Speeding Up Software Delivery With Effective Change Management

Do you have something to say about this article?

Visit the *Journal* pages of the ISACA® website *(www.isaca.org/journal)*, find the article and click on the Comments link to share your thoughts.

https://bit.ly/2M9lFTa

The value in software is realized only when the end product is in the hands of its intended users. Traditional software development can be slow, requiring multiple manual reviews and approvals as well as teams of specialists to perform specific tasks as the software project progresses. To increase the speed with which software is created and delivered, many organizations are moving to continuous integration and continuous deployment (CICD), leveraging automated workflow and small teams responsible for managing a change from start to finish.

Traditionally, auditors have relied on reviews, approvals and segregation of duties (SoD) to prove that software releases meets management's expectation of the product. A CICD process is needed that meets both the need for speed in software delivery and auditors' requirements for efficient and effective change management practices.

Fundamentals of Change Management

The Information Technology Infrastructure Library (ITIL) defines change as "the addition, modification or removal of anything that could have an effect on IT services."[1] Changes may occur to applications, infrastructure components, documentation, configurations, policies, access rules and any other items that comprise a software product.

Change management underscores all software development, enabling the ability to trace management's intention for a product through the delivery of the deployed software and artifacts. Traditional change management is accomplished through reviews, testing, approvals and SoD. Each action often requires the process to stop and wait for manual action, such as a designation of approval, before moving to the next stage of the process. Of the 10 subprocesses defined by ITIL in its change management process,[2] eight incorporate an inherent pause in the process for assessments and authorizations.

Definition of CICD

ITIL's fundamentals of change management are timeless, but modern software development models change the way they are applied. For example, in CICD, software development and operations teams develop and deploy changes continuously to an application in small increments rather than in large, infrequent releases.

CICD combines two elements: continuous integration (CI) and continuous deployment (CD). In CI, developers merge changes into a central source code repository. With each change to the repository, the application is rebuilt and automated test suites are run against the rebuilt application ("the build"). If

Evan Bass, CISA
Is an accomplished audit executive with more than 15 years of combined Big 4 assurance and advisory consulting experience. He is currently an internal auditor senior manager at Intuit, Inc., where he leads various projects from risk advisory, US Sarbanes-Oxley Act (SOX) compliance to operational excellence reviews. Prior to joining Intuit, Bass was a manager of internal audit at James Hardie. Bass also provided worldwide IT security and process advisory services at PricewaterhouseCoopers (PwC).

W. Noel Haskins-Hafer, CISA, CRISC, CISM, CGEIT, CFE, CIA, CRMA, ISA
Is the senior manager of technical compliance for Intuit, Inc.'s Consumer Group. She advises senior leadership on strategies for developing cutting-edge financial management products that comply with US and international laws and regulations and industry standards. The first industry representative to San Diego State University's (California, USA) Center for the Teaching of Critical and Creative Thinking, Haskins-Hafer has published 12 articles on topics of interest to the audit, software engineering, project management and leadership communities.

a test case fails, the developer can investigate and resolve the failure quickly. With continuous integration, the software is in a working state at all times.[3] In theory, each integration improves the application incrementally. CD expands on CI by taking integration output that has met release criteria and updating the application in production with the integration output. Effective continuous deployment includes the ability to abort deployment and/or roll back to the prior version of the application if the integration output could degrade the live application in production.[4] CD depends on automation to detect success or failure of an integration as it moves through CICD stages. In a true CD environment, there is no interruption of the progression of a deliverable from integration to deployment.[5]

Goals and Benefits of CICD

The goal of CICD is to deliver high-quality, valuable software in an efficient, fast and reliable manner.[6] While commonly attributed to Agile programming and the first principle of the Agile Manifesto,[7] there is no requirement to practice Agile, or any other programming methodology, to reap the benefits of CICD.

An effective, high-performing CICD process brings many benefits, including lower release risk due to incremental deliveries, frequent feedback on software quality, increased reliability through automated deployment processes and reduced delays due to manual milestones.

Characteristics of a Compliant CICD Environment

A compliant change management process is one in which all changes conform to the documented process, and exceptions are rare and acceptable to management. Common characteristics include:

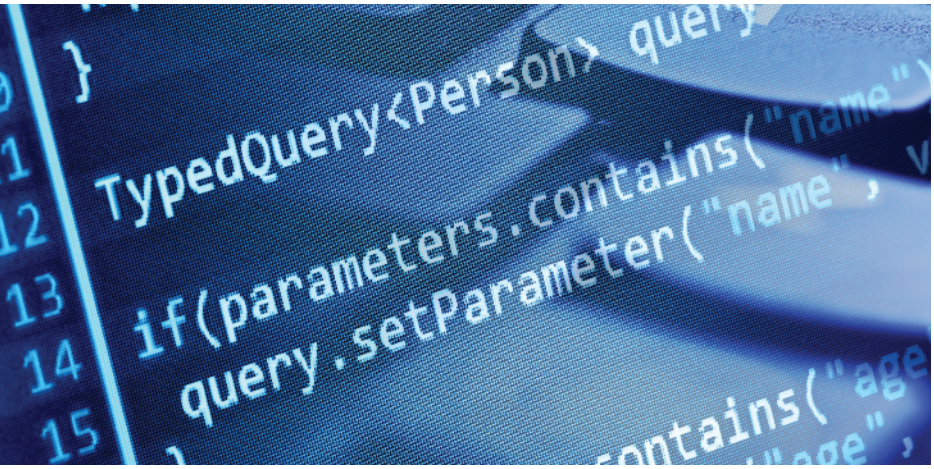
- **Quality**—Everyone involved throughout the build, test and deploy processes, and across all layers of the pipeline, has a stake in the quality of the deliverable and in ensuring errors, which create rework, are avoided across the enterprise.
- **Consistency**—An effective CICD environment includes guidelines for ensuring activities are carried out consistently and potential effects of changes to any and all relevant CICD components are considered during decision-making.

> "IN CICD, SOFTWARE DEVELOPMENT AND OPERATIONS TEAMS DEVELOP AND DEPLOY CHANGES CONTINUOUSLY TO AN APPLICATION IN SMALL INCREMENTS RATHER THAN IN LARGE, INFREQUENT RELEASES."

- **Pervasiveness**—Once established, the CICD environment should become the single standard for deploying all relevant artifacts, including deliverables and the CICD pipeline itself. Artifacts include, but are not limited to, software, documentation, processes, configurations, execution scripts, test scripts, test suites and credentials used to secure deliverables. Special-purpose deployment paths should be discouraged as they can lead to varying levels of quality and interoperability.
- **Scalability**—Effective CICD relies on efficiency throughout the pipeline and the technology stack. Stopping for one-off, tailored actions slows down delivery and defeats the purpose of CICD. The environment should be designed to meet anticipated use and future workload and include flexibility of automation to accommodate that workload.
- **Reliability and adequacy**—Automation is core to CICD, especially where testing is concerned. Since it is not practical to run the entire test suite against every change for every test step in the process, there should be a strategy for testing throughout the CICD process. This should include designating specific goals for each test stage and carefully selecting tests that together result in robust and representative testing through the delivery pipeline. The test data should be comprehensive enough to adequately validate deliverables without unnecessary repetition and delays.

Governance for CICD Change Management

Sustainable CICD is founded on good change management governance. Teams must know what they can expect from CICD as well as what they are expected to do to use the process. CICD change



management governance should hold to these principles:

- To what systems, processes and aspects of a business CICD applies (applicability)
- The processes, pipeline, protocols and other CICD components working as expected at all times (reliability)
- Consistency in the use of the CICD process, pipeline and protocols, regardless of which application or artifact is being processed. Deviations are documented and changes communicated to stakeholders with sufficient notice for the stakeholders to make any adjustments necessary to ensure consistency remains intact.

> "SUSTAINABLE CICD IS FOUNDED ON GOOD CHANGE MANAGEMENT GOVERNANCE."

Additionally, CICD should include these policies:

- **Documentation of each change**—Policies should outline acceptable use of change management ticketing, source control and executable build workflow tools. For example, a change ticket can be required to include specific information, such as a description of and reason for the change and any regulatory requirements for such change.[8] For audit purposes, information regarding build contents, creation, progression through the pipeline and timeline of that progression should be maintained, preferably through automation or configuration of the tools used to build the deliverable. For both audit and troubleshooting, documentation for released artifacts should provide a formal audit trail of evidence demonstrating major activities occurring throughout the change process. Each change should document the reason for the change, how it was evaluated for design and operational effectiveness, and when it was deployed to production.
- **Documentation of CICD processes and acceptance criteria**—The CICD process documentation should be clear and current. The criteria for allowing a change to progress automatically from one stage of the CICD pipeline to the next must be consistent and documented in writing. Exceptions to the criteria should be noted, along with instructions for documenting exception requirements (e.g., management approval, compensating tests or move-forward expectations). If the pipeline can be configured to suit different teams' specific needs, approved pipeline patterns should be used to ensure all required components of a compliance CICD pipeline are in place.Emergency change procedures should require the team to retroactively run the change through the standard change management process and review what led to the emergency to avoid recurrence.
- **Access control**—Because automation replaces many traditional "extra set of eyes" checkpoints, pipeline integrity becomes a proxy for SoD. There should be clear delineation between teams responsible for the pipeline infrastructure and those using it. The source control tool can be configured to effectively segregate duties through access (e.g., only engineers working on the CICD pipeline infrastructure can make changes to infrastructure components and/or configurations and patterns). Exceptions should be rare, approved, documented and monitored while active.
- **Audit trail of all approvals and test results**—Automated decisions made by CICD systems, along with the data used to make such decisions, should be recorded and, ideally, automatically documented into the change request ticket.
- **Logging**—All actions taken by developers and other CICD participants should be logged. There should be no ability to override controls to hide

events and actions taken. Access to change logs should be restricted. Logging activities should differentiate among pipeline activities (infrastructure), pipeline consumers (developers) and systematic pipeline components (build creation and test tools). Additionally, alerts should be issued for any that may have circumvented the established CICD process.

- **CICD component management**—For maximum benefit, the CICD environment and its components must work flawlessly together at all times. A complete inventory of all components (i.e., code, scripts, tests, development and testing criteria), along with each component's intended use, owner, dependencies and interfaces, should be available, current and maintained under source control. This information can reduce the amount of time the pipeline is stopped due to configuration errors.
- **Monitoring acceptance criteria in the CICD pipeline**—Management should define acceptance criteria, or thresholds, all changes must meet before moving to the next step in the automated process. Corresponding documentation should record how each threshold was determined, steps for exception processing if the thresholds are not met, periodic reviews of all thresholds, and actions and consequences for noncompliance. To ensure continued applicability, reliability and consistency of the pipeline, applications and CICD process as they evolve, the acceptance criteria should be monitored and updated as needed and the documentation updated to reflect those changes.
- **Infrastructure monitoring**—The infrastructure, including the CICD pipeline, should undergo periodic regression testing, including penetration or security vulnerability testing as deemed appropriate. Because environments from development through production are tightly integrated, each environment should be secured as if production relies on it—because, in fact, production does.

Controls for CICD Change Management

Change management is concerned with two primary areas of risk:

- An unauthorized or unapproved change is promoted to the production environment.
- Deployed changes do not function properly.

Figure 1 sets forth the minimal required controls for change management in CICD, and **figure 2** lists additional optional controls for consideration. Implementing these controls and retaining evidence that they have been implemented will both minimize the areas of risk and meet auditor expectations.

Figure 1— Controls for Change Management in CICD

Control ID	Minimal Required Controls
CM-01	Changes are documented and tested with evidence of acceptable results retained prior to deployment in the production environment.
CM-02	All changes are documented. Prior to deployment, change documentation must include, at a minimum: • Description of the change • Success criteria • Risk assessment/impact of the proposed change • Rollback procedures • Evidence of reviews and approvals by an independent party • Evidence of testing and results
CM-03	Developers cannot approve their own success criteria.
CM-04	All changes and test cases are reviewed by an individual other than the person who developed the change.
CM-05	Test libraries are periodically reviewed for completeness, accuracy and validity.
CM-06	All changes and test cases must be deployed through an approved release platform, using approved release platform templates.
CM-07	Pipeline templates are defined and periodically reviewed for integrity of milestone criteria.
CM-08	SoD is maintained between product and pipeline teams.
CM-09	The deployment pipeline is managed separately from product artifacts.

Figure 2—Potential Additional Change Management Controls in CICD	
Control ID	**Additional Controls for Consideration**
CM-10	Business approval of proposed changes occurs prior to development activities.
CM-11	Documentation of changes is stored in a central repository.
CM-12	All dependencies are defined and documented prior to deployment of the change.
CM-13	Supporting documentation, including procedures and user guidance, are updated prior to deployment.
CM-14	Deliverables and artifacts are maintained in a central source control repository.
CM-15	Change documentation and test results are stored in accordance with relevant data retention policies.
CM-16	Release content information is retained for a period required to meet relevant regulatory requirements.

Compliant CICD Is Achievable

On the surface, CICD appears to neglect many of the core elements auditors rely on to demonstrate effective change management: independent reviews and approvals, current and detailed documentation of the process, and segregation of duties.

CICD advocates and auditors alike can embrace the benefits of efficient and effective CICD, providing it demonstrates the core attributes of auditing:

- Accuracy and completeness of the product relative to its functional and nonfunctional requirements
- Integrity of processing and use, such that flaws cannot be exploited or abused
- Continuity of operations to support product availability
- Security of processing and data
- Traceability of development and transactional activities such that any activity performed within or on the system can be traced without a gap from its inception to its final disposition

Implementing the recommended governance policies and controls into a CICD methodology supports the attributes listed above, and thus enables an organization to remain compliant with audit and regulatory expectations while realizing the value of delivering software changes to intended users quickly and efficiently.

Change Request Contents

Change requests allow auditors to trace a change through the CICD process. Compliant change requests include:

- Description and priority of the change
- Components being changed
- Artifact category (e.g., code, test script, database table)
- Change category (e.g., enhancement, bug)
- Party/parties responsible for the change
- Tests and acceptance criteria for change progression through the CICD process
- Rollback procedures
- CICD configuration used

While this information should be documented for all changes, it does not need to be included in every change ticket. In some cases, the information can be documented as part of a common process; for example, if the same process is used to manage exceptions within a given CICD stage, the process can be documented once in a process document (wiki or otherwise) and maintained as appropriate. Where a given set of tests is used for regression or as success factors to move to the next stage-gate, it may be more efficient to refer to the test set than to list out all the specific tests, or to document that test set in the process document describing the CICD stage acceptance criteria.

Endnotes

1 AXELOS Limited, "ITIL Glossary of Terms," 2011, *https://www.axelos.com/Corporate/media/Files/Glossaries/ITIL_2011_Glossary_GB-v1-0.pdf*

2 The ITIL subprocesses are: Assessment of Change Proposals, Request for Change Logging and Review, Assessment and

Implementation of Emergency Changes, Change Assessment by the Change Manager, Change Assessment by the Change Approval Board, Change Scheduling and Build Authorization, Change Deployment Authorization, Minor Change Deployment, and Post Implementation Review and Change Closure. Design of Emergency and Minor Changes may not incorporate an in-stream stop for review and approval.

3 Humble, J.; D. Farley; *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation,* Addison-Wesley, USA, 2011

4 Caum, C.; "Continuous Deliver vs. Continuous Deployment: What's the Diff?" Puppet, 30 August 2013, *https://puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff*

5 Some organizations may instead adopt continuous delivery, where a build is promoted to a staging environment, tested extensively and paused to allow a manual inspection to ensure that deliverables are ready for production release. In continuous delivery, every change should be deployable at any time.

6 *Op cit* Humble and Farley

7 Agile, "Principles Behind the Agile Manifesto," 2001, *http://agilemanifesto.org/principles.html*

8 For example, the Payment Card Industry Data Security Standard (PCI DSS) require each change to have documentation of an approval for the change, an assessment of the impact of the change, tests run to validate the change and the status of the testing, and roll-back procedures.