

Zarządzanie kodem programów. Narzędzia do wersjonowania, umożliwiające kooperację zespołów.

Wstęp

W dzisiejszych czasach wytwarzanie oprogramowania odbywa się w niezwykle dynamicznych warunkach. Skuteczne zarządzanie kodem jest kluczowym elementem każdego projektu informatycznego. Niezależnie od jego rozmiaru czy złożoności, prawidłowe wersjonowanie kodu oraz umiejętność współpracy w zespołach są niezbędne do doprowadzenia projektu do końca. W referacie zostanie omówiona istota zarządzania kodem programów oraz różnorodne narzędzia do wersjonowania, które umożliwiają płynną i efektywną kooperację między członkami zespołu programistycznego. W pracy zostaną omówione klasyczne systemy kontroli wersji jak i platformy umożliwiające rozproszoną pracę nad kodem. Analiza tych narzędzi oraz praktyczne wskazówki dotyczące ich wykorzystania, pomogą zrozumieć strategie i techniki niezbędne do efektywnego zarządzania kodem programów w środowisku zespołowym.

Proces wytwarzania oprogramowania

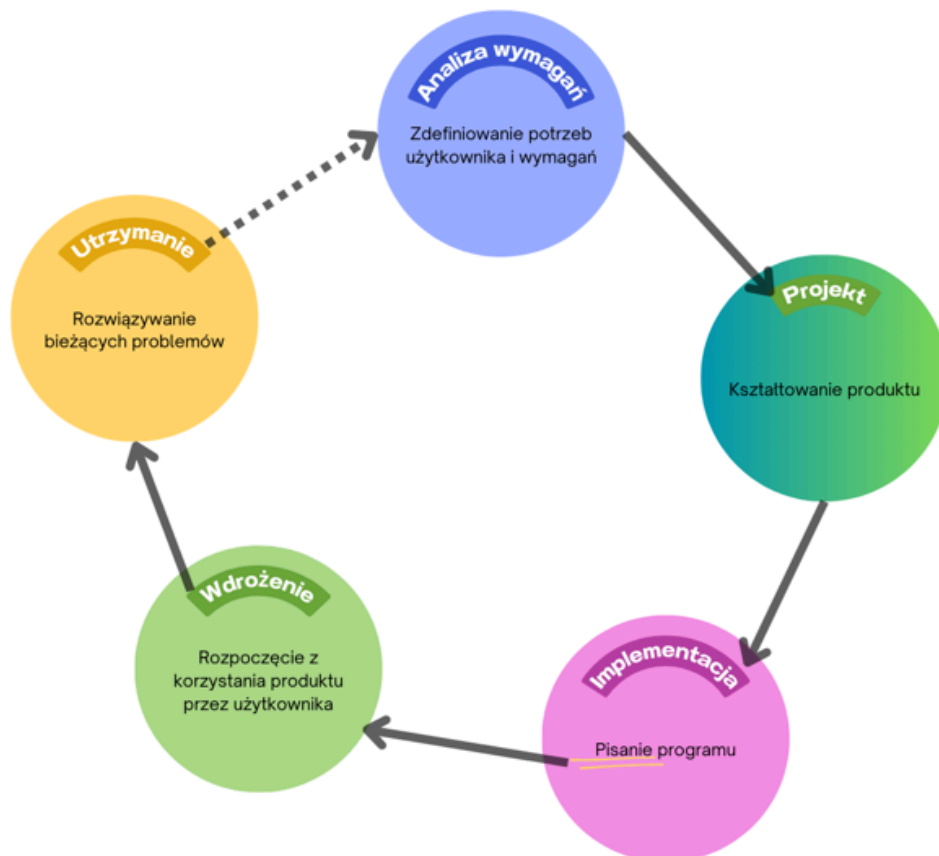
Cykl życia oprogramowania

Cykl życia oprogramowania odzwierciedla kompleksowy proces tworzenia i użytkowania oprogramowania, obejmujący wszystkie jego etapy - od koncepcji przez rozwój, aż po wykorzystanie w praktyce. Można porównać ten proces z mapowaniem drogi prowadzącej od pomysłu do finalnego produktu.

W tym cyklu można dostrzec nie tylko konkretne kroki techniczne, ale także różnorodne role i kompetencje, które współtworzą projekt. Od menedżerów produktu, przez inżynierów oprogramowania po testerów - każdy pełni swoją istotną rolę konieczną do osiągnięcia finalnego efektu w postaci oprogramowania gotowego do użytkowania.

Warto podkreślić, że ten proces rzadko jest liniowy. W trakcie testów i iteracji, oprogramowanie jest ciągle ulepszane, co powoduje, że cykl życia oprogramowania może wielokrotnie powtarzać te same etapy. Istnieje wiele modeli wytwarzania oprogramowania, takich jak Waterfall, metody iteracyjne czy metody zwinne. Każda z tych metod zostaje dobrana zgodnie z cechami projektu, odzwierciedla różnice w podejściu do procesu, specyfikę produktu i charakter działalności firmy.

Wszystkie metody wytwarzania oprogramowania mają wspólny cel - zrozumienie tego, co "stoi" za kodem. Logika i systematyczność procesu pozwalają na skuteczne zarządzanie i tworzenie oprogramowania, niezależnie od wybranej metody.



Obrazek 1 - Cykl życia oprogramowania

Poniżej przedstawiono każdą z faz wytwarzania oprogramowania wraz z krótkim opisem na przykładzie aplikacji.

Określenie wymagań

Pierwszym etapem jest określenie wymagań. Wymagania są definiowane przez doświadczonych specjalistów, którzy korzystają z badań rynku oraz analizują konkurencję. Informacje pozyskane podczas analiz są wykorzystywane podczas planowania całego projektu, przeprowadzenia studium wykonalności czy analizy ryzyka. Specjaliści potrzebni na tym etapie to Analitycy Biznesowi czy Product Ownerzy. Po zaplanowaniu projektu kolejnym etapem jest zdefiniowanie wartości produktu, poprzez określenie jego grupy docelowej - grupy zainteresowanych produktem użytkowników. Posiadając te informacje, architekci oprogramowania mogą „przenieść” informacje zawarte w analizach do wirtualnego świata, projektując funkcjonalności oprogramowania.

Przykładowe role:



Obrazek 2 - Przykładowe role na etapie określenia wymagań

Projekt

Faza projektowania jest drugim krokiem procesu tworzenia oprogramowania, który skupia się na konkretnym kształtowaniu produktu. Tutaj podejmowane są decyzje dotyczące wyglądu i funkcjonalności interfejsu użytkownika. Rozważane jest również, jakie ruchy podejmie użytkownik podczas pierwszego uruchomienia aplikacji i gdzie będzie szukał poszczególnych funkcji.

W tej fazie wybierany jest także język programowania oraz inne techniczne aspekty, takie jak bazy danych i serwery, które najlepiej odpowiadają potrzebom aplikacji. Ważne jest odpowiednie dostosowanie tych elementów, uwzględniając skalę projektu. Na przykład, projekt portalu obsługującego miliardy użytkowników codziennie będzie wymagał innej struktury technicznej niż aplikacja mobilna służąca do jednej konkretnie określonej funkcji.

Przykładowe role:



Obrazek 3 - Przykładowe role na etapie projektu

Implementacja

Faza implementacji to bardzo ważny czas kiedy zaczyna się pisanie programu. Czasami będzie to projekt wykonywany zupełnie od zera z całkiem nowym kodem (wtedy mówimy o tzw. green field development), a innym razem będzie można wykorzystać elementy z innych, istniejących aplikacji (wtedy mamy do czynienia z tzw. legacy code). Programistów specjalizujących się w różnych językach programowania można podzielić na front-endowych (JavaScript, HTML, CSS, React.js, Angular) czy back-endowych (Java, Scala, C, C++, C#.net). W niektórych projektach poszukiwani są także Full Stack Developerzy – czyli tacy, którzy potrafią kodować zarówno po stronie klienta, jak i administrować serwerem czy bazą

danych. Każda aplikacja ma zdefiniowany stack technologiczny, czyli technologie jakie zostały użyte do jej wytworzenia. Stack technologiczny powinien być dopasowany do głównych założeń oprogramowania.

Przykładowe role:



Obrazek 4 - Przykładowe role na etapie implemantacji

Testowanie

Kolejnym etapem jest faza testowania, która pozwala na uzyskanie pewności, że system działa poprawnie. Aplikacja jest poddawana różnorodnym testom, obejmującym funkcjonalność, wydajność i optymalizację, zwłaszcza jeśli ma działać w środowisku o dużej skali. Ważne jest również zapewnienie bezpieczeństwa oprogramowania poprzez sprawdzenie zgodności z wymaganiami.

Testerzy analizują, czy wszystkie funkcje działają zgodnie z oczekiwaniami, korzystając zarówno z testów manualnych (wykonanych przez użytkownika) jak i automatycznych (testy automatyczne tworzone przez testerów). Dodatkowo, użytkownicy mogą brać udział w testowaniu, dostarczając informacji zwrotnej, która pozwala na ciągłe doskonalenie produktu i lepsze dopasowanie do ich potrzeb.

W przypadku wykrycia błędów ("bugów"), często konieczny jest powrót do poprzednich etapów, aby dokonać poprawek w kodzie i zapewnić jakość oprogramowania.

Przykładowe role:



Obrazek 5 - Przykładowe role na etapie implemantacji

Wdrożenie

Deployment - wdrożenie to etap, który oznacza, że oprogramowanie jest gotowe do użycia. Po dokładnym zaplanowaniu, kodowaniu i sprawdzeniu, że wszystko działa zgodnie z oczekiwaniami, oprogramowanie przechodzi do etapu wdrożenia. Oznacza to, że użytkownicy mogą zacząć korzystać z aplikacji.

W początkowej fazie wdrożenia mogą wystąpić drobne błędy, które zostaną szybko wykryte i naprawione. Aplikacje, strony internetowe i inne produkty cyfrowe są poddawane ciągłym zmianom i ulepszeniom. Przykładowo Facebook, przeszedł znaczące zmiany w ciągu ostatnich pięciu lat. Jest to naturalne zjawisko, ponieważ technologia rozwija się, pojawiają się nowe możliwości, a trendy w designie oraz oczekiwania użytkowników także się zmieniają. Dzięki innowacjom produkty i usługi stale ewoluują, aby sprostać potrzebom i oczekiwaniom użytkowników.

Utrzymanie

Ostatnim etapem jest faza utrzymania, gdy aplikacja już działa i użytkownicy z niej korzystają. Pomimo tego, że aplikacja działa, producent musi być czujny, ponieważ mogą pojawić się różne problemy.

W przypadku aplikacji webowych, mogą wystąpić problemy z działaniem na określonych przeglądarkach, być celem ataku złośliwego oprogramowania lub hakera. W miarę wzrostu liczby użytkowników aplikacji, mogą pojawić się problemy z obciążeniem serwerów. W takich sytuacjach konieczne jest podjęcie działań naprawczych, często powracając do odpowiednich etapów cyklu rozwoju oprogramowania.

Naprawa błędów nie zawsze wymaga zaangażowania programistów, ponieważ mogą być to drobne problemy, które można szybko naprawić. Czasem wystarczy wyjaśnić użytkownikowi, jak samemu poradzić sobie z problemem. W takich przypadkach warto skorzystać z pomocy zespołów wsparcia technicznego.

Przykładowe role:



Obrazek 6 - Przykładowe role na etapie utrzymania

Systemy kontroli wersji

System kontroli wersji można opisać jako aplikację wspomagającą monitorowanie modyfikacji w kodzie źródłowym oraz ułatwiającą programistom łączenie zmian wprowadzonych do plików przez różnych użytkowników w różnym czasie.

Systemy kontroli wersji dzielą się na:

- lokalne, pozwalające na zapisanie danych jedynie na lokalnym komputerze, np. SCCS oraz RCS,
- scentralizowane, oparte na architekturze klient-serwer, np. CVS, Subversion,
- rozproszone, oparte na architekturze P2P (Peer to Peer), np. BitKeeper, Code Co-op Git, svn.

Systemy VCS (Version Control System) zapewniają każdej osobie współtworzącej kod ujednolicony i spójny widok projektu, wyświetlając prace, które są już w toku. Wgląd w przejrzystą historię zmian, kto ich dokonał i w jaki sposób przyczyniają się one do rozwoju projektu, pomaga członkom zespołu zachować spójność podczas niezależnej pracy.

W rozproszonym systemie kontroli wersji każdy deweloper ma pełną kopię projektu i jego historii. W przeciwieństwie do popularnych niegdyś scentralizowanych systemów kontroli wersji, DVCS nie wymagają stałego połączenia z centralnym repozytorium.

Git

Git jest najpopularniejszym rozproszonym systemem kontroli wersji. Git jest powszechnie używany zarówno do tworzenia oprogramowania open source, jak i komercyjnego, przynosząc znaczące korzyści osobom, zespołom i firmom.

Historia Gita

Git jest stosunkowo nowym narzędziem, które stanowi przykład tego jak potrzeba może stymulować innowacje. W 2005 roku twórca BitKeeper, narzędzia do kontroli wersji, używanego przez osoby rozwijające Linuksa, obwieścił iż jego oprogramowanie nie będzie już w pełni darmowe. W odpowiedzi na to Linus Torvalds i inni liderzy projektu postanowili opracować własne narzędzie do zarządzania wersjami, które będzie odznaczać się cechami BitKeepera, a jednocześnie będzie miało otwarty kod. W ten sposób powstał Git, który niezwykle szybko zdobył popularność. Co ciekawe, w 2016 roku BitKeeper stał się oprogramowaniem otwartym, choć wielu uważa, że ten krok nastąpił zbyt późno, by mogło ono konkurować z młodszym rywalem, jakim jest Git autorstwa Torvaldsa.



Obrazek 7 - Linus Torvalds

Git a Github

Kluczową różnicą między Gitem a GitHubem jest to, że Git jest darmowym narzędziem kontroli wersji typu open source, które programiści instalują lokalnie na swoich komputerach osobistych, natomiast GitHub jest płatną usługą online stworzoną do uruchamiania Git w chmurze.

Git to oprogramowanie. GitHub to internetowa usługa SaaS (Software as a Service). Jednak pomimo tego rozróżnienia, Git i GitHub nie są konkurencyjnymi ofertami. Zamiast tego współpracują ze sobą i wzajemnie się uzupełniają.

Git to proste i łatwe w użyciu narzędzie do rozproszonej kontroli wersji.

Programiści mogą wykonywać migawki swojego kodu w różnych momentach, tworząc historię wersji, która odwzorowuje proces tworzenia oprogramowania. (Deweloperzy mogą wspólnie udostępniać te migawki kodu lub zatwierdzenia dowolnej liczbie innych deweloperów z dowolnego miejsca na świecie.

Git

- stworzony w 2005 roku
- jest instalowany i utrzymywany lokalnie na komputerze
- używany do kontroli wersji
- śledzi zmiany plików

GitHub

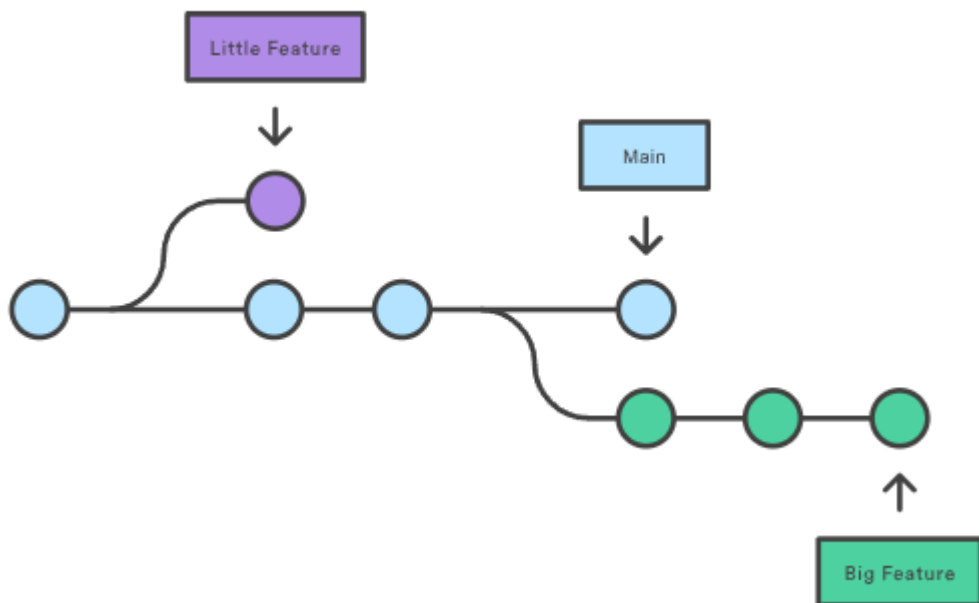
- opiera się na chmurze
- używany do hostowania repozytoriów gitowych
- posiada interfejs, który pozwala na wgląd w pliki



Obrazek 8 - Git a Github

Branche

Prawie każdy system kontroli wersji oferuje funkcję tworzenia gałęzi. Rozgałęzienie polega na odseparowaniu się od głównej linii rozwoju, umożliwiając kontynuację pracy bez wprowadzania chaosu do głównego projektu. W wielu narzędziach kontroli wersji jest to dość zasobożerny proces, który często wymaga stworzenia nowej kopii katalogu z kodem. W przypadku dużych projektów może to zajmować znaczną ilość czasu. Gałąź to niezależna ścieżka programowania. Służy jako warstwa abstrakcji dla edytowania, przechowywania i zatwierdzania zmian. Można ją postrzegać jako metodę na utworzenie nowego katalogu roboczego, strefy przejściowej i historii projektu. Nowe commity są zapisywane w historii aktualnej gałęzi, co powoduje rozgałęzienie (fork) w historii projektu.



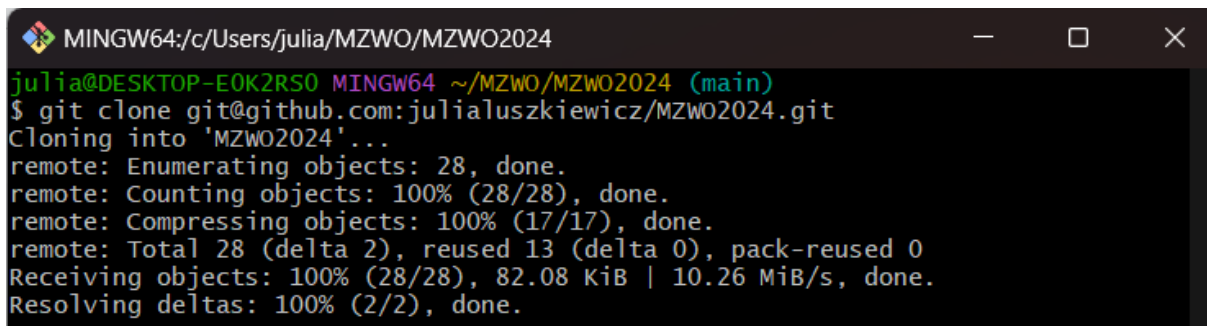
Obrazek 9 - Branche

Podstawowe polecenia w Gicie

W celu dodawania plików na zdalne repozytorium programiści często korzystają z aplikacji - emulatora, który pozwala na wykonywanie poleceń zrozumiałych dla Gita (jeśli używają systemu operacyjnego Windows). Użytkownicy Linuxa lub MacOS nie muszą instalować emulatora.

Poniżej opisano kilka najważniejszych komend gitowych:

- git init - tworzenie pustego repozytorium,
- git clone - pobieranie projektu ze zdalnego repozytorium, które znajduje się na platformie GitHub,
- git checkout nazwa_brancha - przełączenie się na inną gałąź,
- git status - podanie informacji na temat tego co aktualnie dzieje się w repozytorium,
- git add ścieżka_do_pliku - dodanie pliku z lokalnej maszyny na zewnętrzne repozytorium,
- git commit -m "opis_commita" - “zapisanie” pliku z najnowszymi zmianami w kodzie z komentarzem,
- git push origin nazwa_brancha - wysłanie zmian do zewnętrznego repozytorium,
- git pull origin nazwa_brancha - pobranie aktualnej wersji brancha.



```
MINGW64:/c/Users/julia/MZWO/MZWO2024
julia@DESKTOP-E0K2RS0 MINGW64 ~/MZWO/MZWO2024 (main)
$ git clone git@github.com:julialuszkiewicz/MZWO2024.git
Cloning into 'MZWO2024'...
remote: Enumerating objects: 28, done.
remote: Counting objects: 100% (28/28), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 28 (delta 2), reused 13 (delta 0), pack-reused 0
Receiving objects: 100% (28/28), 82.08 KiB | 10.26 MiB/s, done.
Resolving deltas: 100% (2/2), done.
```

Obrazek 10 - Klonowanie repozytorium przy użyciu klucza SSH

SVN

Subversion (znane również jako SVN) to system kontroli wersji stworzony jako następca CVS. SVN jest darmowym i otwartym oprogramowaniem, udostępnianym na licencji Apache. W przeciwieństwie do Gita jest to system lokalny. SVN pozwala kilku użytkownikom na jednoczesną pracę nad tym samym projektem, a także na łatwe powrót do wcześniejszych wersji plików w razie potrzeby. SVN jest zatem narzędziem niezbędnym dla wielu programistów i innych osób pracujących nad plikami tekstowymi.

Historia SVN

SVN został stworzony jako następca popularnego systemu CVS, który miał liczne wady i ograniczenia. Głównym celem SVN było zapewnienie lepszej wydajności, większej stabilności i bezpieczeństwa oraz prostszej obsługi. System ten został zaprojektowany z myślą o dużych projektach, w których wiele osób pracuje nad tym samym kodem źródłowym. Dzięki SVN każdy członek zespołu może śledzić zmiany, wprowadzać swoje poprawki i wersjonować kod w sposób uporządkowany i kontrolowany. W kolejnych latach SVN przeszedł liczne zmiany i ulepszenia, aż do osiągnięcia obecnej wersji 1.14. Pozostaje jednym z najpopularniejszych systemów kontroli wersji i jest szeroko wykorzystywany w wielu projektach na całym świecie.

Jak działa SVN

SVN działa na zasadzie centralnego repozytorium, do którego użytkownicy mogą przysyłać (commitować) nowe pliki lub zmieniać istniejące, a także pobierać najnowsze wersje plików (update). Każda zmiana w plikach jest rejestrowana i oznaczana unikalnym numerem rewizji, co umożliwia śledzenie historii zmian i ewentualne cofanie się do wcześniejszych wersji. SVN posiada również mechanizmy zarządzania konfliktami zmian, które mogą wystąpić, gdy dwóch użytkowników modyfikuje ten sam plik jednocześnie, oraz narzędzia do tworzenia i zarządzania gałęziami (branches) projektu.

SVN jest dostępny na wielu platformach i systemach operacyjnych, w tym Windows, Linux, macOS i innych. Można korzystać z niego za pomocą wiersza poleceń lub za pomocą specjalnych narzędzi graficznych, takich jak TortoiseSVN czy SmartSVN. SVN jest również

zintegrowany z wieloma środowiskami programistycznymi (IDE), takimi jak Eclipse, Visual Studio czy PyCharm, co umożliwia wygodne i efektywne korzystanie z niego w codziennej pracy. Jako system otwartoźródłowy dostępny za darmo, SVN jest atrakcyjnym wyborem dla wielu użytkowników.



Obrazek 11 - Logo SVN

Wady SVN

Oprócz zalet, SVN ma również pewne wady i ograniczenia. Jako system oparty na centralnym repozytorium, wszystkie zmiany muszą być wprowadzane za jego pośrednictwem, co może stanowić problem, gdy repozytorium jest niedostępne lub gdy potrzebna jest szybka edycja pliku poza repozytorium. SVN nie pozwala również na tworzenie lokalnych kopii (tzw. "forks") projektu, co może utrudniać pracę nad różnymi gałęziami. W porównaniu do innych systemów kontroli wersji, takich jak Git, SVN może być mniej elastyczny i wolniejszy. Te aspekty warto rozważyć przy wyborze odpowiedniego systemu kontroli wersji dla swoich potrzeb.

Porównanie Git a SVN

Git i SVN oferują różne podejścia do zarządzania kodem. Git jest bardziej elastyczny i wspiera pracę rozproszoną, podczas gdy SVN, będąc systemem scentralizowanym, jest

bardziej odpowiedni dla projektów, które wymagają centralnego zarządzania. Wybór odpowiedniego systemu zależy od specyfiki projektu i potrzeb zespołu.

Podsumowanie

W dzisiejszym dynamicznym środowisku tworzenia oprogramowania, zarządzanie kodem i efektywna współpraca zespołów są kluczowe dla sukcesu projektów. Referat omawia znaczenie kontroli wersji oraz różnorodne narzędzia wspierające kooperację w zespołach programistycznych. Przedstawiono zarówno klasyczne systemy kontroli wersji, jak i nowoczesne platformy do pracy rozproszonej, oferując praktyczne wskazówki i strategie zarządzania kodem.

Źródła:

<https://talentplace.pl/blog/dla-rekruterow/cykl-tworzenia-oprogramowania-czyli-co-warto-wiedziec-rekrutujac-w-it/>

<https://mylovieview.pl/fototapeta-cykl-zycia-oprogramowania-nr-555C5D0>

<https://docs.github.com/en/get-started/using-git/about-git>

https://pl.wikipedia.org/wiki/System_kontroli_wersji

https://www.w3schools.com/git/git_intro.asp?remote=github

<https://www.dobreprogramy.pl/git-system-kontroli-wersji-czyli-szara-eminencja-swiata-it,6628568271157377a>

<https://www.theserverside.com/video/Git-vs-GitHub-What-is-the-difference-between-them>

<https://git-scm.com/book/pl/v2/Ga%C5%82%C4%99zie-Gita-Czym-jest-ga%C5%82%C4%85%C5%BA>

<https://www.atlassian.com/pl/git/tutorials/using-branches>

<https://pl.wikipedia.org/wiki/Subversion>

<https://boringowl.io/blog/svn>