Programowanie Zespołowe

Systemy kontroli wersji

dr Rafał Skinderowicz
mgr inż. Michał Maliszewski

Systemy kontroli wersji

- Śledzenie zmian, np.: w kodzie źródłowym
- Łączenie zmian dokonanych w plikach
- Ułatwienie współpracy wielu osób nad tymi samymi danymi
- Synchronizacja w czasie
- Architektura modułowa

Rodzaje systemów kontroli wersji

Ze względu na architekturę:

- Lokalne – zapisują dane jedynie na lokalnym komputerze (np. RCS)
- Scentralizowane – oparte o architekturę klient-serwer (np. SVN)
- Rozproszone – oparte o architekturę P2P (np. Git, Mercurial)

SVN vs Git

Dwa najpopularniejsze systemy kontroli wersji.

SVN	Git
• System scentralizowany • Globalne repozytorium danych • Prosty w obsłudze • Stały dostęp do centralnego repozytorium • Wsparcie dla wszystkich platform oraz IDE • Powolny • Brak restrykcji • Intuicyjna numeracja rewizji • Operacje na fragmentach repozytorium (checkout)	• System rozproszony • Lokalne oraz globalne repozytorium • Skomplikowany w obsłudze • Nie wymaga stałego dostępu do głównego repozytorium • Słabe wsparcie platformy Windows • Szybki* • Numeracja rewizji poprzez SHA-1 • Operacje na całym repozytorium (checkout) • Mniejszy rozmiar repozytorium • Łatwiejsze łączenie gałęzi (merge)

GitHub

- Największa społeczność programistów tworzących otwarte oprogramowanie
- Wykorzystuje system kontroli wersji Git
- Darmowy hosting programów typu open source
- Płatne prywatne repozytoria

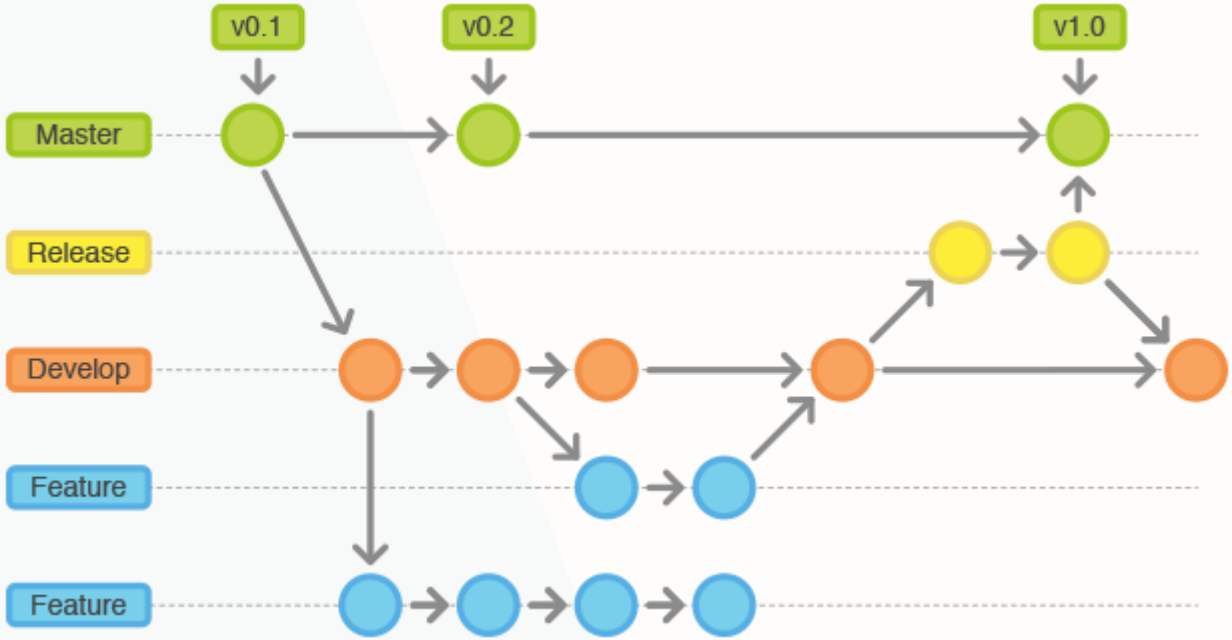
Repozytorium

- Przechowuje dane dotyczące pojedynczego projektu
- W zależności od systemu kontroli wersji może, lecz nie musi zawierać pod-wariantów projektu (trunk, branches, tags)
- Zawiera wszystkie składowe projektu (elementy potrzebne do jego inicjalizacji)
- Nie zawiera skompilowanego kodu źródłowego (zazwyczaj)
- Nie zawiera danych pobieranych z innych źródeł (np. artifactory)

Branches

- Przechowują inne wersje repozytoriów (eksperymenty, wsparcie dla innych platform, wyspecjalizowane wersje dla klientów itp.)
- W przypadku poprawy błędów konieczność zmiany w wielu miejscach
- Podstawowe narzędzie w produkcji oprogramowania w przypadku Gita
- Rozwój fragmentów systemu tzw. „wysokiego ryzyka”
- Możliwość połączenia z głównym repozytorium (!)

Branches



Commit

- Zapisanie w repozytorium zmian nosi nazwę „commit”
- Każdy commit powinien posiadać informacje na temat wprowadzonych zmian
- Commity tworzą historię repozytorium
- W zależności od systemu kontroli wersji commity mogą trafiać bezpośrednio do głównego repozytorium, lub do lokalnych/pośrednich repozytoriów (np.: w celu dokonania *Code Review*)
- Niektóre systemy kontroli wersji pozwalają na cofanie commitów z repozytorium (!)

Revert

- Większość systemów kontroli wersji zezwala na cofnięcie dokonanych w repozytorium zmian
- Prawie zawsze przyjmują one formę nowego commita

Update/Synchronize

- Aktualizacja repozytorium zapisanego lokalnie odbywa się poprzez mechanizm aktualizacji/synchronizacji
- Systemy kontroli wersji dopuszczają aktualizację do najnowszej wersji poszczególnych plików lub do wybranej rewizji
- Przed wykonaniem kolejnego commitu zaleca się zaktualizowanie lokalnej wersji repozytorium w celu rozwiązania ewentualnych konfliktów między wersjami plików

Merge

- Operacja łączenia różnych wersji repozytorium lub pojedynczych plików
- O merge zazwyczaj mówi się w kontekście rozwiązywania konfliktów (tzw. *Merge conflict*), operacji która wymaga uwagi programisty/autora i wybrania właściwego sposobu połączenia pliku (manualnie)

Zadanie 1

- Zarejestruj się w serwisie GitHub (https://github.com)
- Zainstaluj aplikację GitHub (https://desktop.github.com)
- Połącz swoje konto GitHub z aplikacją GitHub Desktop
- Wykonaj dołączony do aplikacji tutorial

Zadanie 2

- Grupy 5-6 osobowe
- Utwórzcie repozytorium testowe
- Dodajcie wszystkim osobom z zespołu prawo odczytu/zapisu (poprzez stronę https://github.com)
- Spróbujcie dodać lub zmodyfikować kilka plików a następnie dołączyć je do głównej gałęzi repozytorium
- Każda osoba musi wykonać co najmniej jednego commita

Projekt zaliczeniowy

Zostaliście poproszeni o stworzenie systemu dla jednego z kin w Katowicach. System będzie używany do rezerwacji biletów przez Internet.

- Utwórzcie repozytorium na platformie GitHub
- Dołączcie wszystkie osoby z zespołu do repozytorium
- Stwórzcie początkowy Product Backlog
- Wybierzcie technologie (najlepiej z zakresu OOP)
- Ustalcie priorytety dla stworzonych US
- Oszacujcie stopień trudności US
- Wykonajcie pierwsze planowanie

Projekt zaliczeniowy c.d.

- Wybierzcie zadania do Sprint Backlogu (1 iteracja)
- Wykorzystajcie Sprint Planning i Sprint Review do poprawy jakości waszej pracy
- Stwórzcie własne DoD
- Stwórzcie kryteria akceptacji dla poszczególnych US
- Postarajcie się komunikować bezpośrednio również poza zajęciami w celu realizacji projektu
- Przetestujcie swoje rozwiązania
- Część modułów może zostać zamockowana (np.: moduł płatności)

Projekt zaliczeniowy c.d.

- Rozdzielcie zadania
- Wykonajcie projekt przy pomocy metodyk zwinnych (Scrum + Kanban)
- Rolę Product Ownera na potrzeby projektów pełni dr Rafał Skinderowicz
- Rolę Scrum Mastera na potrzeby projektów pełni mgr inż. Michał Maliszewski
- Wykorzystajcie wiedzę zdobytą na zajęciach do realizacji projektu
- Projekt obejmuje pracę na zajęciach oraz pracę własną zespołu

Projekt zaliczeniowy c.d.

- Pod koniec każdych zajęć dostarczajcie działające oprogramowanie
- Linki do repozytoriów powinny być dostępne dla prowadzących zajęcia
- Nie bójcie się pytać
- Na każdych zajęciach organizujcie Daily Scrum
- **Deadline** – ostatnie zajęcia w semestrze

Powodzenia ☺

Dziękuję za uwagę