Continuous Delivery Patterns and Anti-Patterns

ORIGINAL BY PAUL DUVALL, CTO AND CO-FOUNDER, STELLIGENT
UPDATED BY MICHAEL OLSON, PRINCIPAL PRODUCT MARKETING MANAGER, PUPPET

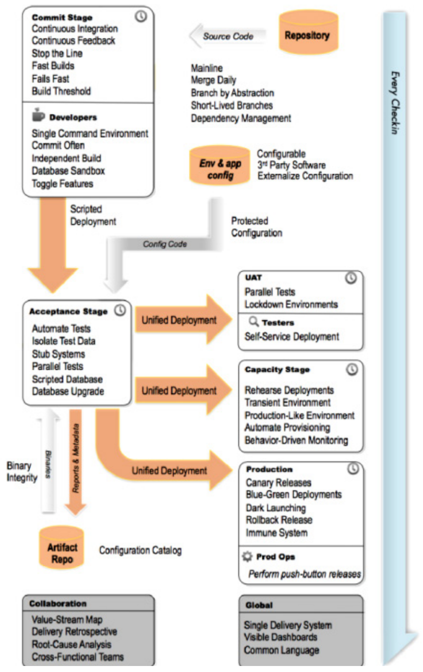
CONTENTS

ABOUT CONTINUOUS DELIVERY

With Continuous Delivery (CD), teams continuously deliver new versions of software to production by decreasing the cycle time between an idea and usable software through the automation of the entire software delivery process: code commit, build, test, deployment, and release. CD is enabled through the Deployment Pipeline, which encompasses a collection of patterns described in this Refcard.

CD is concerned with "…how all the moving parts fit together: configuration management, automated testing, continuous integration and deployment, data management, environment management, and release management."

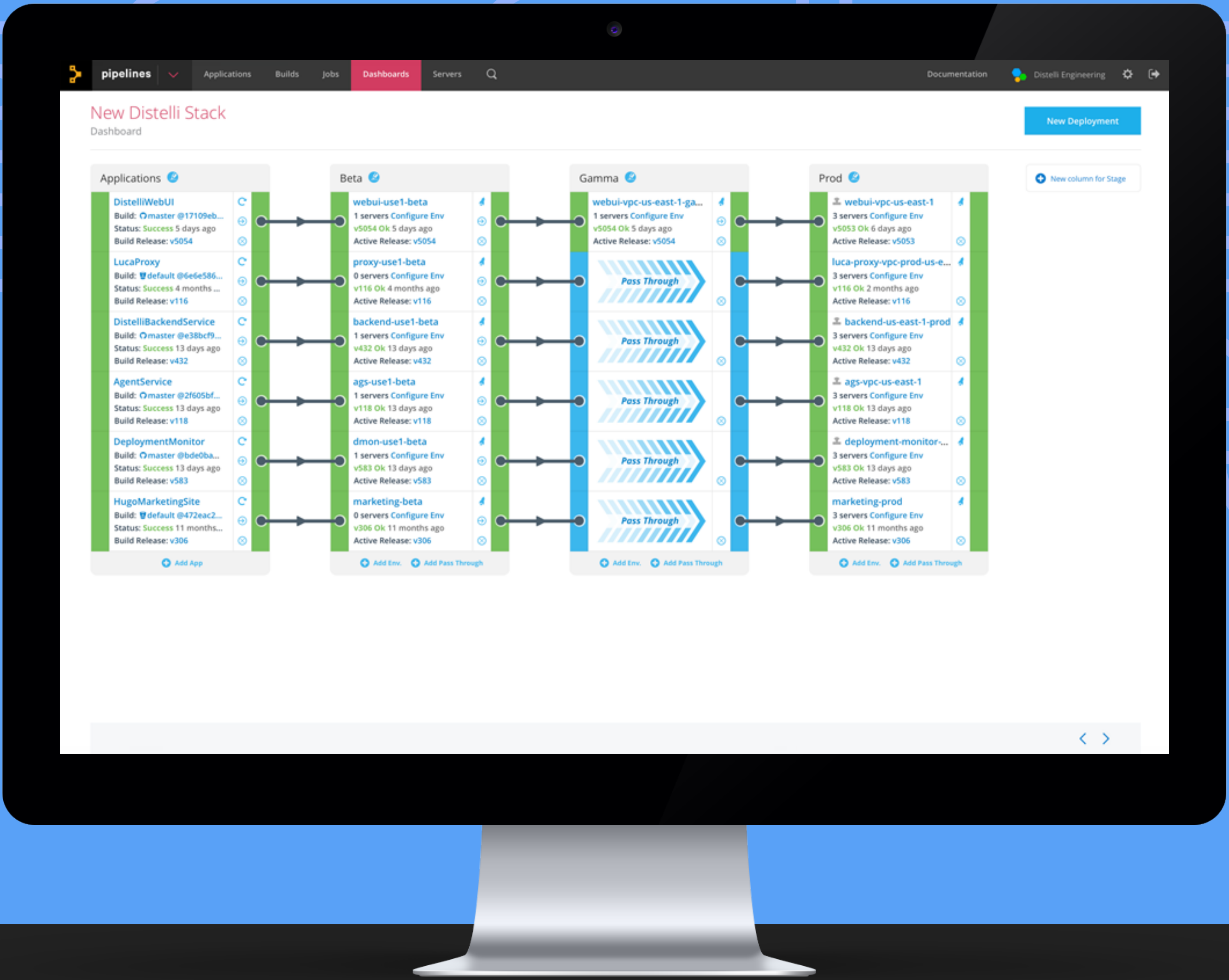
THE DEPLOYMENT PIPELINE



The purpose of the deployment pipeline is threefold:

- **Visibility:** All aspects of the delivery process – building, testing, deploying, and releasing – are visible to all team members promoting collaboration.
- **Feedback:** Team members learn of problems as soon as they occur so that issues are fixed as soon as possible.
- **Continually Deploy:** Through a fully automated process, you can deploy and release any version of the software to any environment faster and more frequently.

In the Deployment Pipeline diagram above, all of the patterns are shown in context. There are some patterns that span multiple stages of the pipeline, so I chose the stage where it's most predominantly used.



New Distelli Stack
Dashboard

Applications

DistelliWebUI
Build: master @17109eb...
Status: Success 5 days ago
Build Release: v5054

LucaProxy
Build: default @6e6e586...
Status: Success 4 months ...
Build Release: v116

DistelliBackendService
Build: master @e38bcf9...
Status: Success 13 days ago
Build Release: v432

AgentService
Build: master @2f605bf...
Status: Success 13 days ago
Build Release: v118

DeploymentMonitor
Build: master @bde0ba...
Status: Success 13 days ago
Build Release: v583

HugoMarketingSite
Build: default @472eac2...
Status: Success 11 months...
Build Release: v306

Beta

webui-use1-beta
1 servers Configure Env
v5054 Ok 5 days ago
Active Release: v5054

proxy-use1-beta
0 servers Configure Env
v116 Ok 4 months ago
Active Release: v116

backend-use1-beta
1 servers Configure Env
v432 Ok 13 days ago
Active Release: v432

ags-use1-beta
1 servers Configure Env
v118 Ok 13 days ago
Active Release: v118

dmon-use1-beta
1 servers Configure Env
v583 Ok 13 days ago
Active Release: v583

marketing-beta
0 servers Configure Env
v306 Ok 11 months ago
Active Release: v306

Gamma

webui-vpc-us-east-1-ga...
1 servers Configure Env
v5054 Ok 5 days ago
Active Release: v5054

Prod

webui-vpc-us-east-1
3 servers Configure Env
v5053 Ok 6 days ago
Active Release: v5053

luca-proxy-vpc-prod-us-e...
3 servers Configure Env
v116 Ok 2 months ago
Active Release: v116

backend-us-east-1-prod
3 servers Configure Env
v432 Ok 13 days ago
Active Release: v432

ags-vpc-us-east-1
3 servers Configure Env
v118 Ok 13 days ago
Active Release: v118

deployment-monitor-...
3 servers Configure Env
v583 Ok 13 days ago
Active Release: v583

marketing-prod
3 servers Configure Env
v306 Ok 11 months ago
Active Release: v306

Get to continuous delivery faster with Puppet Pipelines.

- Enable self-service builds and deployments for your developers
- Get deep visibility into your delivery pipeline & the status of every deployment
- Define continuous delivery pipelines for your cloud-native and traditional apps in under 15 minutes

Meet Puppet Pipelines > **Request a demo >**

Learn more at **puppet.com**

puppet Pipelines™

BENEFITS

- **Empowering Teams:** Because the deployment pipeline is a pull system, testers, developers, operations, and others can self-service the application version into an environment of their choice.
- **Reducing Errors:** Ensuring the correct version, configuration, database schema, etc. are applied the same way every time through automation.
- **Lowering Stress:** Through push-button releases to production and rehearsing deployments, a release becomes commonplace without the typical stress.
- **Deployment Flexibility:** Instantiate a new environment or configuration by making a few changes to the automated delivery system.
- **Practice Makes Perfect:** Through the deployment pipeline, the final deployment into production is being rehearsed every single time the software is deployed to any target environments.

CONFIGURATION MANAGEMENT

Configuration Management is "the process by which all artifacts relevant to your project, and the relationships between them, are stored, retrieved, uniquely identified, and modified." (1)

Note: Each pattern is cited with a number in parentheses that corresponds to the source in the References section.

CONFIGURABLE THIRD-PARTY SOFTWARE (1)

Pattern	Evaluate and use third-party software that can be easily configured, deployed, and automated.
Anti-patterns	Procuring software that cannot be externally configured. Software without an API or command line interface that forces teams to use the GUI only.

CONFIGURATION CATALOG (1)

Pattern	Maintain a catalog of all options for each application, how to change these options and storage locations for each application. Automatically create this catalog as part of the build process.
Anti-patterns	Configuration options are not documented. The catalog of applications and other assets is "tribal knowledge".

MAINLINE (3)

Pattern	Minimize merging and keep the number of active code lines manageable by developing on a mainline.
Anti-patterns	Multiple branches per project.

MERGE DAILY (1)

Pattern	Changes committed to the mainline are applied to each branch on at least a daily basis.
Anti-patterns	Merging every iteration once a week or less often than once a day.

PROTECTED CONFIGURATION (5), (1)

Pattern	Store configuration information in secure remotely accessible locations such as a database, directory, or registry.
Anti-patterns	Open text passwords and/or single machine or share.

REPOSITORY (3), (1)

Pattern	All source files — executable code, configuration, host environment, and data — are committed to a version control repository.
Anti-patterns	Some files are checked in, others, such as environment configuration or data changes, are not. Binaries – which can be recreated through the build and deployment process – are checked in.

SHORT-LIVED BRANCHES (1)

Pattern	Branches must be short lived – ideally less than a few days and never more than an iteration.
Anti-patterns	Branches that last more than an iteration. Branches by product feature that live past a release.

SINGLE COMMAND ENVIRONMENT (1)

Pattern	Check out the project's version-control repository and run a single command to build and deploy the application to any accessible environment, including the local development.

Anti-patterns	Forcing the developer to define and configure environment variables. Making the developer install numerous tools in order for the build/deployment to work.

SINGLE PATH TO PRODUCTION (1)

Pattern	Configuration management of the entire system - source, configuration, environment, and data. Any change can be tied back to a single revision in the version-control system.
Anti-patterns	Parts of the system are not versioned. Inability to get back to a previously configured software system.

CONTINUOUS INTEGRATION (CI)

BUILD THRESHOLD (5)

Pattern	Fail a build when a project rule is violated – such as architectural breaches, slow tests, and coding standard violations.
Anti-patterns	Manual code reviews. Learning of code quality issues later in the development cycle.

COMMIT OFTEN (6)

Pattern	Each team member checks in regularly to trunk — at least once a day but preferably after each task to trigger the CI system.
Anti-patterns	Source files are committed less frequently than daily due to the number of changes from the developer.

CONTINUOUS FEEDBACK (6)

Pattern	Send automated feedback from CI system to all cross-functional team members.
Anti-patterns	Notifications are not sent; notifications are ignored; CI system spams everyone with information they cannot use.

CONTINUOUS INTEGRATION (6)

Pattern	Building and testing software with every change committed to a project's version control repository.
Anti-patterns	Scheduled builds, nightly builds, building periodically, building exclusively on developer's machines, or not building at all.

STOP THE LINE (5), (1), (4), (12)

Pattern	Fix software delivery errors as soon as they occur; stop the line. No one checks in on a broken build, as the fix becomes the highest priority.
Anti-patterns	Builds stay broken for long periods of time, thus preventing developers from checking out functioning code.

INDEPENDENT BUILD (6)

Pattern	Write build scripts that are decoupled from IDEs. These build scripts are executed by a CI system so that software is built at every change.
Anti-patterns	Automated build relies on IDE settings. Builds are unable to be run from the command line.

VISIBLE DASHBOARDS

Pattern	Provide large visible displays that aggregate information from your delivery system to provide high quality feedback to the Cross-Functional Team in real time.
Anti-patterns	Email-only alerts or not publicizing the feedback to the entire team.

TESTING

AUTOMATE TESTS

Pattern	Automate the verification and validation of software to include unit, component, capacity, functional, and deployment tests
Anti-patterns	Manual testing of units, components, deployment, and other types of tests.

Unit: Automating tests without any dependencies.

Component: Automating tests with dependencies to other components and heavyweight dependencies such as the database or file system.

Deployment: Automating tests to verify the deployment and configuration were successful. Sometimes referred to as a "smoke tests."

Functional: Automating tests to verify the behavior of the software from a user's perspective.

Capacity: Automating load and performance testing linear production conditions.

ISOLATE TEST DATA (1)

Pattern	Use transactions for database-dependent tests (e.g. component tests) and roll back the transaction when done. Use a small subset of data to effectively test behavior.
Anti-patterns	Using a copy of production data for Commit Stage tests. Running tests against a shared database.

PARALLEL TESTS (1)

Pattern	Run multiple tests in parallel across hardware instances to decrease the time in running tests.
Anti-patterns	Anti-patterns Running tests on one machine or instance. Running dependent tests that cannot be run in parallel.

STUB SYSTEMS (1)

Pattern	Use stubs to simulate external systems to reduce deployment complexity.
Anti-patterns	Manually installing and configuring interdependent systems for Commit Stage build and deployment.

DEPLOYMENT PIPELINE

Pattern	A deployment pipeline is an automated implementation of your application's build, test, deploy, and release process.
Anti-patterns	Deployments require human intervention (other than approval or clicking a button). Deployments are not production ready.

VALUE-STREAM MAP (4)

Pattern	Create a map illustrating the process from check in to the version control system to the software release to identify process bottlenecks.
Anti-patterns	Separately defined processes and views of the check-in to release process.

BUILD AND DEPLOYMENT SCRIPTING

DEPENDENCY MANAGEMENT (5)

Pattern	Centralize all dependent libraries to reduce bloat, class path problems, and repetition of the same dependent libraries and transitive dependencies from project to project.
Anti-patterns	Multiple copies of the same binary dependencies in each and every project. Redefining the same information for each project. This is classpath hell!

COMMON LANGUAGE (1)

Pattern	As a team, agree upon a common scripting language — such as Perl, Ruby, or Python — so that any team member can apply changes to the Single Delivery System.
Anti-patterns	Each team uses a different language making it difficult for anyone to modify the delivery system reducing cross-functional team effectiveness.

EXTERNALIZE CONFIGURATION (5)

Pattern	Changes between environments are captured as configuration information. All variable values are externalized from the application configuration into build/deployment-time properties.
Anti-patterns	Hardcoding values inside the source code or per target environment.

FAIL FAST (6)

Pattern	Fail the build as soon as possible. Design scripts so that processes that usually fail run first. These processes should be run as part of the commit stage.
Anti-patterns	Common build mistakes are not uncovered until late in the deployment process.

FAST BUILDS (6)

Pattern	The commit build provides feedback on common build problems as quickly as possible — usually in under 10 minutes.

Anti-patterns	Throwing everything into the commit stage process, such as running every type of automated static analysis tool or running load tests such that feedback is delayed.

SCRIPTED DEPLOYMENT (5)

Pattern	All deployment processes can be written in a script, checked in to the version-control system, and run as part of the single delivery system.
Anti-patterns	Deployment documentation is used instead of automation. Manual deployments or partially manual deployments.

UNIFIED DEPLOYMENT (5)

Pattern	The same deployment script is used for each deployment. The protected configuration – per environment — is variable but managed.
Anti-patterns	Different deployment script for each target environment or even for a specific machine. Manual configuration after deployment for each target environment.

DEPLOYING AND RELEASING APPLICATIONS

BINARY INTEGRITY (5)

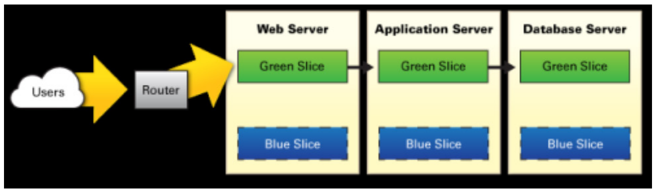
Pattern	Build your binaries once, while deploying the binaries to multiple target environments, as necessary.
Anti-patterns	Software is built in every stage of the deployment pipeline.

CANARY RELEASE

Pattern	Release software to production for a small subset of users (e.g. 10%) to get feedback prior to a complete rollout.
Anti-patterns	Software is released to all users at once.

BLUE-GREEN DEPLOYMENTS (1)

Pattern	Deploy software to a non-production environment (call it blue) while production continues to run. Once it's deployed and "warmed up," switch production (green) to non-production and blue to green simultaneously.
Anti-patterns	Production is taken down while the new release is applied to production instance(s).



DARK LAUNCHING (11)

Pattern	Launch a new application or features when it affects the least number of users.
Anti-patterns	Software is deployed regardless of number of active users.

ROLLBACK RELEASE (5)

Pattern	Provide an automated single command rollback of changes after an unsuccessful deployment.
Anti-patterns	Manually undoing changes applied in a recent deployment. Shutting down production instances while changes are undone.

SELF-SERVICE DEPLOYMENT (1)

Pattern	Any Cross-Functional Team member selects the version and environment to deploy the latest working software.
Anti-patterns	Deployments released to team are at specified intervals by the "build team." Testing can only be performed in a shared state without isolation from others.

INFRASTRUCTURE AND ENVIRONMENTS

AUTOMATE PROVISIONING (1)

Pattern	Automate the process of configuring your environment to include networks, external services, and infrastructure.
Anti-patterns	Configured instances are "works of art" requiring team members to perform partially or fully manual steps to provision them.

BEHAVIOR-DRIVEN MONITORING (1)

Pattern	Automate tests to verify the behavior of the infrastructure. Continually run these tests to provide near real-time alerting.
Anti-patterns	No real-time alerting or monitoring. System configuration is written without tests.

IMMUNE SYSTEM (9)

Pattern	Deploy software one instance at a time while conducting behavior-driven monitoring. If an error is detected during the incremental deployment, a rollback release is initiated to revert changes.
Anti-patterns	Non-incremental deployments without monitoring.

LOCKDOWN ENVIRONMENTS (1)

Pattern	Lock down shared environments from unauthorized external and internal usage, including operations staff. All changes are versioned and applied through automation.
Anti-patterns	The "Wild West:" any authorized user can access shared environments and apply manual configuration changes, putting the environment in an unknown state and leading to deployment errors.

PRODUCTION-LIKE ENVIRONMENTS (1)

Pattern	Target environments are as similar to production as possible.
Anti-patterns	Environments are "production-like" only weeks or days before a release. Environments are manually configured and controlled.

TRANSIENT ENVIRONMENTS

Pattern	Utilizing the Automate Provisioning, Scripted Deployment, and Scripted Database patterns. Any environment should be capable of terminating and launching at will.
Anti-patterns	Environments are fixed to "DEV," "QA," or other predetermined environments.

DATA

DATABASE SANDBOX (7)

Pattern	Create a lightweight version of your database – using the Isolate Test Data pattern. Each developer uses this lightweight DML to populate his local database sandboxes to expedite test execution.
Anti-patterns	Shared database. Developers and testers are unable to make data changes without it potentially adversely affecting other team members immediately.

DECOUPLE DATABASE (1)

Pattern	Ensure your application is backward and forward compatible with your database so you can deploy each independently.
Anti-patterns	Application code data are not capable of being deployed separately.

DATABASE UPGRADE (7)

Pattern	Use scripts to apply incremental changes in each target environment to a database schema and data.
Anti-patterns	Manually applying database and data changes in each target environment.

SCRIPTED DATABASE (7)

Pattern	Script all database actions as part of the build process.
Anti-patterns	Using data export/import to apply data changes. Manually applying schema and data changes to the database.

INCREMENTAL DEVELOPMENT

BRANCH BY ABSTRACTION (2)

Pattern	Instead of using version-control branches, create an abstraction layer that handles both an old and new implementation. Remove the old implementation.
Anti-patterns	Branching using the version-control system leading to branch proliferation and difficult merging. Feature branching.

TOGGLE FEATURES (10)

Pattern	Deploy new features or services to production but limit access dynamically for testing purposes.
Anti-patterns	Waiting until a feature is fully complete before committing the source code.

COLLABORATION

DELIVERY RETROSPECTIVE (1)

Pattern	For each iteration, hold a retrospective meeting where everybody on the Cross-Functional Team discusses how to improve the delivery process for the next iteration.
Anti-patterns	Waiting until an error occurs during a deployment for Dev and Ops to collaborate. Having Dev and Ops work separately.

CROSS-FUNCTIONAL TEAMS (1)

Pattern	Everybody is responsible for the delivery process. Any person on the Cross-Functional Team can modify any part of the delivery system.
Anti-patterns	Siloed teams: Development, Testing, and Operations have their own scripts and processes and are not part of the same team.

Amazon.com has an interesting take on this approach. They call it “You build it, you run it”. Developers take the software they’ve written all the way to production.

ROOT-CAUSE ANALYSIS (1)

Pattern	Learn the root cause of a delivery problem by asking “why” of each answer and symptom until discovering the root cause.
Anti-patterns	Accepting the symptom as the root cause of the problem.

CONTINUOUS DELIVERY TOOLS

This is meant to be an illustrative list, not an exhaustive list, to give you an idea of the types of tools and some of the vendors that help to enable effective Continuous Delivery.

CATEGORY	EXAMPLE TOOLS
Product Planning	Atlassian JIRA, Jama, CA Rally, Aha!, CollabNet VersionOne, Pivotal
Source Code Management	GitHub, GitLab, Atlassian Bitbucket, Microsoft Team Foundation Server, Perforce, Subversion
Continuous Integration	Jenkins, CircleCI, CloudBees, GitLab, Atlassian Bamboo, Travis CI, JetBrains TeamCity, Microsoft Azure Pipelines, Puppet Pipelines
Build	Ant, Gant, Gradle, make, Maven, Rake, Fabric, Func
Testing	Twist , AntUnit, Cucumber, DbUnit, webrat, easyb, Fitnesse, JMeter, JUnit, NBehave, SoapUI, Selenium, RSpec, SauceLabs, Perfecto
Artifact Repository	JFrog Artifactory, Ivy, Archiva, Sonatype Nexus, Bundler
Continuous Delivery & Release Automation	Puppet Pipelines, AWS CodePipeline, CA Automic, Electric Cloud, IBM UrbanCode, Octopus Deploy, Spinnaker, XebiaLabs
Infrastructure Automation	Puppet Enterprise, Chef, Ansible
Cloud Provisioning & Orchestration	HashiCorp Terraform, Puppet, Ansible
Container Management System & Application Platform-as-a-Service	Kubernetes, Mesos, HashiCorp Nomad, Docker Swarm, CloudFoundry
Container Registry	Puppet Container Registry, Docker Hub, AWS, Microsoft Azure, Google Cloud, JFrog, Red Hat Quay, Harbor
Application Performance Monitoring	New Relic, AppDynamics, Datadog, Splunk, Dynatrace

Software Delivery Performance Management	Puppet Insights, CloudBees DevOptics, XebiaLabs
Collaboration	Slack, JIRA, Trello

REFERENCES

1. Jez Humble and David Farley, "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation", Addison Wesley Professional, 2010
2. Paul Hammant and continuousdelivery.com
3. Stephen P. Berczuk and Brad Appleton, "Software Configuration Management Patterns.", Addison Wesley Professional, 2003
4. Mary and Tom Poppendieck, "Leading Lean Software Development", Addison Wesley, 2009
5. Paul M. Duvall, "Continuous integration. Patterns and Antipatterns", DZone refcard #84, 2010 bit.ly/l8rfVS
6. Paul M. Duvall, "Continuous integration. Improving Software Quality and Reducing Risk", Addison Wesley, 2007
7. Scott W. Ambler and Pramodkumar J. Saladage, "Refactoring Databases. Evolutionary Database Design", Addison Wesley, 2006.
8. Paul M. Duvall, IBM developerWorks series "Automation for the people" ibm.co/iwwvPX
9. IMVU: bit.ly/jhqP5f
10. Martin Fowler and Facebook: on.fb.me/miBrOM
11. Facebook Engineering: on.fb.me/miBrOM
12. Paul Julius, Enterprise Continuous Integration Maturity Model, bit.ly/m7h5vC

Written by **Michael Olson** , Principal Product Marketing Manager, Puppet

Michael Olson is a Principal Product Marketing Manager at Puppet, where he's responsible for product launches and go-to-market strategy for Puppet's products. When he's not working closely with the product teams at Puppet, you can find Michael traveling around the world to advise organizations about DevOps and Continuous Delivery practices and running around on the soccer field.