CI / CD

- **CI / CD - praktyki** a nie narzędzia!

Referencje

- **Martin Fowler – Continuous Integration (2006):** https://www.martinfowler.com/articles/continuousIntegration.html
- **Podsumowanie:** https://en.wikipedia.org/wiki/Continuous_integration
- **Dodatkowe referencje:** https://martinfowler.com/delivery.html

Referencje

- **Martin Fowler – Continuous Delivery:** https://youtu.be/aoMfbgF2D_4
- **Sanjeev Sharma – Continuous Delivery vs. Continuous Deployment:** https://youtu.be/hQ0recUXk9o
- **ThoughtWorks – Continuous Delivery 101 (Part 1):** https://youtu.be/HnWuljUw_Q8
- **ThoughtWorks – Continuous Delivery 101 (Part 2):** https://youtu.be/a9r-IXLDLvk
- **ThoughtWorks – Continuous Delivery 101 (Part 3):** https://youtu.be/S0g91mryV4c

Referencje

- **Continuous deployment in 5 easy steps:** http://radar.oreilly.com/2009/03/continuous-deployment-5-eas.html
- **Continuous Deployment at IMVU: Doing the impossible fifty times a day**: http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/

Wprowadzenie

Wprowadzenie

7

- **CI** – nazwa zaproponowana w roku **1991** przez **Grady'ego Booch'a** w jego metodzie obiektowego wytwarzania oprogramowania
- CI jako **jedna z 12 praktyk eXtreme Programming** (XP): lokalne testy jednostkowe przed commit'em zmian do repozytorium + częste commit'y (**kilka razy dziennie**)
- Unikanie *integration hell* / *monster merge*

Wprowadzenie

8

- Z czasem – CI z użyciem **serwerów budowania** (automatyzacja):
  - budowanie aplikacji (okresowo lub po zmianie kodu w repozytorium)
  - kontrola jakości: testy integracyjne
  - pomiar wydajności: testy wydajnościowe
  - testy akceptacyjne
  - ekstrakcja dokumentacji z kodu źródłowego
  - ...

Wprowadzenie

9

- **Continuous delivery (CD)**:
  - dalsze **rozszerzenie idei CI**
  - ciągłe zapewnianie, że oprogramowanie jest w **stanie**, **umożliwiającym jego szybkie dostarczenie użytkownikom końcowym** (umożliwiającym wdrożenie w środowisku produkcyjnym)
  - **Wydanie** (release) oprogramowania jest decyzją o charakterze biznesowym

10

*"CD is a continuous run of a **deployment pipeline** that tests if the software is in a state to be delivered"*

*"CD is putting the **release schedule in the hands of the business**, not in the hands of IT"*

Wprowadzenie

11

- **Continuous deployment (CD?)**: ciągły proces **udostępniania najnowszej wersji produktu** użytkownikom końcowym, w środowisku produkcyjnym

12

*„It's not **done**, if it's not **live**"*

CI / CD

- **Automatyzacja** wszystkiego, co da się zautomatyzować:
  - build, testy, generowanie dokumentacji,
  - konfiguracja środowisk uruchomieniowych aplikacji,
  - konfiguracja baz danych (migracje)
  - wdrożenie aplikacji w różnych środowiskach uruchomieniowych: testowym, pre-produkcyjnym (staging environment), produkcyjnym

CI / CD

- Konieczne jest **zarządzanie konfiguracją**! – w repozytorium powinno znaleźć się **wszystko co jest potrzebne do uruchomienia aplikacji** na „czystym" systemie:
  - kod źródłowy
  - schematy baz(y) danych
  - skrypty
  - konfiguracje serwerów
  - ...

Dobre praktyki CI

Dobre praktyki CI

- Używanie **repozytorium kodu źródłowego**: trunk (mainline) – miejsce dla działającej wersji oprogramowania
- **Automatyzacja build'ów** – kompilacja, generowanie dokumentacji, generowanie wersji dystrybuowanej oprogramowania, ...
- **Automatyzacja wykonania testów** na zbudowanym oprogramowaniu (testy jednostkowe, regresyjne, akceptacyjne)

Dobre praktyki CI

- Każdy programista **testuje lokalnie** i następnie commit'uje zmiany **przynajmniej 1 raz dziennie; podział pracy na mniejsze części**, których realizacja trwa kilka godzin
- Każdy **commit** do głównej gałęzi powinien skutkować **zbudowaniem** oprogramowania na serwerze ciągłej integracji (automated CI)

Dobre praktyki CI

- Jeżeli build na serwerze ciągłej integracji się nie powiedzie, należy **jak najszybciej naprawić błędy**
  - Kent Beck: „nobody has a higher priority task than fixing the build"
  - naprawienie błędu – **usunięcie oczywistej przyczyny błędu** lub cofnięcie ostatniej zmiany (**revert**) i **poszukiwanie przyczyn** błędnego builda **lokalnie**

Dobre praktyki CI

19

- Utrzymywanie **niskiego czasu budowania** aby móc szybko reagować na problemy z integracją zmian i móc powtarzać proces wielokrotnie w ciągu dnia
  - **deployment pipeline** – szybki build pierwszego poziomu (tzw. ***commit stage build***), następnie dłuższy build drugiego poziomu (zazwyczaj dłuższe testy)
  - **równoległe** wykonywanie testów

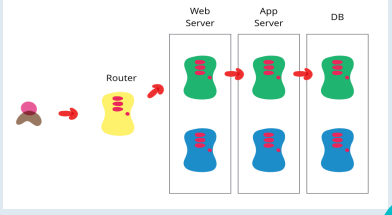
Dobre praktyki CI

20

- Testowanie z użyciem **środowiska "pre-produkcyjnego"**:
  - „separate pre-production environment (*staging*) should be built to be a **scalable version of the actual production environment** to both alleviate costs while maintaining technology stack composition and nuances."
  - np. **blue-green deployment** https://martinfowler.com/bliki/BlueGreenDeployment.html

Dobre praktyki CI

21



Dobre praktyki CI

22

- Udostępnianie **aktualnych wersji** zbudowanych artefaktów (łatwość pobrania najnowszych wersji zbudowanych plików)
- Każdy uczestnik projektu widzi **build status** projektu – np. serwer CI z **interfejsem webowym**

Dobre praktyki CI

23

- **Automatyczne wdrażanie aplikacji**:
  - **w środowisku testowym**, w celu wykonania podstawowych testów
  - **w środowisku pre-produkcyjnym**, w celu wykonania bardziej zaawansowanych testów
  - **w środowisku produkcyjnym** (release), w celu udostępniania użytkownikom końcowym = **continuous deployment**

Dobre praktyki CI

24

4 x DON'T

(https://www.thoughtworks.com/continuous-integration):

- Don't check in **broken code**
- Don't check in **untested code**
- Don't check in when the **build is broken**
- **Don't go home** after checking in until the system builds

25

Zagadnienia CI / CD

26

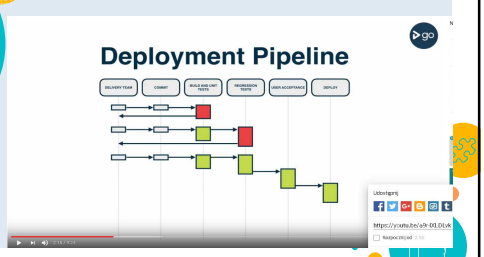
Zagadnienia CI / CD

- Różnica pomiędzy **agile releases** (funkcje pokazywane właścicielowi produktu) a **continuous deployment** (funkcje udostępniane użytkownikom końcowym)
- Różnica między **nightly builds** a **CI**
- Cluster environment – **rolling deployment**
- Interesujący wariant: **deployment of trial builds** (dla podpopulacji użytkowników)
- **Devops culture change** (kooperacja programiści – zespół wdrożeniowy)

27

Zagadnienia CI / CD

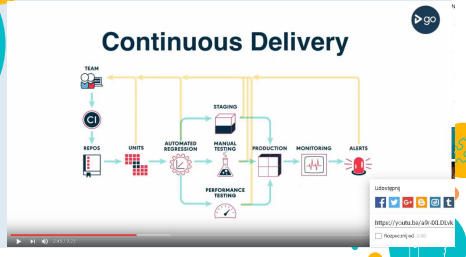
Deployment pipeline (code commit -> production)



28

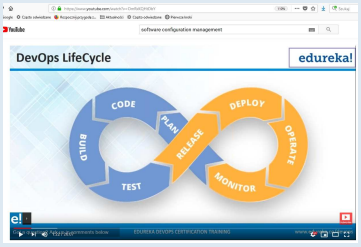
Zagadnienia CI / CD

Sprzężenia zwrotne w potoku wdrażania oprogr.



29

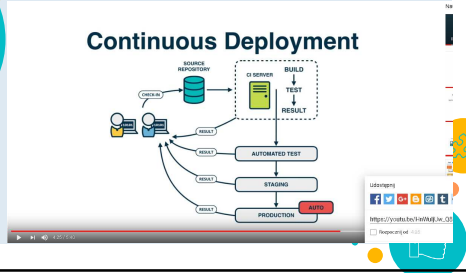
DevOps – cykl życia



30

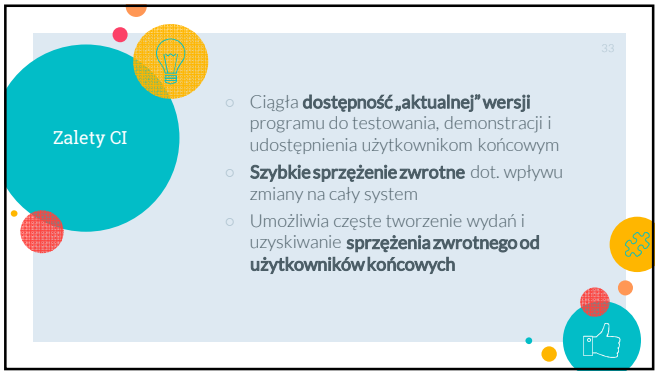
Zagadnienia CI / CD

Continuous deployment




31
Zalety CI


32
Zalety CI
Zmniejszenie ryzyka niepoprawnej integracji zmian w kodzie aplikacji (częste integrowanie małych zmian)
Możliwość śledzenia postępów w projekcie (real progress)
Łatwiejsze lokalizowanie błędów w kodzie gdy build się nie powiedzie
Mniejsza utrata funkcjonalności przy wycofywaniu ostatnich zmian


33
Zalety CI
Ciągła dostępność „aktualnej" wersji programu do testowania, demonstracji i udostępnienia użytkownikom końcowym
Szybkie sprzężenie zwrotne dot. wpływu zmiany na cały system
Umożliwia częste tworzenie wydań i uzyskiwanie sprzężenia zwrotnego od użytkowników końcowych


34
Zalety CI
Możliwość automatycznego wyznaczania różnych miar oprogramowania, np. pokrycia kodu, złożoności kodu
Ukierunkowanie programistów na tworzenie bardziej modularnego kodu


35
Koszty CI


36
Koszty CI (+ propozycje rozwiązań)
Stworzenie i utrzymanie zautomatyzowanych testów
Stworzenie i utrzymywanie systemu budowania kodu (-> Ant, Maven, Gradle, ...)
Kolejkowanie build'ów (-> zrównoleglanie testów)
Konieczność szybkiej naprawy niedziałającego kodu (-> więcej dyscypliny ze strony programistów)

37

Narzędzia

38

Zagadnienia CI

Narzędzia

Some of the tools

SCM	Orchestration	CM	Testing	Deployment
Git Subversion TFS Mercurial Perforce	GoCD Jenkins Bamboo Travis CI Team City	Puppet Chef Ansible Salt CF Engine	Selenium Cucumber JUnit Gauge JMeter FitNesse	Capistrano CA Nolio Rapid Deploy MDT Octopus

Szacowanie pracochłonności – metoda *Wideband Delphi*

Marcin Szeląg, IIn, PP

40

Wideband Delphi

Metoda delficka - wprowadzona w Rand Corporation (koniec lat **40-tych**)

- Początek lat **70-tych** – rozszerzenie do obecnej wersji **Wideband Delphi** (Barry Boehm, *Software Engineering Economics*,Prentice Hall, 1981) – więcej interakcji w procesie szacowania
- Metoda opisana w książce Watts'a Humphrey'a *Managing the Software Process* (Addison Wesley, 1990)
- Karl Wiegers - artykuł *Stop Promising Miracles*, Software Development, February 2000 (http://www.uml.org.cn/SoftWareProcess/pdf/delphi.pdf)

41

Wideband Delphi

Estimation meeting

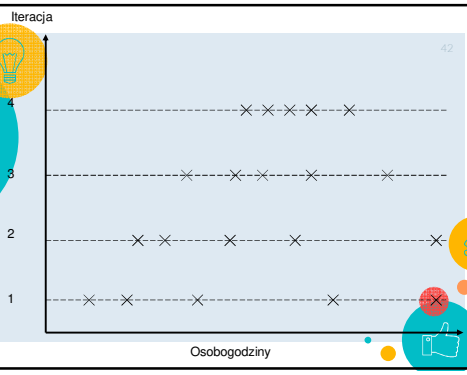
Dążenie do konsensusu ekspertów

Przebieg procesu szacowania:

1. Dostarczenie materiałów ekspertom
2. **Kickoff Meeting** – ustalenie celu i zasad estymacji
3. Każdy ekspert **anonimowo** definiuje **zadania**, **zapisuje założenia** i dokonuje **szacowania zadań**
4. Moderator **anonimowo** agreguje oszacowania – min, max, average i przekazuje reszcie uczestn.
5. Dyskusja
6. Kolejna iteracja - powrót do pkt. 3
7. Zebranie zadań
8. Przegląd uzyskanych wyników, retrospektywa

42

Wideband Delphi – przykład




Wideband Delphi
43
Wady:
Czasochłonna
Konieczna obecność kilku ekspertów
Zalety:
Brak obciążenia
Mniej przeoczeń istotnych zadań
Powstają dodatkowe materiały (lista zadań, przyjęte założenia)


44
Dziękuję!
Pytania?