DEVOPS CI/CD-PIPELINE

DevOps bringt die **Softwareentwicklung** (Dev) und den **IT-Betrieb** (Ops) zusammen. Dieses **Zusammenarbeitsmodell** erfordert oft einen Wandel in der Unternehmenskultur. Ein wesentlicher Bestandteil ist dabei **CI/CD** als Abfolge von Prozessschritten um letztendlich ein **lauffähiges Produkt** auszuliefern.

Continuous Integration fokussiert sich auf die Umwandlung von Programmcode in ausführbare Software.

Continuous Delivery beschreibt das Testen und Ausliefern der ausführbaren Software, um sie manuell oder automatisch in Produktion nehmen zu können.

Ziele und Vorteile

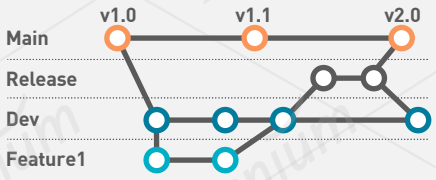
- Häufigere und robustere Deployments
- Schnellere Time-to-Market
- Weniger Fehler in der Software
- Schnelleres Einspielen von Fixes

Code

Quellcode wird in einer **Versionskontrolle** verwaltet. Das Team kann parallel an mehreren Software-Ständen in **Branches** arbeiten.

- Änderungen zwischen Versionen nachvollziehbar
- Gemeinsame einheitliche Standards → Styleguide
- Enthält die Konfiguration der CI/CD Pipeline

Branches beschreiben unterschiedliche Software-Stände. **GitFlow** ist ein gängiges Branching-Konzept:



Main: Stand für Produktivumgebung. Basis für Hotfixes.

Release: Stabile Vorstufe zu Main. Umgebung für zusätzliche Tests und Abnahmen.

Dev: Hauptentwicklung und Testen.

FeatureX: Übergreifende Änderungen, können parallel entwickelt werden.

Build

Sobald er Quellcode aktualisiert wurde, baut der **Build-Server** automatisch die ausführbaren **Software-Artefakte.**

- Statische Code-Analyse findet früh Fehler
- Security-Analyse und Lizenz-Management
- Frühes Feedback vor aufwendigen Tests

0	3	95
Fehler	Warnungen	Code Smells
2	2-	19%
Sicherheitslücken	Wartbarkeit	Testabdeckung

Statische **Code-Analyse** liefert hilfreiche Metriken als frühes Feedback und erleichtert die Überwachung der Definition-of-Done (**DoD**)-**Kriterien**. Unzureichende Werte führen zum Abbruch des Prozesses.

Unit Test

Im Rahmen des **Builds** werden die einzelnen Software-**Module** isoliert von einander **vollautomatisch** durchgetestet.

- Schafft Vertrauen in regressionsfreie Änderungen
- Schnell viele Parameter-Kombinationen abtesten
- Externe Abhängigkeiten werden simuliert (Mock)

```
CODE    function Mittelwert(a, b)
          return a + b/2

TEST 1  test('Mittelwert von 0 und 4 ist 2')
          expect( Mittelwert(0, 4) ).toBe(2)

TEST 2  test('Mittelwert von 2 und 4 ist 3')
          expect( Mittelwert(2, 4) ).toBe(3)
```

Der 2. **Beispiel-Unit-Test** schlägt wegen der vergessenen Klammern fehl: (a + b) / 2. Der 1. Test ist nur wegen der Parameterwahl zufällig erfolgreich.

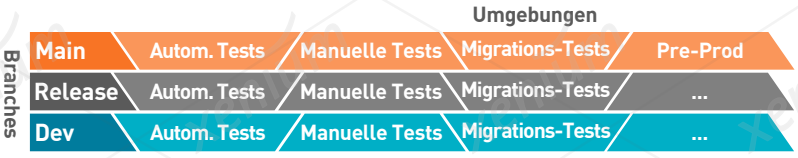
FAILED Mittelwert Test
✓ Mittelwert von 0 und 4 ist 2
✕ **Mittelwert von 2 und 4 ist 3**
- Ergebnis: 4; erwartet: 3

Deploy Internally

Die Software wird nach erfolgreichem **Unit Test** in einer **internen Umgebung** deployed. Dies ermöglicht das **Testen** von übergreifender Funktionalität.

- Umgebungskonfiguration möglichst nah am Betrieb
- Mehrere Umgebungen für verschiedene Branches
- Container für reproduzierbare Deployments

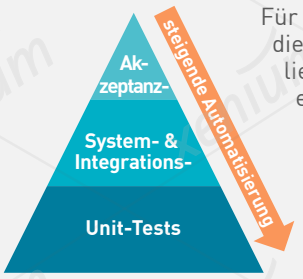
Mehrere **Umgebungen pro Branch** erleichtern das Testen. Komplexere **Migrationen** sollten explizit auf einer eigenen Umgebung getestet werden. Für die Analyse von Produktionsumgebungs-Bugs sollte die **Pre-Prod** diese so weit wie möglich widerspiegeln.



Test

Die **Testdurchführung** für die Gesamtfunktionalität ist weitgehend automatisiert. Tester führen komplexere, sowie abschließende **Akzeptanztests** manuell durch.

- Verständliche Aufbereitung der Testergebnisse
- So weit wie möglich automatisiert
- Umfassender als Unit Tests



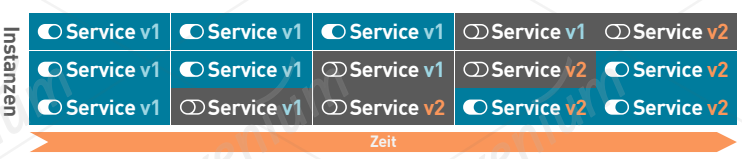
Für ein **erfolgreiches CI/CD** sollte sich die Software stets im Status eines auslieferbaren Produktes befinden. Dies erfordert eine weitgehende **Testautomatisierung**. Die Teststruktur und Automatisierungsgrade veranschaulicht die **Testpyramide**. Manuelle Tests sollten nur vereinzelt durchzuführen sein und sich primär auf **exploratives Testen** fokussieren.

Deploy to Production

Nach Bestehen **aller Tests**, wird die Software auf die **Produktionsumgebung** aufgespielt. Dabei wird ein **Zero-Downtime-Deployment** angestrebt.

- Alt- und Neuversionen kurzzeitig parallel in Betrieb
- Sukzessive Ablösung minimiert Downtime
- Manuelle oder automatische Freigabe

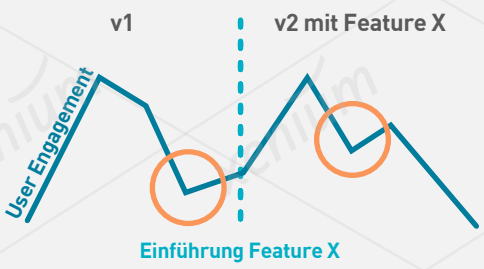
Für ein **Zero-Downtime-Deployment** werden die Service-Instanzen nicht alle auf einmal, sondern **sukzessive** auf die neue Version **migriert** – also deaktivert, aktualisiert und reaktiviert. Kurzzeitig sind beide Versionen in Betrieb.



Monitor

Das Verhalten der Software wird **kontinuierlich überwacht**, um Probleme **proaktiv** zu verhindern und **Verbesserungen** zu finden.

- Sammlung von Telemetriedaten aus versch. Quellen
- Health-Checks, Systemauslastung, Logs für Bug-Analyse/Angriffsversuche/Nutzerverhalten



Monitoring erlaubt die Validierung von **Hypothesen**. Das Beispiel zeigt das User Engagement vor und nach Einführung eines neuen Features. Die Abnahme nach einem Peak fällt geringer aus. Somit ist das Feature hilfreich.

Der Ablauf wird im Rahmen der Entwicklungs-**Iterationen** durchgeführt. Es kann **mehrere parallele Stränge** geben, welche sich jeweils in unterschiedlichen Etappen befinden.

www.xenium.com