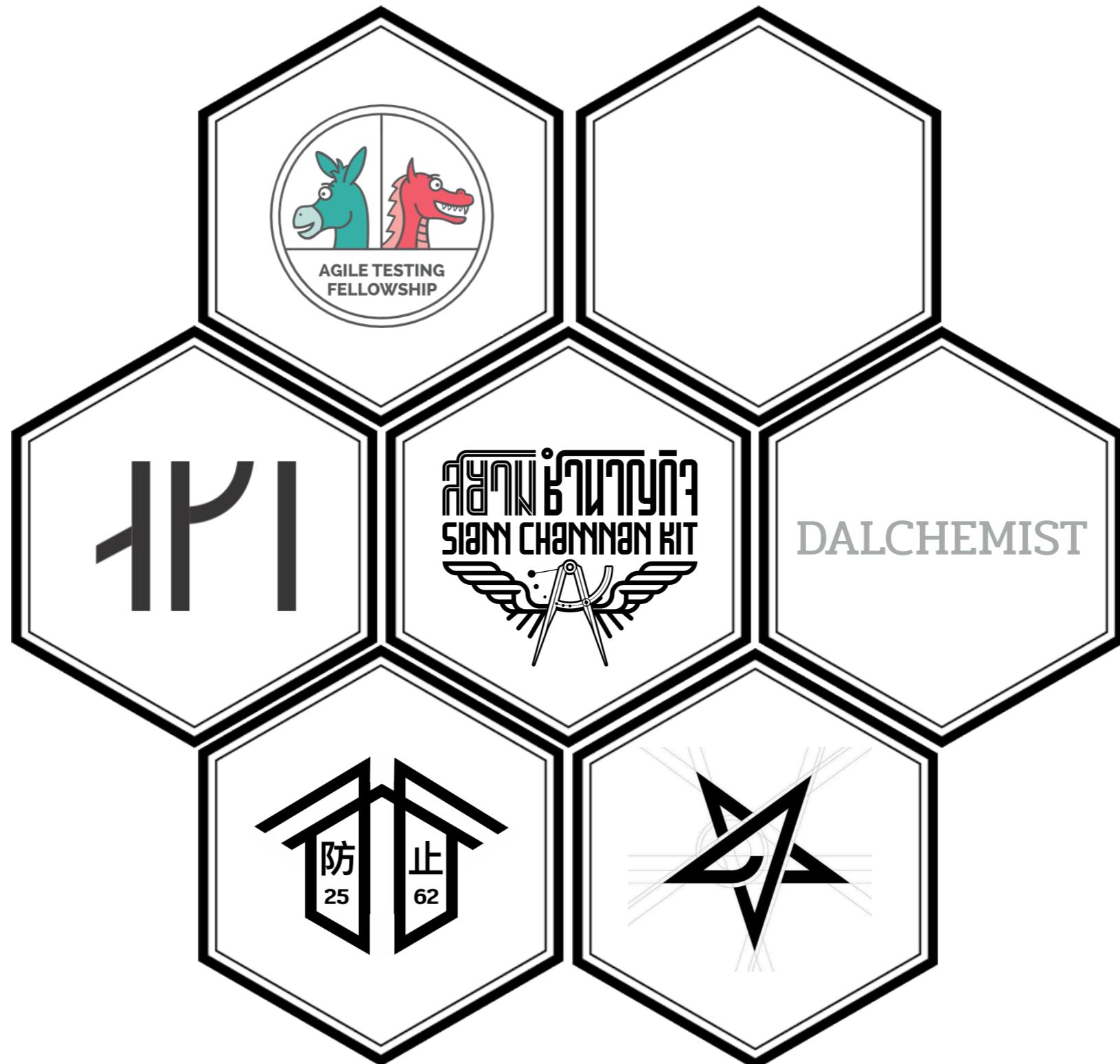
- Automation over Documentation
- Enforcing Traceability
- Working in Silos
- Change Management



Share — copy and redistribute the material in any medium or format.  
Adapt — remix, transform, and build upon the material.

NonCommercial — You may not use the material for commercial purposes.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.



Share — copy and redistribute the material in any medium or format.  
 Adapt — remix, transform, and build upon the material.  
 NonCommercial — You may not use the material for commercial purposes.  
 This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.