

Zarządzanie konfiguracją oprogramowania

Configuration management

część 1

Zagadnienia zarządzania konfiguracją



- Zarządzanie zmianą
- Zarządzanie wersjami
- Budowanie systemu
- Zarządzanie wydaniami

Zarządzanie konfiguracją (Configuration Management - CM)

- Oprogramowanie podlega ciągłym zmianom
 - System może być postrzegany jako zbiór wersji, z których każda musi być zarządzana i wspierana.
 - Wersje reprezentując implementacje zatwierdzonych propozycji zmian, adaptacji dla różnego typu sprzętu czy systemów operacyjnych
 - Zarządzanie konfiguracją związane jest ze strategią, procesami oraz narzędziami pozwalającymi na zarządzanie zmieniającym się systemem.
 - Bez zarządzania konfiguracją trudno byłoby odpowiedzieć na takie pytania jak:
 - Jakie zmiany zostały wprowadzone w tej wersji systemu
 - Jakie wersje komponentów są elementami tej wersji systemu

Czynności Zarządzania Konfiguracją (1 z 4)

■ Zarządzanie zmianą

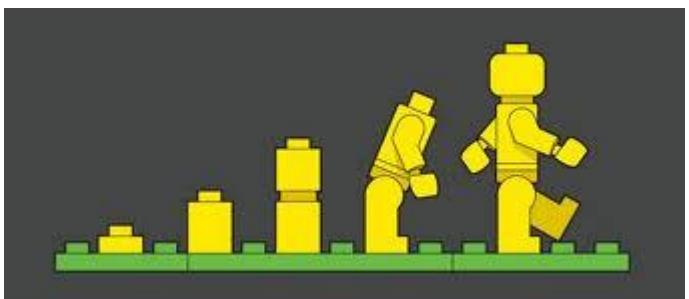
- Śledzenie zgłoszeń dotyczących potrzeby zmiany oprogramowania (od klientów, deweloperów, ...).
- Określanie kosztów oraz wpływu zmiany.
- Podejmowanie decyzji o implementacji lub odrzuceniu zmiany



Czynności Zarządzania Konfiguracją (2 z 4)

■ Zarządzanie wersjami

- Śledzenie zmieniających się wersji komponentów systemu
- Zapewnianie środowiska, w którym praca różnych deweloperów nie interferuje ze sobą



Czynności Zarządzania Konfiguracją (3 z 4)



■ Budowanie systemu

- Proces łączenia komponentów programu, danych, bibliotek do postaci wynikowego systemu
 - Konwersja kodu do postaci wynikowej

```
./configure --enable-maintainer-mode --without-maildir -c
checking for a thread-safe malloc... yes
checking for gawk... gawk
checking whether make sets $NAME... yes
checking if libtool bootstrap functionality is wanted... no
checking whether libtool compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether we have a working C++ compiler... yes
checking for c++ compiler option to accept ISO C99... none needed
checking for style of include used by make... GNU
checking dependency style of colordiff... gcc3
checking whether colordiff and colr understand -n and -w together... yes
checking whether make sets $NAME... loaded yes
checking for a sed that does not truncate output...
checking for yawsales in /usr/local/lib... yes
checking how to run the C preprocessor... colordiff -B
checking for grep that handles long lines and -e... /usr/bin/grep -E
checking for egrep... /usr/bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking phonewall usability... yes
checking phonewall presence... yes
checking for phonewall... yes
checking for size_t... yes
checking for pid_t... yes
checking for uid_t... yes
checking for mode_t... yes
checking for struct... yes
checking that generated files are never older than configure... done
configuring creating ./configurestatus
configuring creating Makefile
configuring executing makefiles commands
```

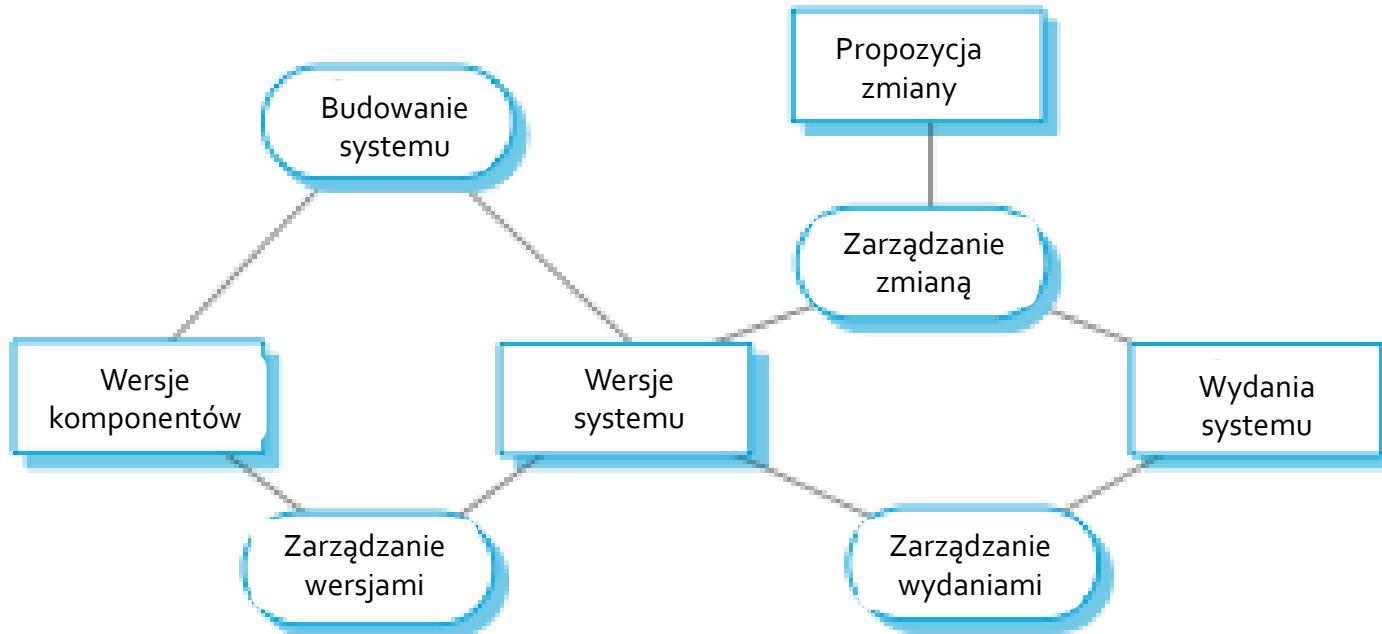
Czynności Zarządzania Konfiguracją (4 z 4)

■ Zarządzanie wydaniami

- Przygotowanie oprogramowania do wydania zewnętrznego
- Śledzenie wersji systemu, które zostały dostarczone klientom.



Czynności Zarządzania Konfiguracją



Terminologia zarządzania konfiguracją - 1



Termin	Wyjaśnienie
Jednostka konfiguracji lub jednostka konfiguracji oprogramowania (ang. configuration item - CI software configuration item - SCI)	Każdy element powiązany z wytwarzaniem oprogramowania (projekt, kod źródłowy, dane testowe, dokument, itp.) który został poddany kontroli konfiguracji. Jednostki konfiguracji mogą istnieć w różnych wersjach. Każda jednostka konfiguracji ma unikatową nazwę.
Kontrola konfiguracji (ang. Configuration control)	Proces zapewniania, że wersja systemu oraz komponentów została zapisana i jest zarządzana. Oznacza to, że każda wersja systemu oraz jego komponentów jest identyfikowalna i zapisana w sposób trwałym (dostępna przez cały czas życia systemu).
Wersja (ang. Version)	Instancja jednostki konfiguracji, różniąca się od innych instancji tej samej jednostki. Wersja zawsze ma unikatowy identyfikator (np. Nazwa jednostki konfiguracji + numer wersji).
Linia bazowa (ang. Baseline)	Zbiór wersji komponentów tworzących system. Linia bazowa jest kontrolowana – wersje komponentów nie mogą zostać zmienione. Oznacza to, że istnieje możliwość ponownego utworzenia linii bazowej z odpowiednich wersji komponentów.
Linia kodu (ang. Codeline)	Zbiór wersji komponentów oprogramowania oraz innych jednostek konfiguracji, od których zależy dany komponent.

Terminologia zarządzania konfiguracją - 2



Termin	Wyjaśnienie
Linia główna/rozwojowa (ang. Mainline)	Sekwencja linii bazowych reprezentujących różne wersje systemu.
Wydanie (ang. Release)	Wersja systemu, która została udostępniona do użytku klientom (lub innym użytkownikom w ramach organizacji).
Przestrzeń robocza (ang. Workspace)	Prywatna przestrzeń, w obrębie której dokonanie modyfikacji oprogramowania nie ma wpływu na pracę innych deweloperów, którzy również pracują na oprogramowaniem.
Utworzenie gałęzi (ang. Branching)	Powołanie do życia nowej linii kodu (ang. codeline) na podstawie wybranej wersji w istniejącej linii. Tak utworzona linia może być rozwijana niezależnie.
Łączenie (ang. Merging)	Utworzenie nowej wersji komponentu oprogramowania poprzez połączenie oddzielnych wersji znajdujących się w różnych liniach kodu. Linie te mogły powstać poprzez uprzednio utworzenie gałęzi.
Budowanie systemu (ang. System building)	Utworzenie wykonywalnego systemu poprzez skompilowanie i połączenie odpowiednich wersji komponentów oraz bibliotek wchodzących w skład systemu.

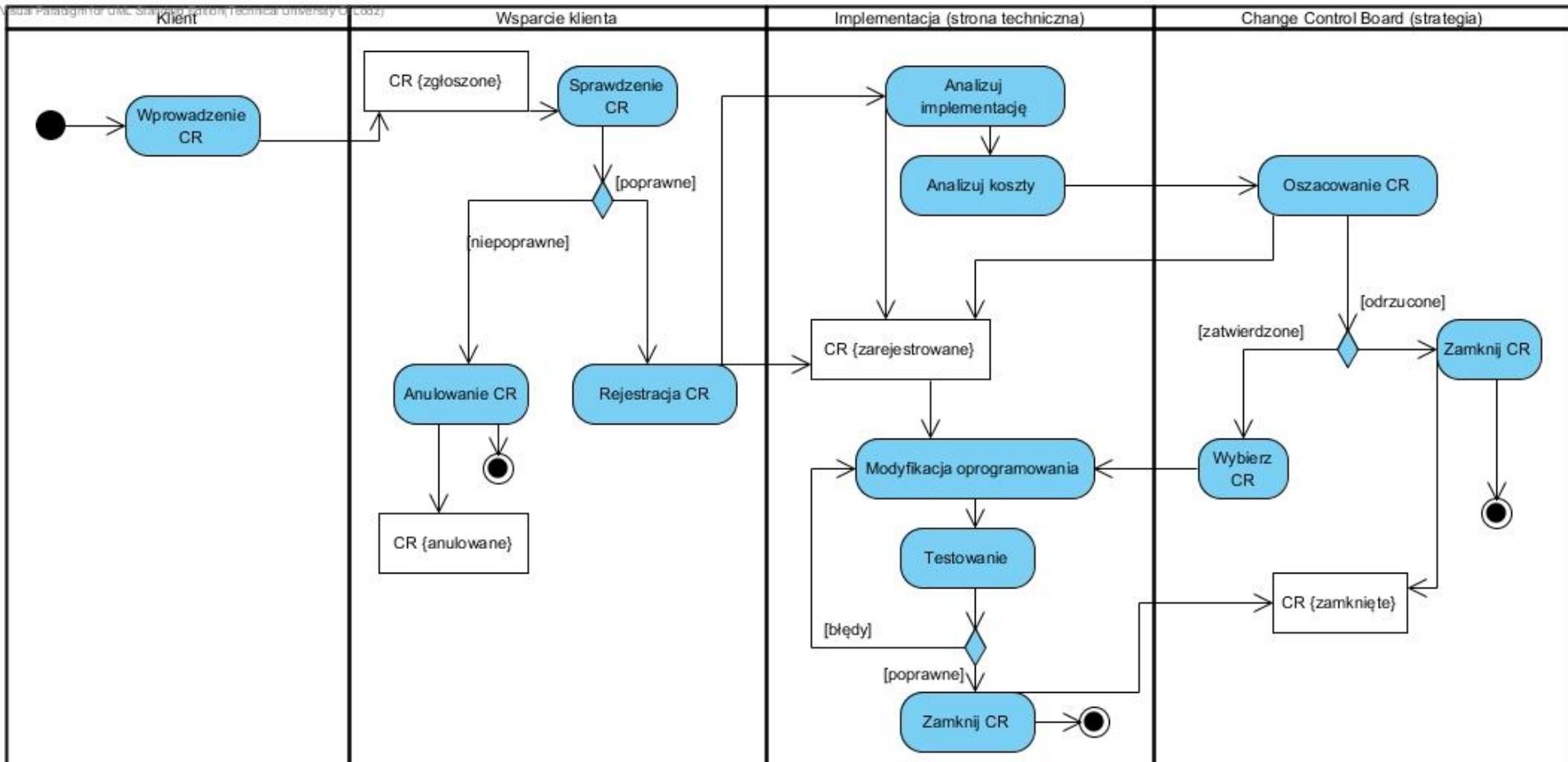
Zarządzanie zmianą

Zarządzanie zmianą



- Potrzeby organizacyjne, zmiany w wymaganiach pojawiające się w trakcie życia systemu, błędy wymagające naprawy, adaptacja systemu do środowiska.
- Celem zarządzania zmianą jest zapewnienie, że ewolucja systemu jest procesem kontrolowanym oraz, że najwyższy priorytet przypisany jest zawsze zmianom najpiśniejszym i najbardziej efektywnym z punktu widzenia kosztów.
- Proces zarządzania zmianą związany jest z:
 - Analizą kosztów i potencjalnych zysków z wprowadzenia proponowanej zmiany.
 - Zatwierdzaniem proponowanej zmiany (jeżeli jest tego warta)
 - Śledzeniem wymaganych zmian w określonych komponentach systemu (związanych z zaakceptowaną propozycją zmiany)

Proces zarządzania zmianą



Formularz zgłoszenia zmiany

Change Request Form

Project: SICSA/AppProcessing

Number: 23/02

Change requester: I. Sommerville

Date: 20/01/09

Requested change: The status of applicants (rejected, accepted, etc.) should be shown visually in the displayed list of applicants.

Change analyzer: R. Looek

Analysis date: 25/01/09

Components affected: ApplicantListDisplay, StatusUpdater

Associated components: StudentDatabase

Change assessment: Relatively simple to implement by changing the display color according to status. A table must be added to relate status to colors. No changes to associated components are required.

Change priority: Medium

Change implementation:

Estimated effort: 2 hours

Date to SGA app. team: 28/01/09

CCB decision date: 30/01/09

Decision: Accept change. Change to be implemented in Release 1.2

Change implementor:

Date of change:

Date submitted to QA:

QA decision:

Date submitted to CM:

Comments:

Czynniki brane pod uwagę w oszacowaniu zmiany



1. Konsekwencje nie wprowadzenia zmiany
2. Zyski wynikające z wprowadzenia zmiany
3. Liczba użytkowników, na których zmiana będzie miała wpływ
4. Koszt wprowadzenia zmiany
5. Wpływ zmiany na cykl wydań produktu

Zarządzanie zmianą w metodykach zwinnych (ang. agile)

- W niektórych metodykach klienci są bezpośrednio zaangażowani w zarządzanie zmianą (np. XP).
- Proponują oni zmianę w wymaganiach a następnie współpracują z zespołem w celu określenia wpływu zmiany oraz decydują czy zmiana powinna być bardziej priorytetowa niż cechy zaplanowane w kolejnym przyroście (inkrementacji) systemu.
- Zmiany związane udoskonalaniem systemu są po stronie programistów.
- Refaktoryzacja, w ramach której oprogramowanie jest udoskonalane w sposób ciągły, nie jest postrzegana jako dodatkowe obciążenie, ale jako niezbędna część procesu rozwoju systemu.

Historia wynikania



// SICSA project (XEP 6087)

//

// APP-SYSTEM/AUTH/RBAC/USER_ROLE

//

// Object: currentRole

// Author: R. Looek

// Creation date: 13/11/2009

//

// © St Andrews University 2009

//

// Modification history

Version	Modifier	Date	Change	Reason
1.0	J. Jones	11/11/2009	Add header	Submitted to CM
1.1	R. Looek	13/11/2009	New field	Change req. R07/02

Narzędzia zarządzania zmianą

- Śledzenie zgłoszeń
 - *bug/defect tracking, issue-tracking systems*
 - Bugzilla, Trac, Redmine, JIRA, Zentrac, Pivotal Tracker, FogBugz, Lighthouse

Zarządzanie wersjami

Zarządzanie wersjami

- Proces śledzenia różnych wersji komponentów oprogramowania lub innych jednostek konfiguracji oraz systemów, w których zostały wykorzystane.
- Zapewnienie, że zmiany dokonane w tych samych komponentach przez różnych deweloperów nie wpływają na siebie nawzajem.
- Proces zarządzania wersjami jest tak procesem zarządzania liniami kodu oraz liniami bazowymi.

Linie kodu i linie bazowe

- **Linia kodu** (ang. codeline) jest sekwencją wersji kodu źródłowego, w której późniejsze wersje są wyprowadzone z wersji wcześniejszych.
 - Linia kodu związana jest z komponentami systemu. W jej ramach istnieją różne wersje komponentów.
- **Linia bazowa** (ang. baseline) jest definicją systemu
 - Linia bazowa określa wersje komponentów włączanych do systemu wraz z wersjami bibliotek, plików konfiguracyjnych, itp.

Linie kodu i linie bazowe



Codeline (A)



Codeline (B)



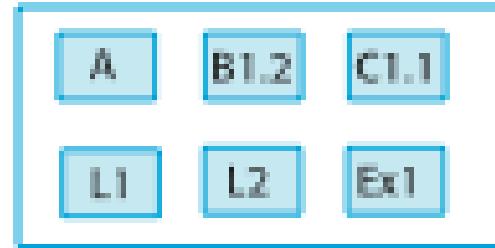
Codeline (C)



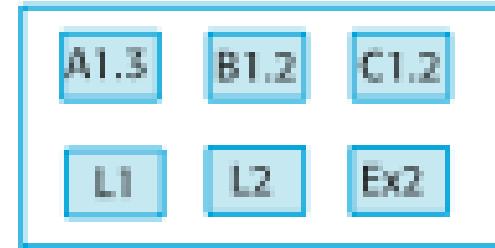
Libraries and external components



Baseline - V1



Baseline - V2



Mainline

Linie bazowe

- Mogą być określane z wykorzystaniem języka konfiguracji, który pozwala definiować, które komponenty są włączane do określonej wersji systemu.
- Linie bazowe są istotne ponieważ bardzo często pojawia się potrzeba ponownego utworzenia określonej wersji kompletnego systemu.
 - Na przykład istnieje linia produkcyjna systemu dla różnych klientów, jeden z klientów zgłasza błąd – powstaje potrzeba ponownego utworzenia wersji systemu w celu weryfikacji.

Systemy zarządzania wersjami

(1 z 2)



1. **Identyfikacja wersji oraz wydania**
 - Wersjom będącym pod kontrolą systemu są przypisywane identyfikatory w momencie wprowadzania ich do systemu.
2. **Zarządzanie przestrzenią przechowującą dane**
 - W celu minimalizacji przestrzeni wymaganej do przechowywania wersji, które często różnią się w nieznaczny sposób, system zarządzania wersjami udostępnia dedykowane mechanizmy składowania.
3. **Zapisywane historii zmian**
 - Każda zmiana w kodzie komponentu będącego pod kontrolą systemu jest zapamiętywana i możliwa do przejrzenia.

Systemy zarządzania wersjami (2 z 2)



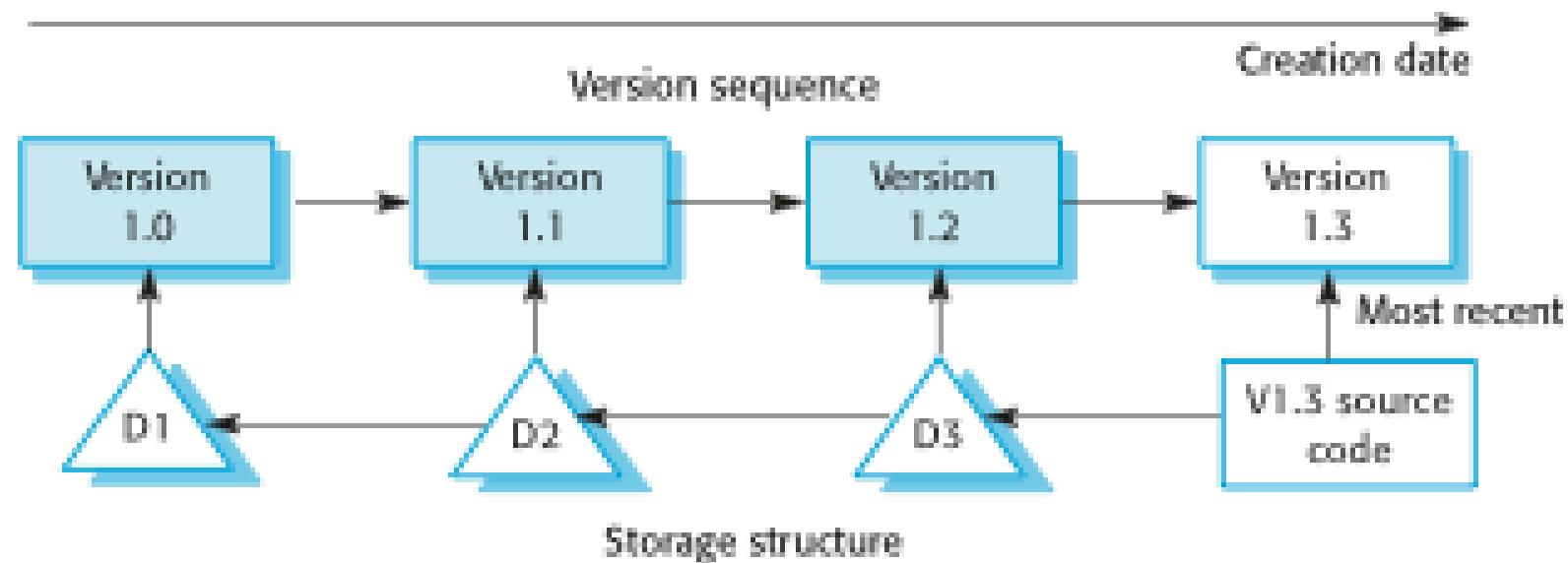
4. Niezależny rozwój

- System zarządzania wersjami śledzi komponenty, które zostały pobrane (ang. check out) do edycji w przestrzeni roboczej programisty. System zapewnia, że zmiany w komponencie wykonane przez jednego programistę nie będą miały wpływu na pracę innych programistów.

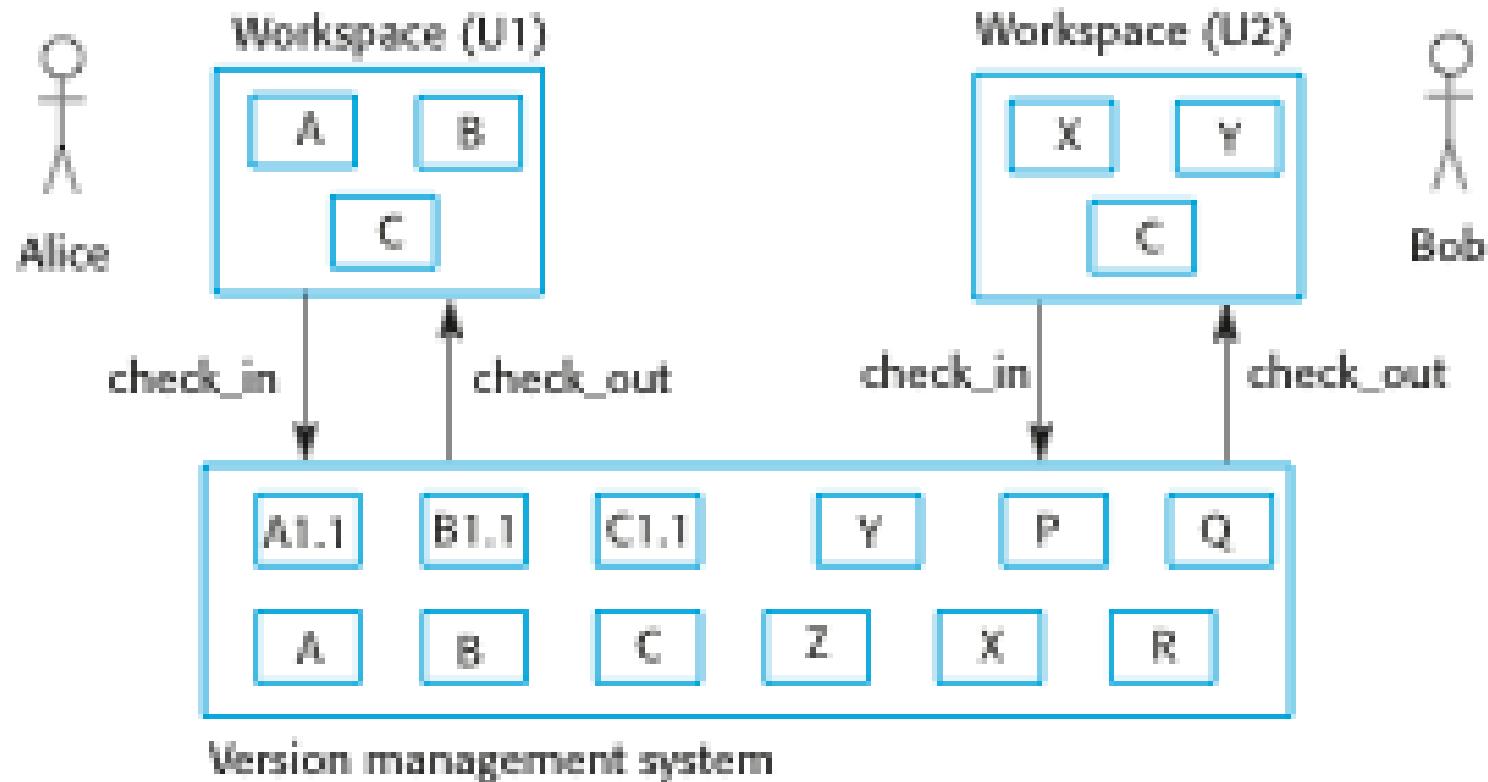
5. Wsparcie dla projektów

- System zarządzania wersjami może wspomagać rozwijanie kilku projektów, które współdzielą komponenty.

Zarządzanie składem wersji z wykorzystaniem delt



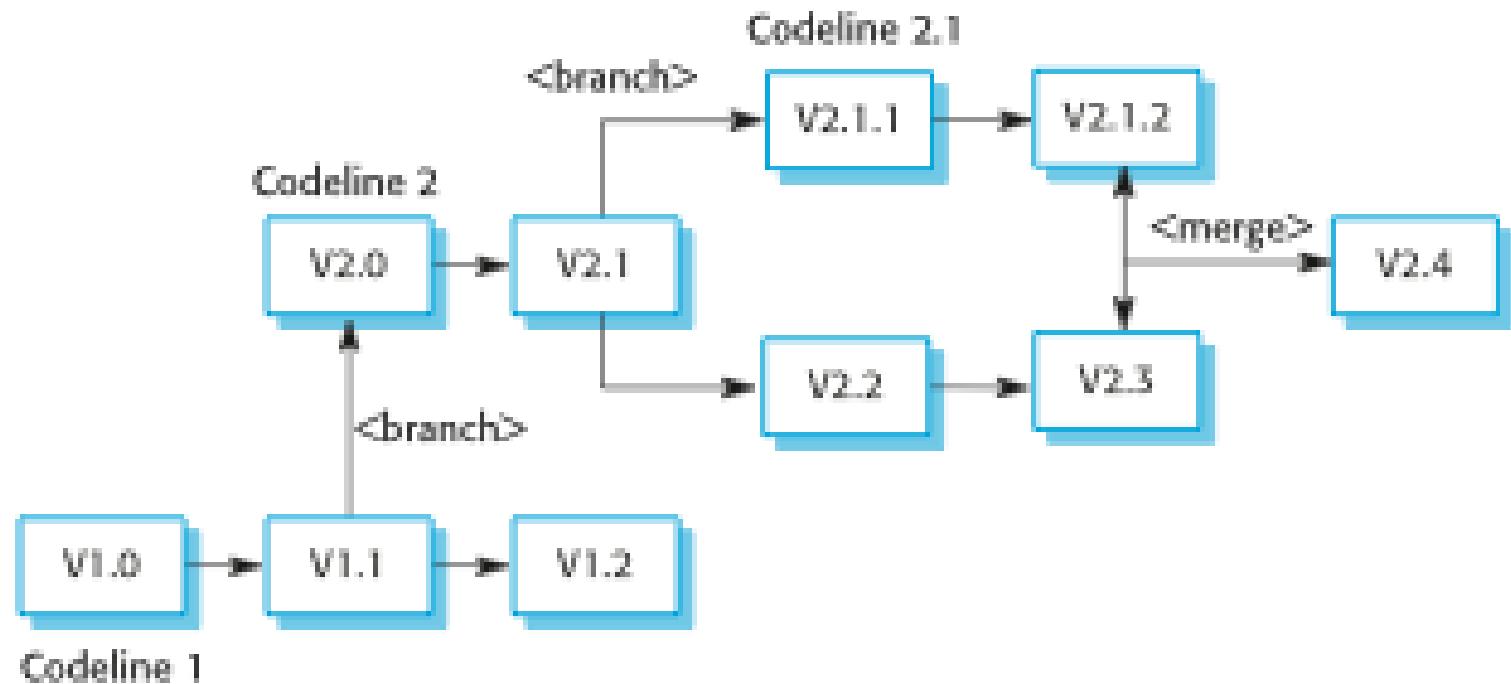
Pobieranie wersji i wprowadzanie zmian do repozytorium wersji



Gałęzie linii kodu

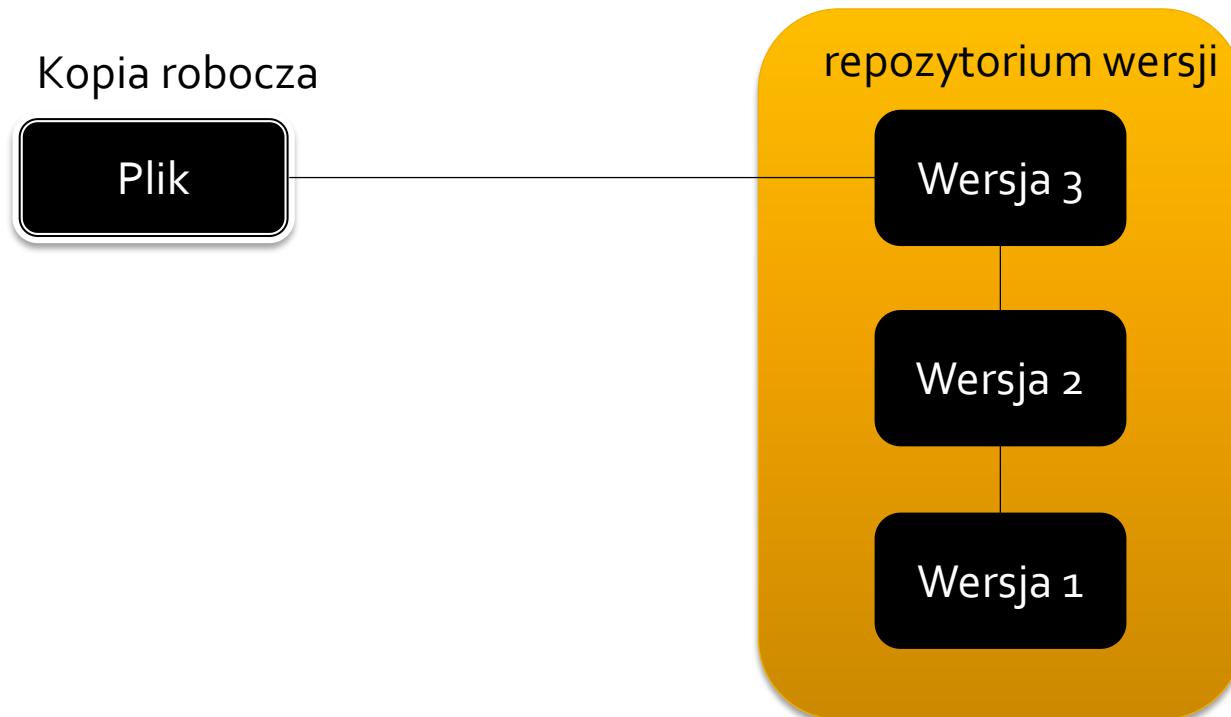
- Posiadanie jednej sekwencji wersji (linii kodu) nie jest zazwyczaj wystarczające. Często istnieje potrzeba posiadania kilku równoległych linii (gałęzi – ang. **branches**).
 - Różni programiści pracują nad różnymi wersjami kodu źródłowego zmieniając go w różny sposób.
- Aby utworzyć komponent zawierający zmiany wprowadzane w równoległych gałęziach należy je połączyć (ang. **merge**)
 - Jeżeli zmiany dotyczyły różnych części kodu, wersje komponentu mogą zostać połączone automatycznie poprzez połączenie różnic (delt) w trakcie tworzenia wersji połączonej.

Tworzenie i łączenie rozgałęzień



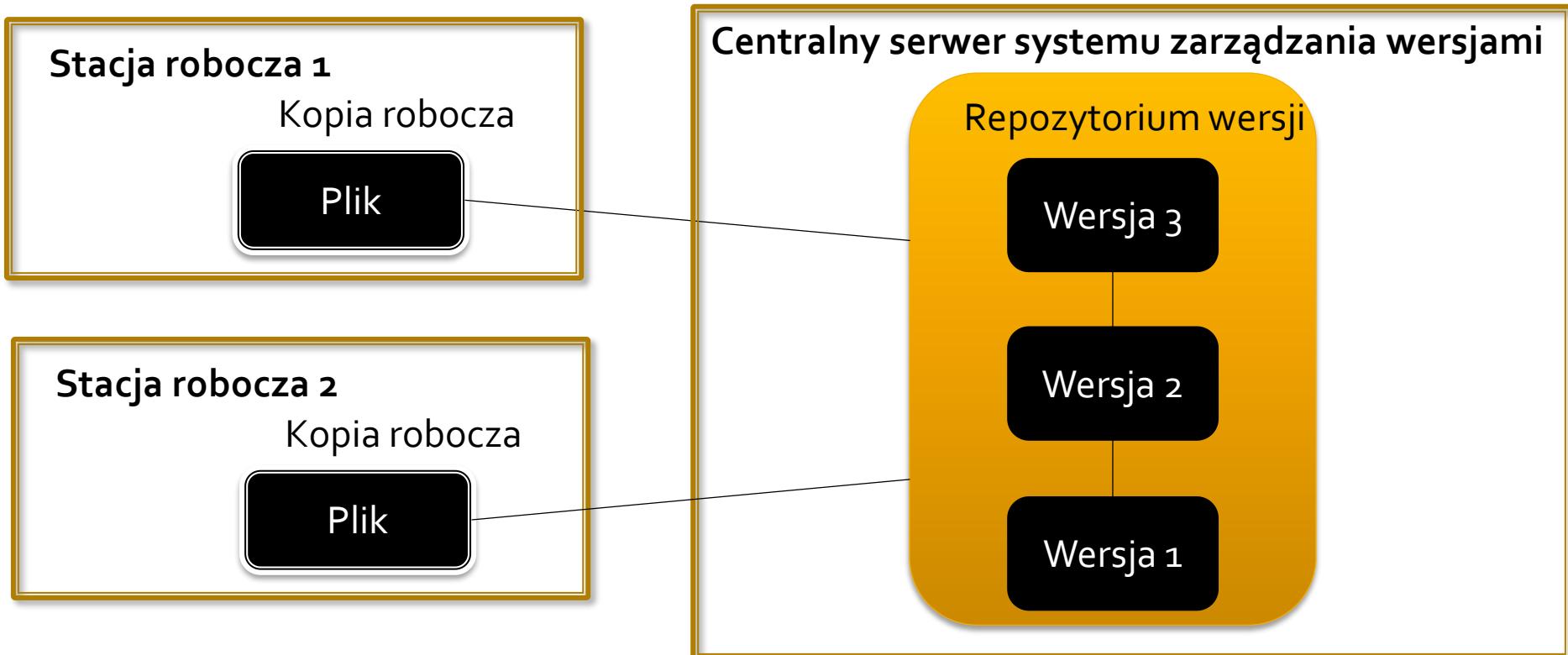
Systemy kontroli wersji (1 z 3)

- **Lokalne** – lokalna baza danych wersji przechowująca wersje plików dodanych do kontroli wersji (np. rcs)

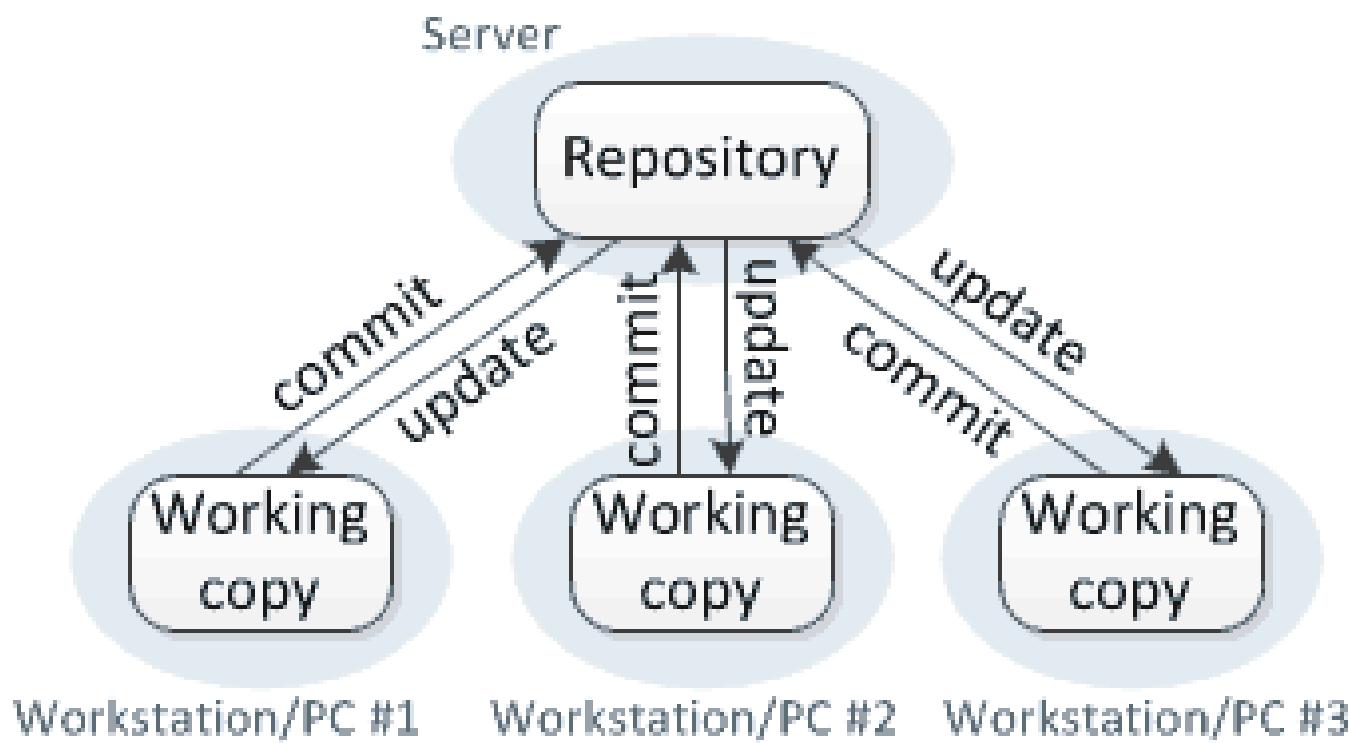


Systemy kontroli wersji (2 z 3)

- **Scentralizowane** – pojedynczy serwer zarządzający repozytorium wersji. Dowolna liczba klientów pobierających i zatwierdzających wersje (np. CVS, Subversion, Perforce).

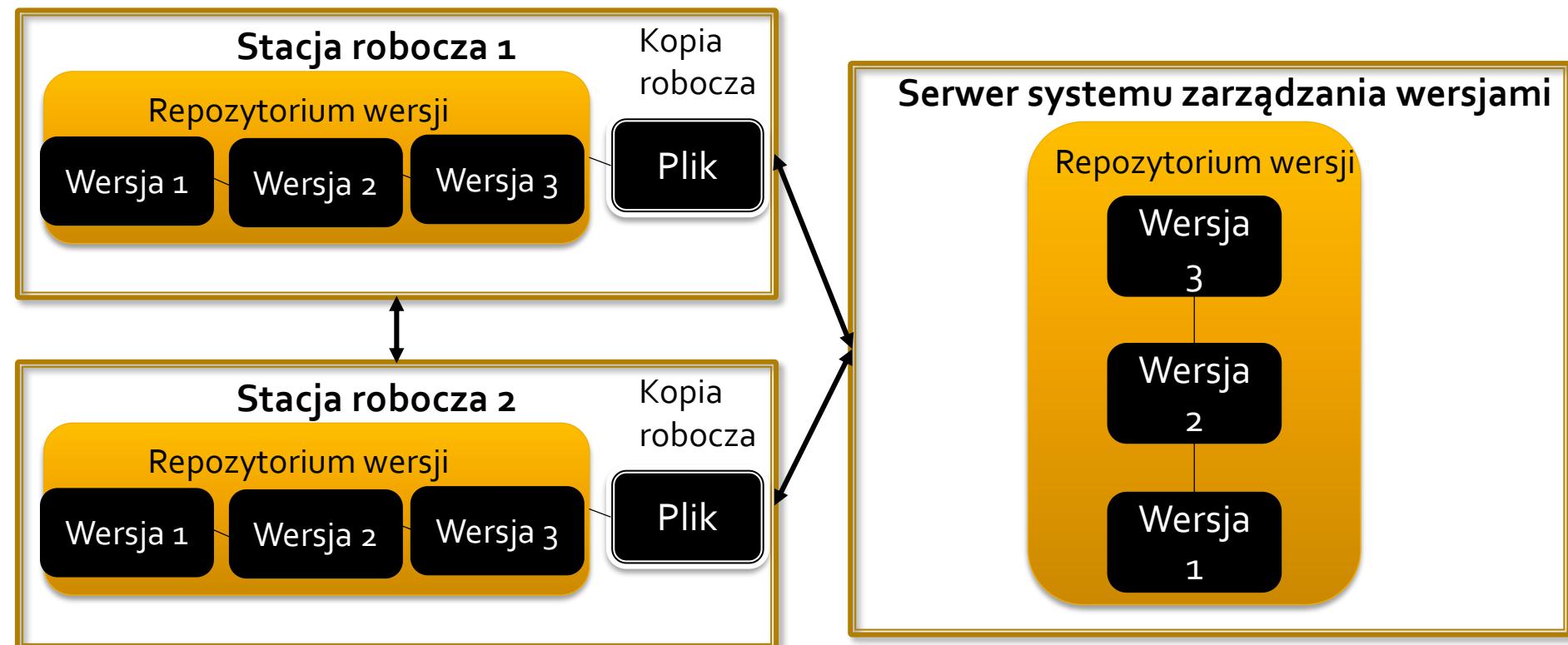


Centralized version control

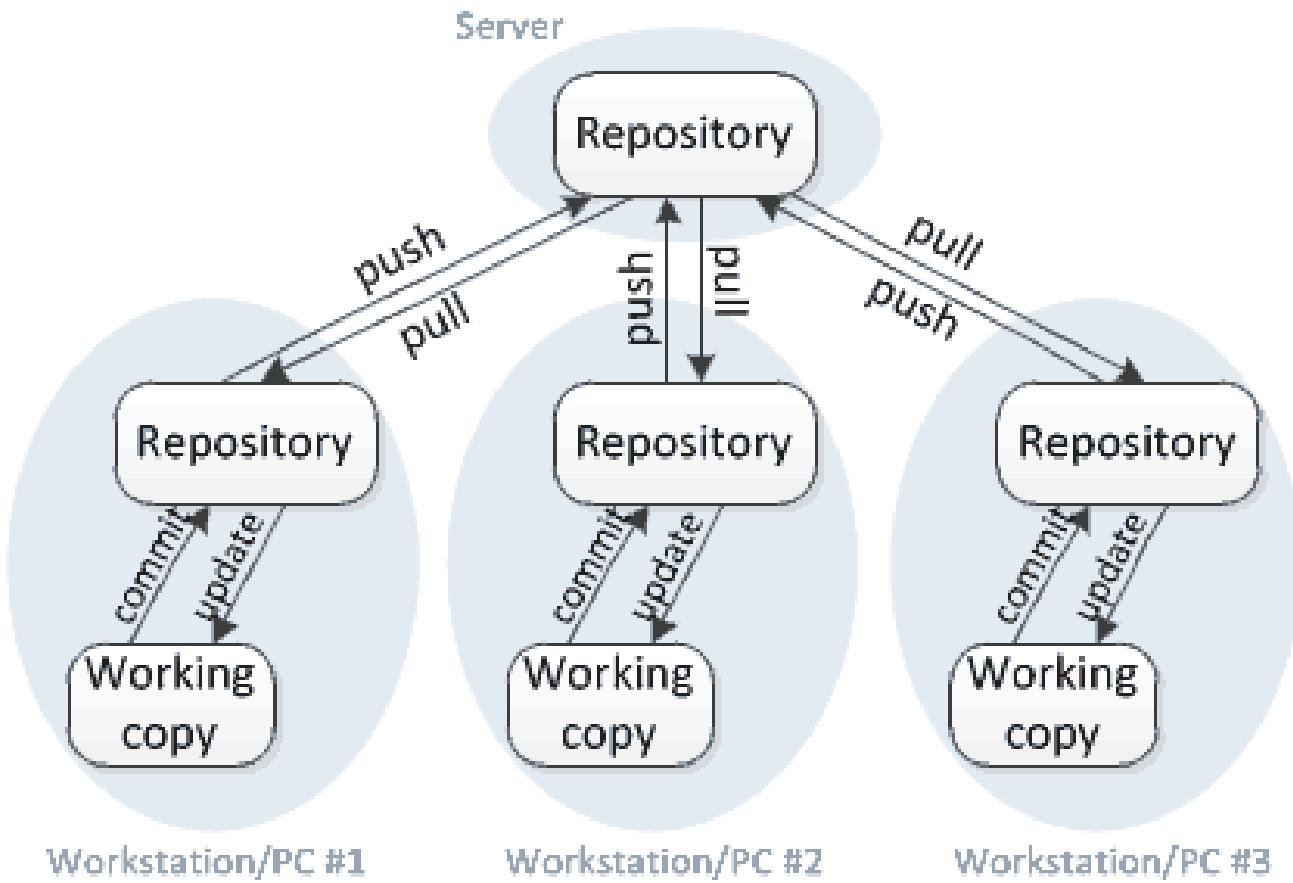


Systemy kontroli wersji (3 z 3)

- **Rozproszone** – klient posiada pełną kopię repozytorium, możliwa jest współpraca w ramach więcej niż jednego repozytorium (np. Git, Mercurial, Bazaar, Darcs)

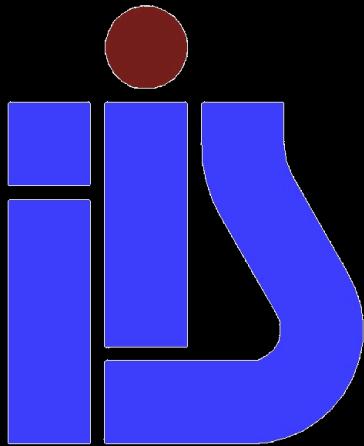


Distributed version control



Podsumowanie – część 1

- Zarządzanie konfiguracją oprogramowania (CM) to zarządzanie systemem będącym w ewolucji. W kontekście konserwacji oprogramowania rolą CM jest zapewnianie, że zmiany wprowadzane do systemu są kontrolowane a szczegóły ich dotyczące są zapamiętane.
- Podstawowymi procesami CM są zarządzanie zmianą, zarządzanie wersjami budowanie systemu oraz zarządzanie wydaniami.
- Zarządzanie zmianą uwzględnia również oszacowanie propozycji zmiany w systemie pod kątem opłacalności jej implementacji w nowej wersji systemu.
- Zarządzanie wersjami dotyczy śledzenia różnych wersji komponentów oprogramowania powstających w momencie wprowadzania do nich zmian.



Zarządzanie konfiguracją oprogramowania

Configuration management

część 2

Automatyzacja procesu budowy systemu

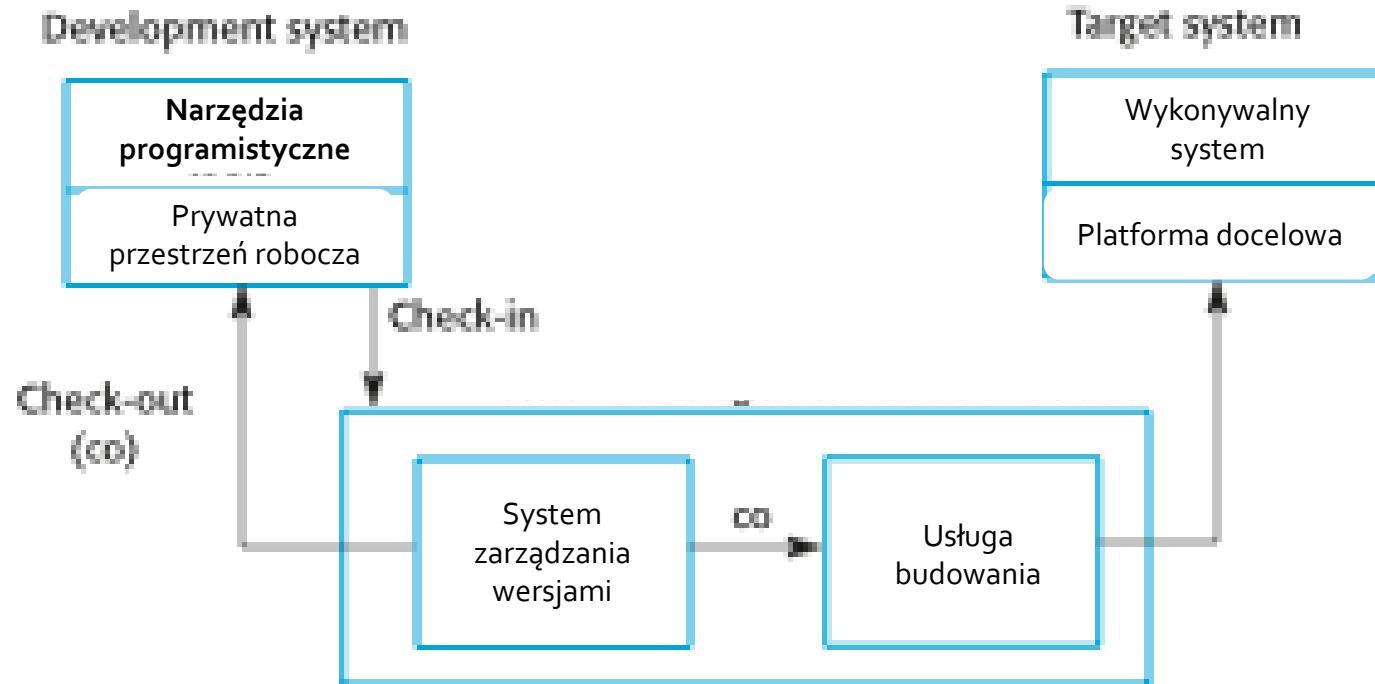
Budowanie systemu

- Proces tworzenia kompletnego, wykonywalnego systemu poprzez komplikację i połączenie komponentów systemu, bibliotek, plików konfiguracyjnych, itp.
- Narzędzia służące do budowania systemu oraz narzędzia służące zarządzaniu wersjami muszą komunikować się ze sobą ponieważ proces budowy wymaga pobrania (check-out) wersji komponentów z repozytorium zarządzanego przez system zarządzania wersjami.
- Opis konfiguracji wykorzystywany do identyfikacji linii bazowej jest również wykorzystywany przez narzędzia budowania systemu.

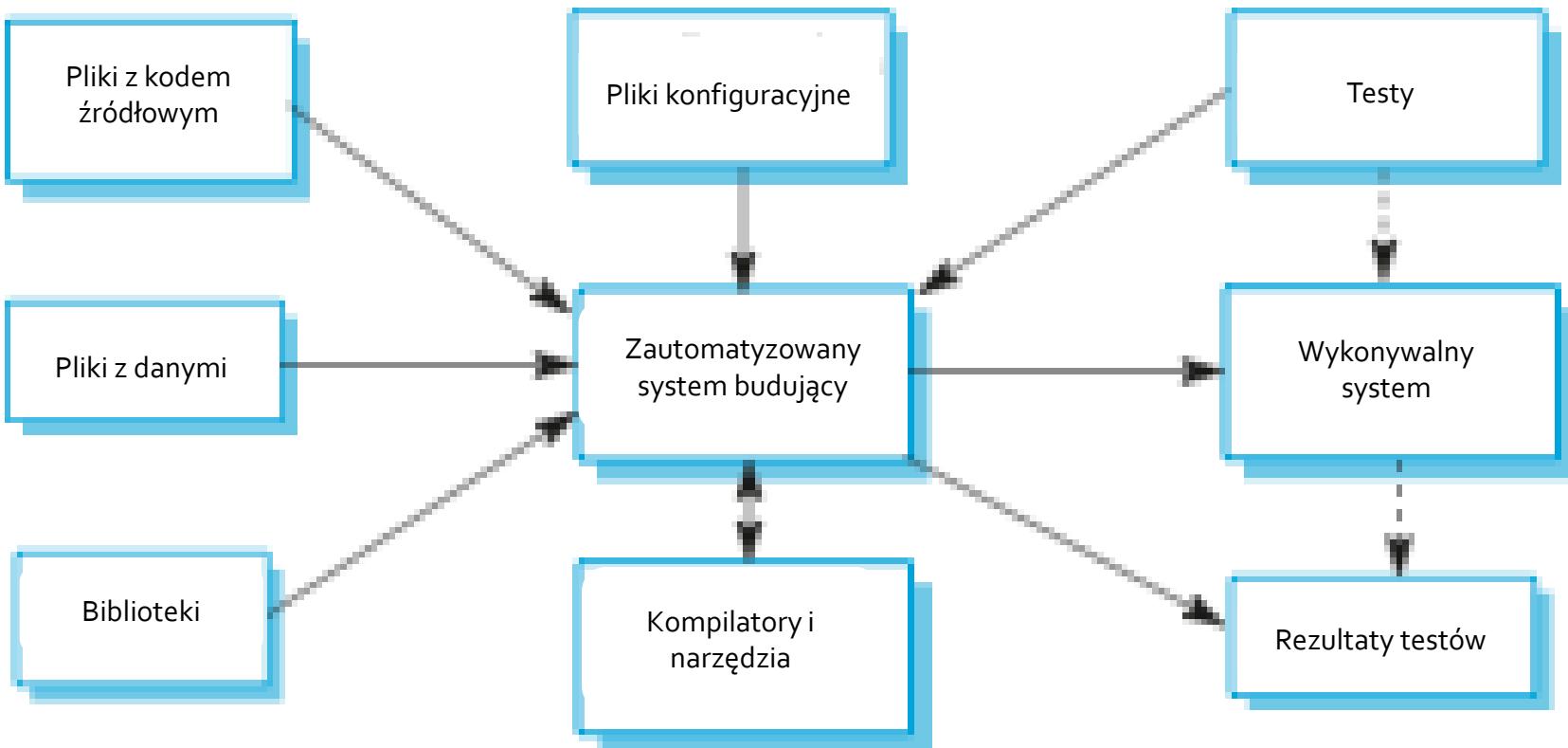
Elementy platformy budowania

1. System służący rozwojowi oprogramowania, w którego skład wchodzą narzędzia takie jak kompilatory, edytory kodu źródłowego, itd.
 - Programiści pobierają kod źródłowy z repozytorium wersji do prywatnej przestrzeni roboczej przed dokonaniem zmian w systemie.
2. Usługa budowania (ang. build server), wykorzystywana do budowania ostatecznej, wykonywalnej wersji systemu.
 - Programiści pobierają kod źródłowy z repozytorium wersji przed uruchomieniem procesu budowy. Budowa systemu może wymagać zewnętrznych bibliotek, które nie są przechowywane w systemie zarządzania wersjami.
3. Środowisko docelowe, reprezentujące platformę na które system będzie wykonywany.

Rozwój, budowa i platforma docelowa



Budowanie systemu



Funkcjonalności systemu budującego

- 
1. Generowanie skryptów budujących
 2. Integracja z systemem zarządzania wersjami
 3. Minimalizacja potrzeby rekompilacji komponentów
 4. Tworzenie wykonywalnego systemu
 5. Automatyzacja testów
 6. Raportowanie
 7. Generacja dokumentacji

Build script



```
<project name="MyProject" default="dist" basedir=".">
  <description>
    simple example build file
  </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist"  location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
         description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
         description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean"
         description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

Zarządzanie zależnościami

```
<groupId>edu.iis.paw</groupId>
<artifactId>wicket-lab2-todowebapp</artifactId>
<packaging>war</packaging>
<version>1.0-SNAPSHOT</version>
<name>quickstart</name>

<dependencies>
    <dependency>
        <groupId>org.apache.wicket</groupId>
        <artifactId>wicket-core</artifactId>
        <version>${wicket.version}</version>
    </dependency>

    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.6.4</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.16</version>
    </dependency>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.10</version>
    </dependency>
```

Minimalizacja rekompilacji

- Narzędzia są projektowane w taki sposób aby rekompilacja komponentów była wykonywana tylko gdy jest na prawdę potrzebna.
- System sprawdza czy dostępna jest skompilowana wersja komponentu. Jeżeli tak – komplikacja nie jest wymagana.
- Każdy kod źródłowy i kod skompilowany posiada unikatową sygnaturę. Sygnatura jest zmieniana gdy kod źródłowy zostaje poddany edycji.
- Porównanie sygnatur pozwala na podjęcie decyzji o potrzebie ponownej generacji kodu wynikowego dla komponentu.
 - Znaczniki czasowe
 - Sygnaturą kodu źródłowego jest czas i data modyfikacji.
 - Suma kontrolna kodu źródłowego
 - Sygnaturą pliku zawierającego kod źródłowy jest suma kontrolna wyliczona z danych zawartych w pliku.

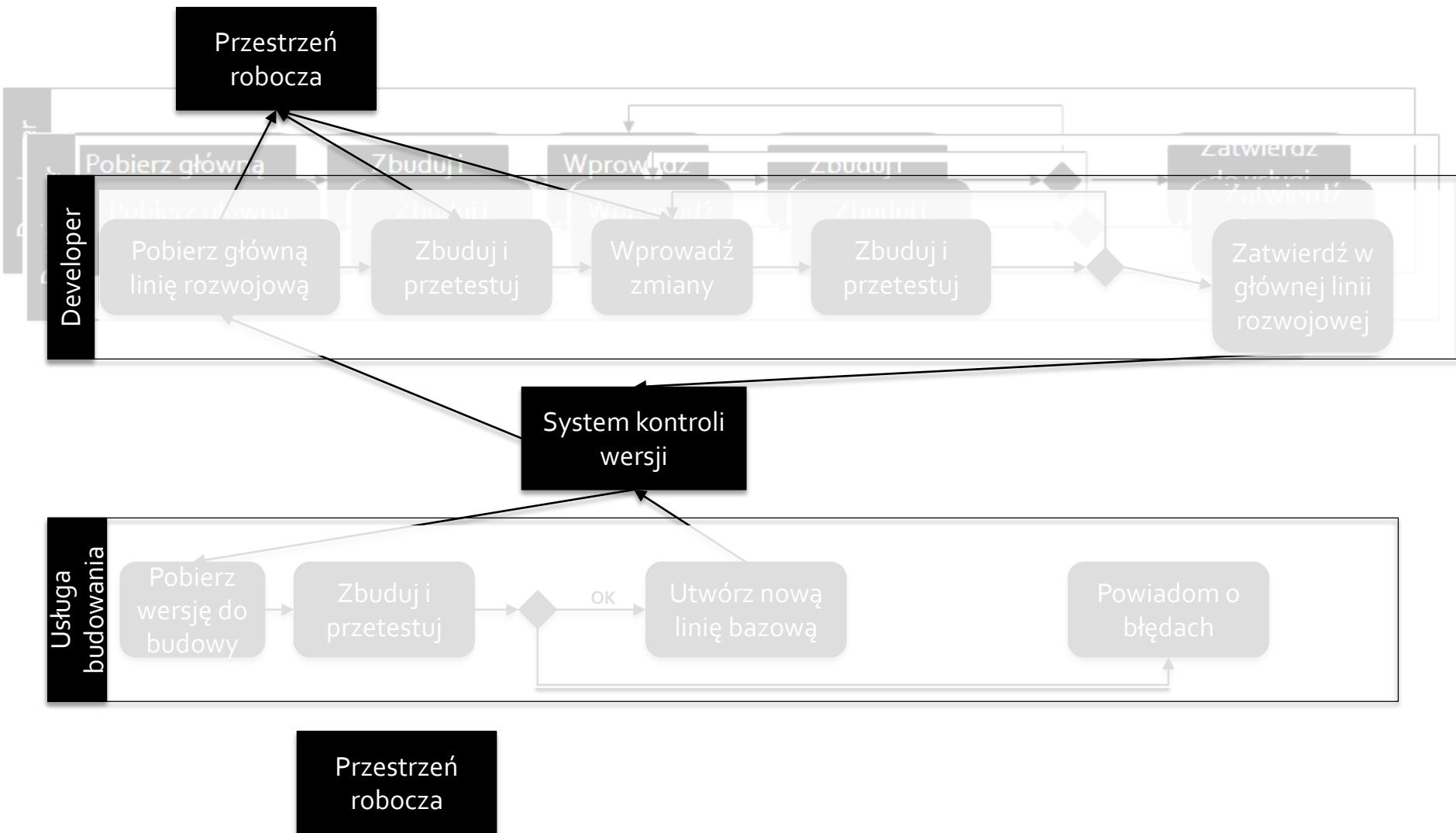
Zwinne budowanie - 1

- Pobieranie głównej linii systemu z systemu zarządzania wersjami do prywatnej przestrzeni roboczej programisty.
- Budowanie systemu i automatyczne uruchamianie testów w celu zapewnienia, że system przejdzie testy poprawnie. W przypadku błędów – rezultat budowy jest błędny – osoba, która wprowadziła do repozytorium błędą wersję powinna zostać poinformowana.
- Dokonanie zmian w komponentach systemu.
- Zbudowanie systemu w ramach przestrzeni prywatnej, uruchomienie testów. Jeżeli testy zakończą się błędami – kontynuacja edycji.

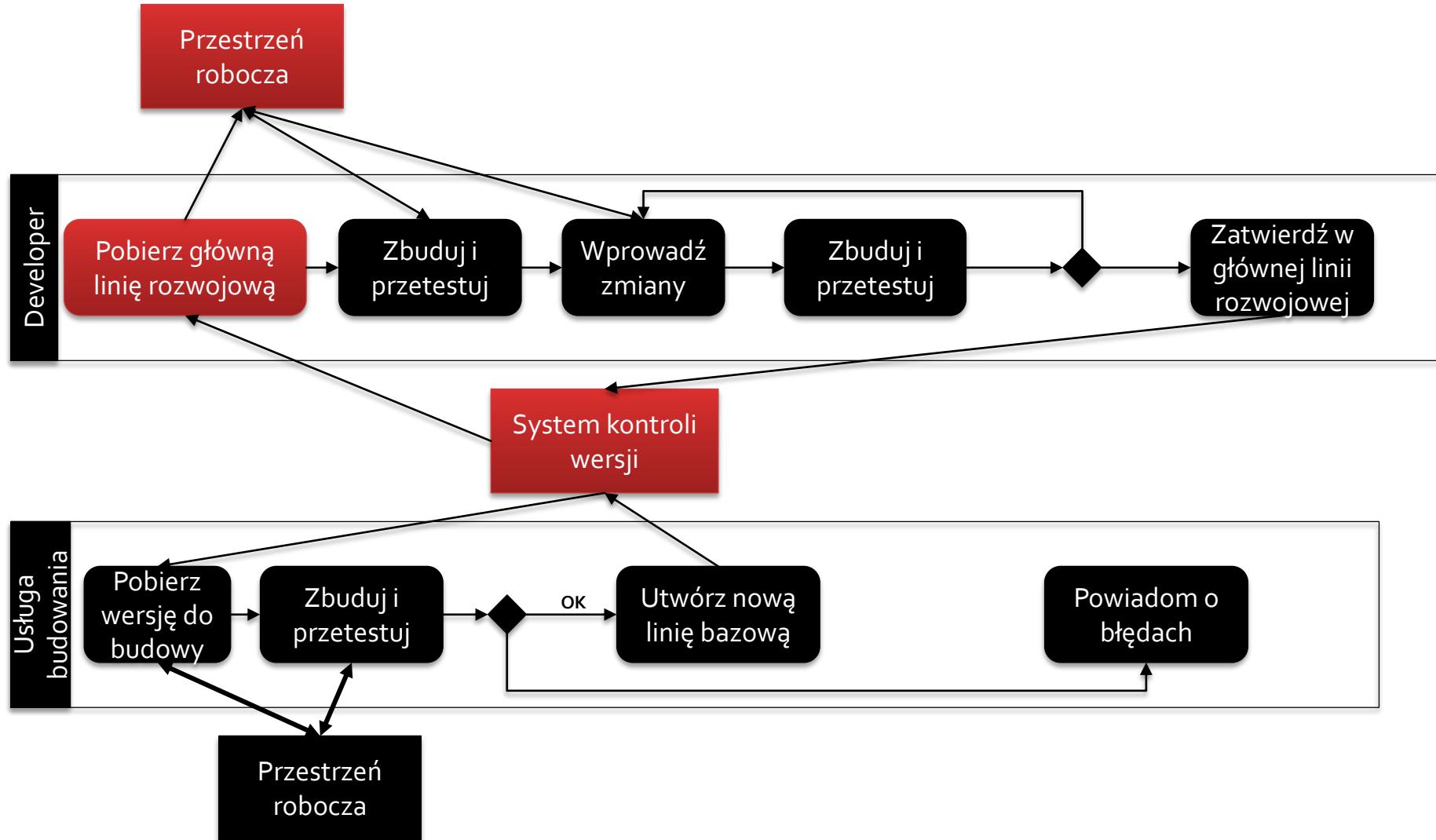
Zwinne budowanie - 2

- Jeżeli system przeszedł testy poprawnie, przekazanie systemu do usługi budującej (bez zatwierdzania zmian w repozytorium zarządzania wersjami).
 - Zbudowanie systemu i uruchomienie testów w przestrzeni usługi budowania.
 - Istnieje możliwość, że w międzyczasie inne komponenty systemu zostały zmodyfikowane. Jeżeli tak się stało i pojawiły się błędy, zmienione komponenty muszą być pobrane z repozytorium – poddane edycji do momentu poprawnego wykonania testów w przestrzeni prywatnej.
- Jeżeli system przejdzie testy w przestrzeni usługi budującej – zmiany mogą zostać zatwierdzone do systemu zarządzania wersjami w celu utworzenia nowej linii bazowej w głównej linii systemu.

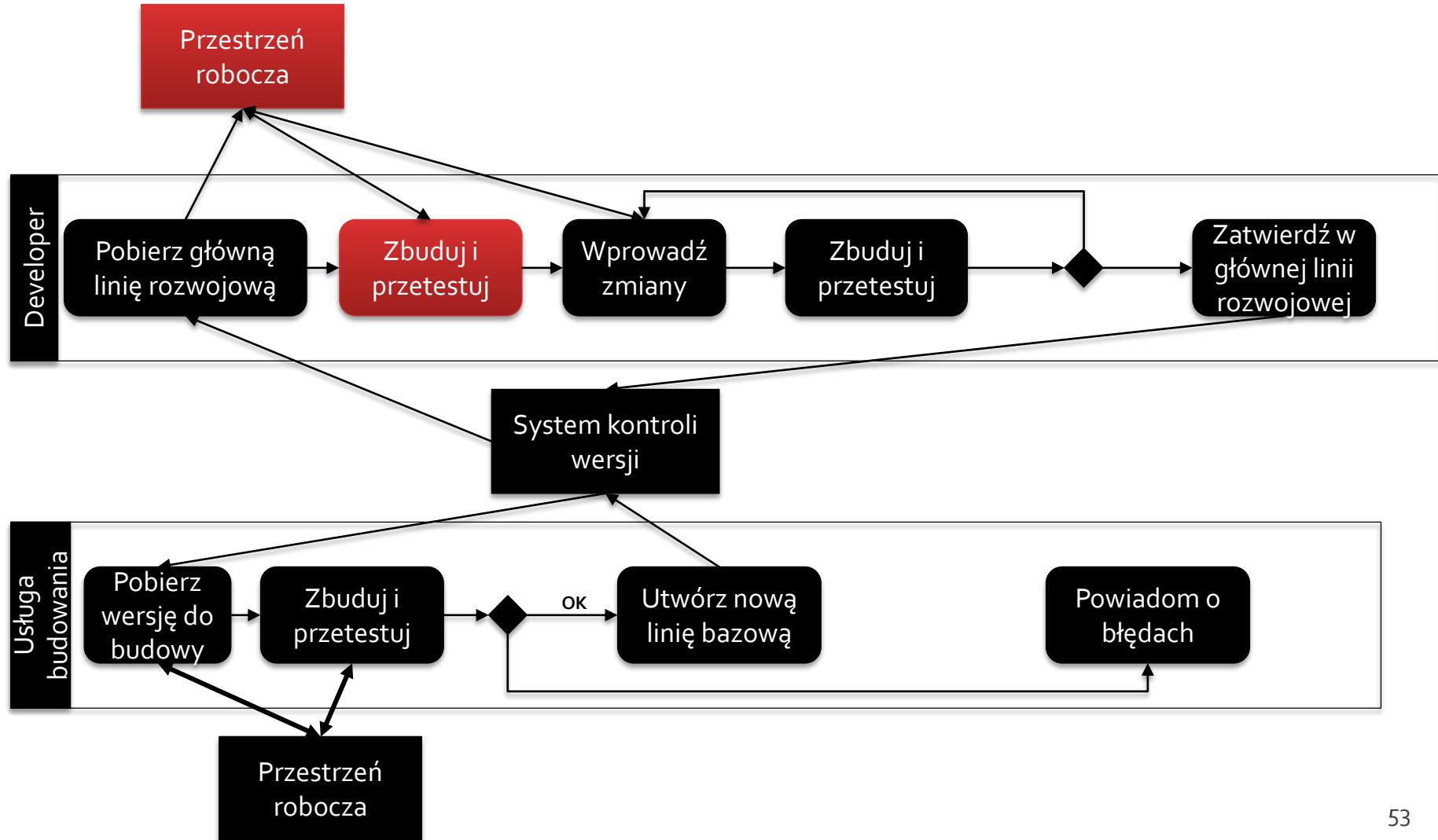
Ciągła integracja (ang. continuous integration)



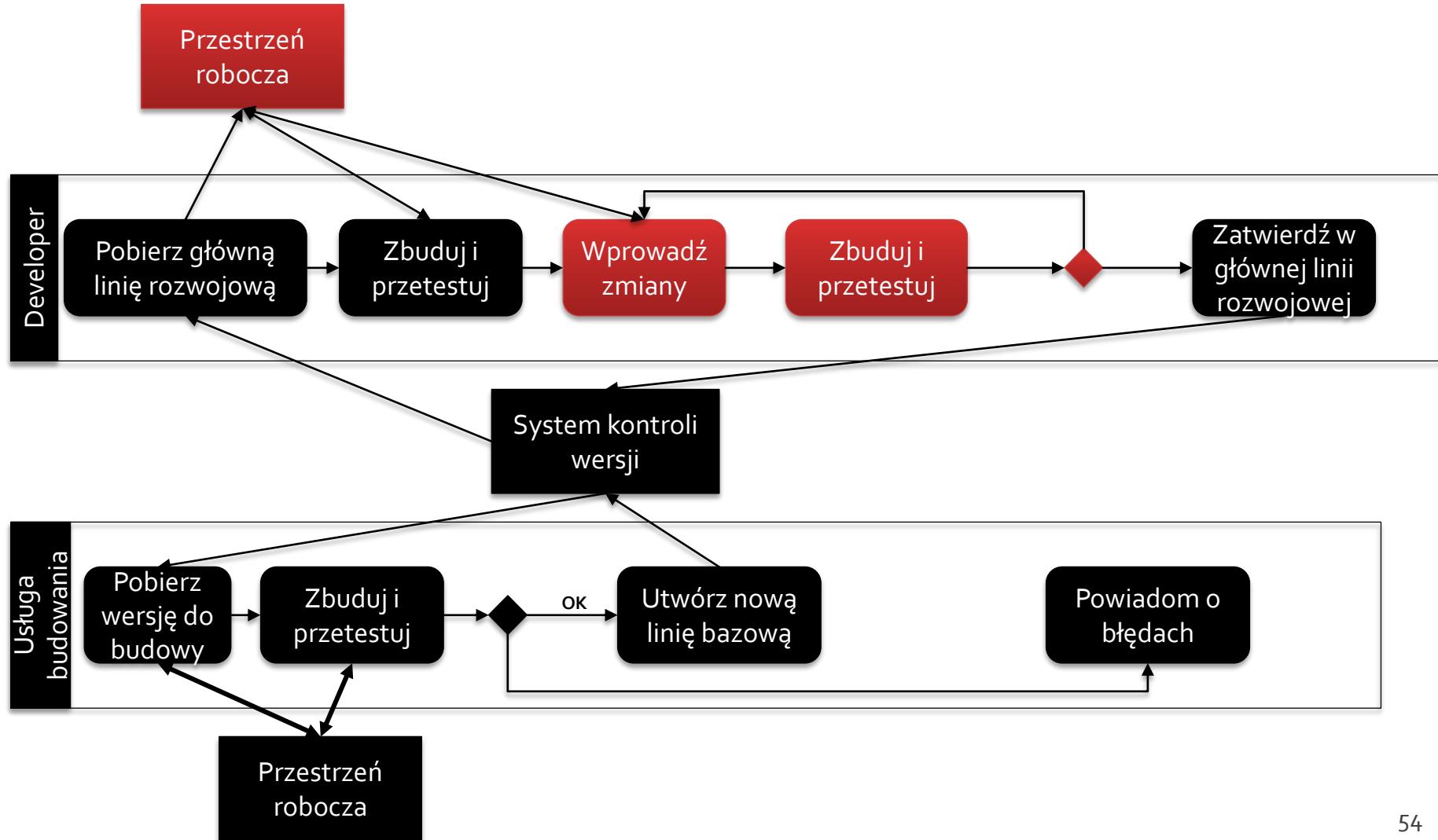
Ciągła integracja (1)



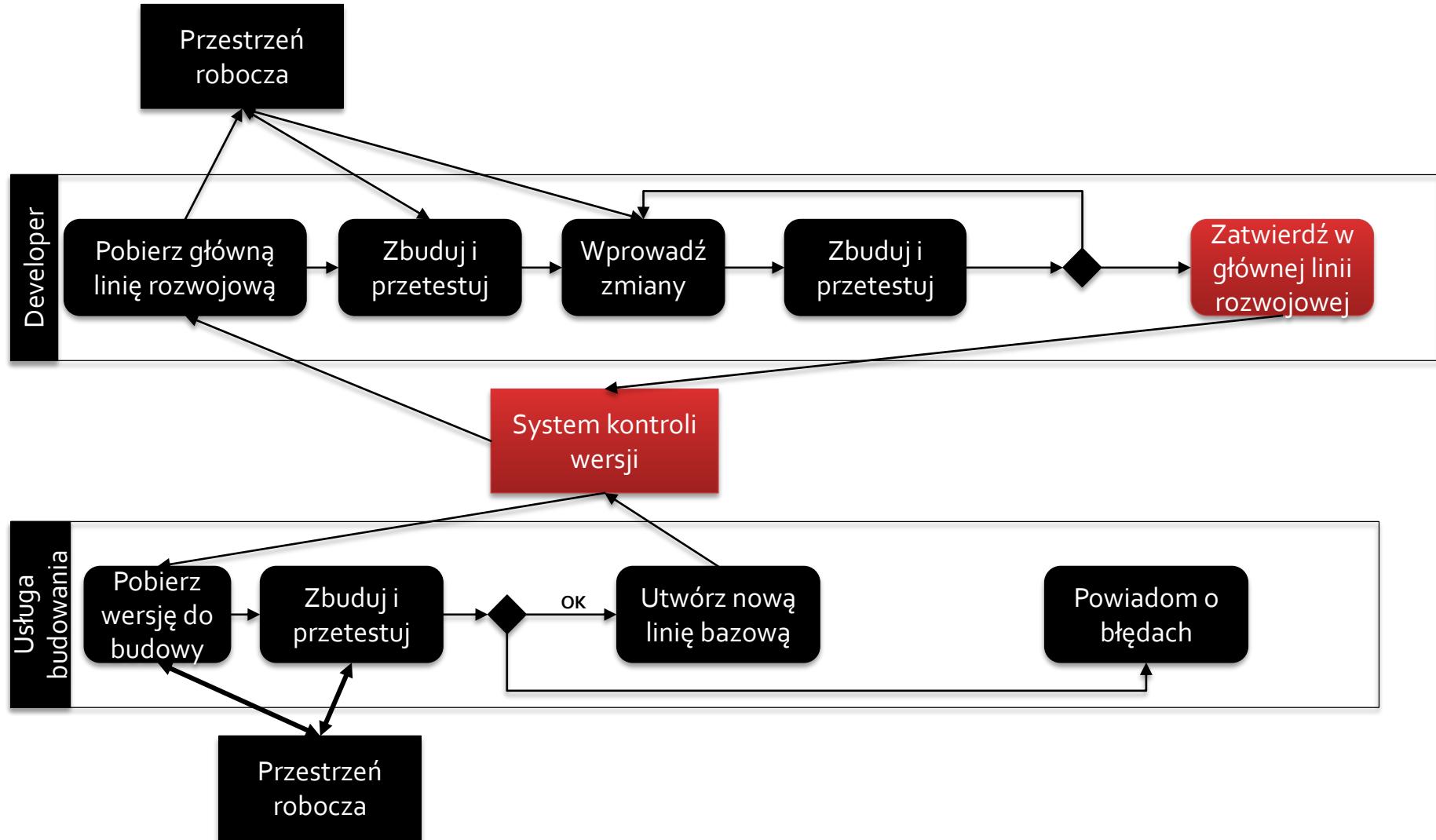
Ciągła integracja (2)



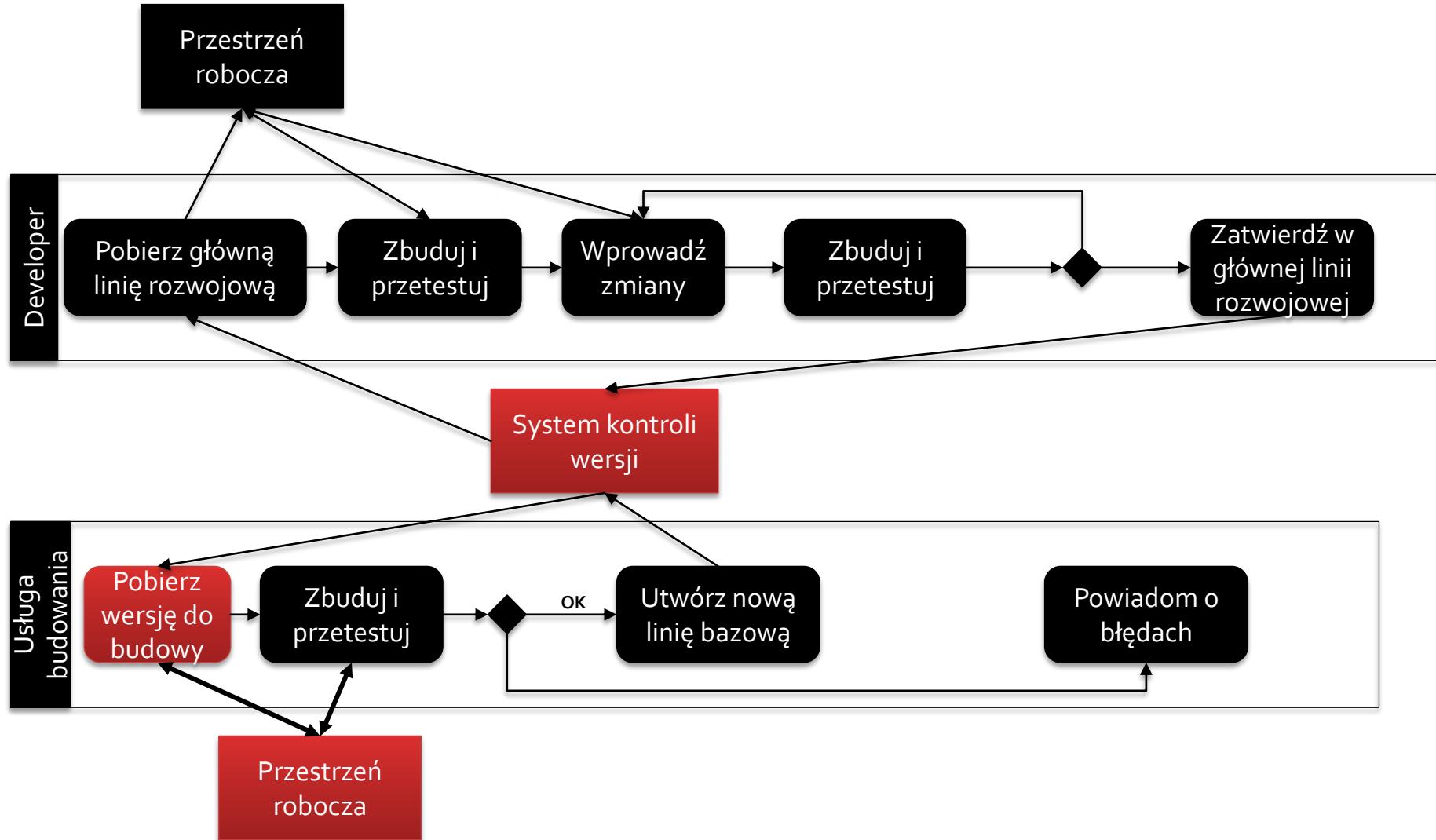
Ciągła integracja (3)



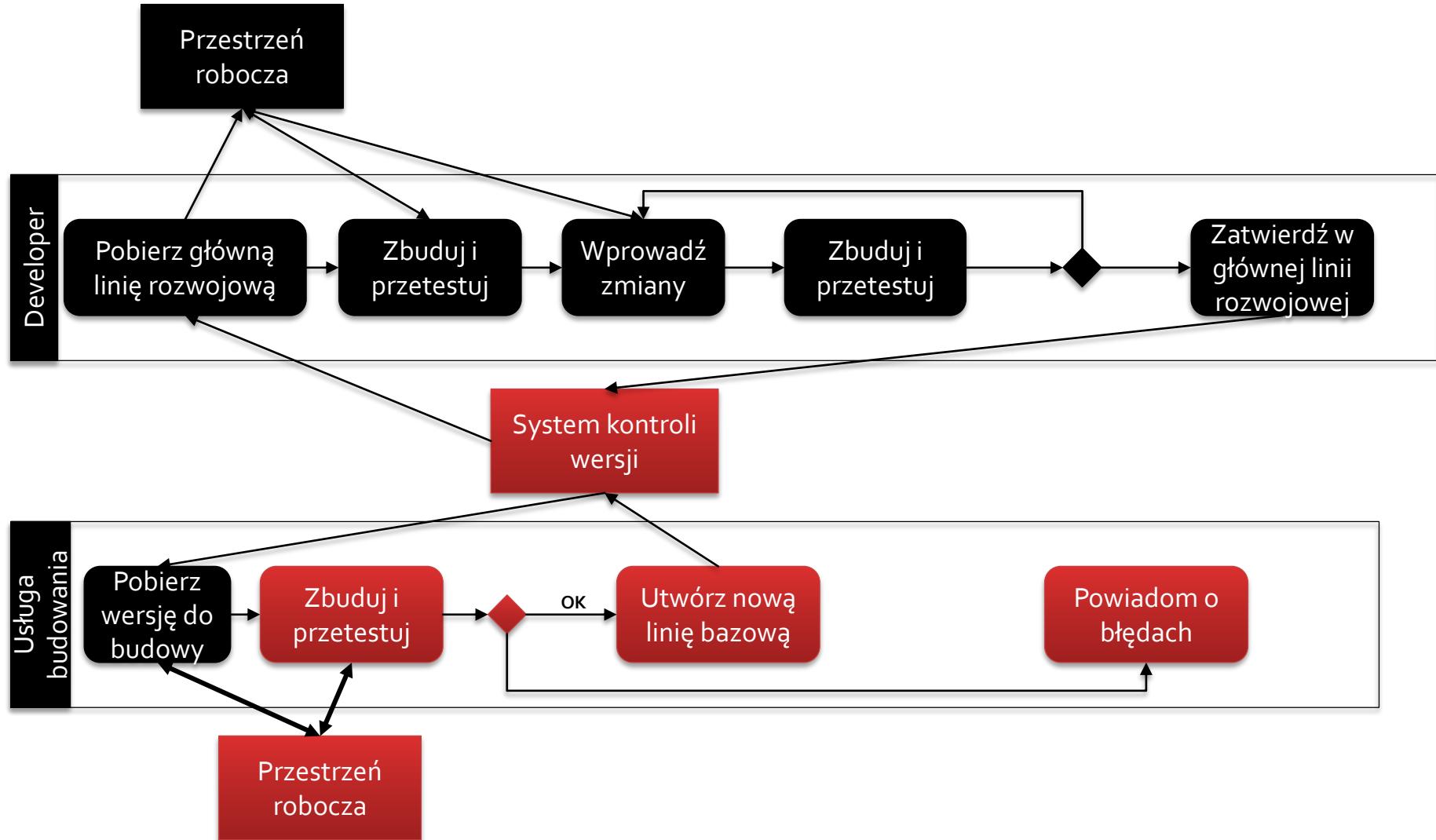
Ciągła integracja (4)



Ciągła integracja (5)



Ciągła integracja (6)



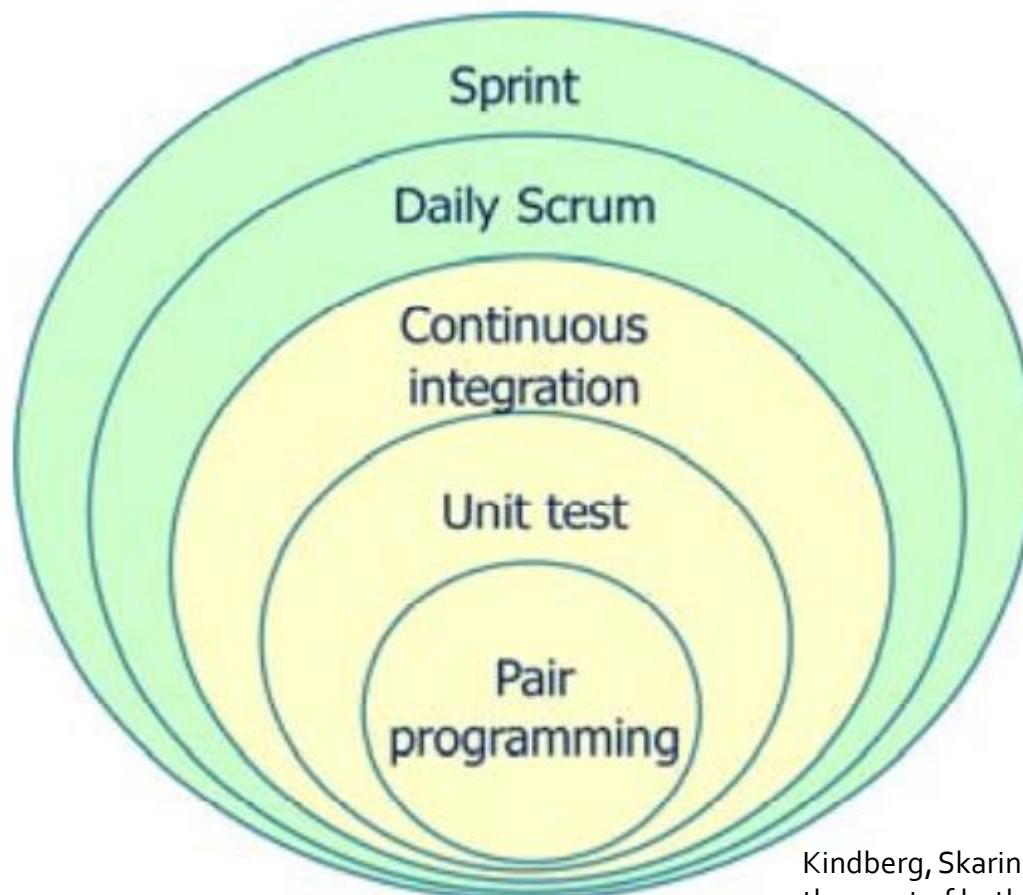
Zalety ciągłej integracji



- Zapewnienie, że zespół pracuje **wspólnie**
-
- Szybka informacja zwrotna (ang. feedback loop)
- Wymusza poprawność komplikacji i wykonanie testów

Pętle zwrotnie w wytwarzaniu zwinnym

- Na przykładzie Scrum oraz praktyk inżynierskich



Kindberg, Skarin: Kanban and Scrum making
the most of both, 2009

Ograniczenia ciągłej integracji

- Zespoły odpowiedzialne za przygotowanie środowiska uruchomieniowego czekają na poprawki w dokumentacji
- Testerzy czekają na „dobre buildy”
- Zespół rozwijający system otrzymuje raporty z błędami tydzień po tym jak rozpoczął pracę nad nowymi funkcjonalnościami
 - (długą pętlą zwrotną pomiędzy zespołami: (development i operational))
- Brak zapewnienia, że architektura aplikacji pozwala spełnić wymagania niefunkcjonalne

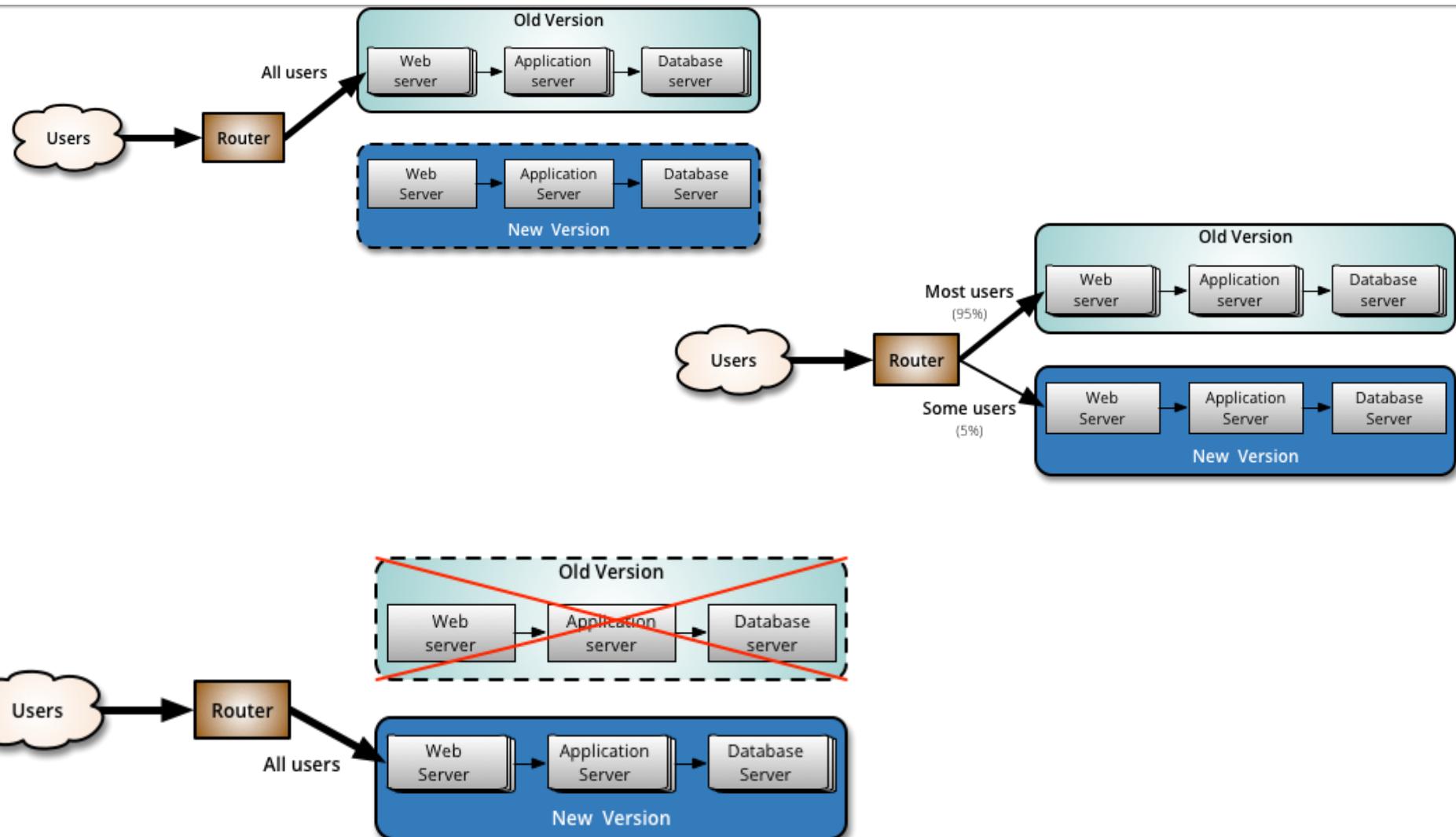
Ciągłe wdrażanie (ang. continuous deployment)

- Ciągła integracja – automatyzacja procesu rozwoju
- Ciągłe wdrażanie - automatyzacja całości procesu inżynierii oprogramowania
 - W dowolnym momencie wdrożenie wybranej wersji zbudowanego oprogramowania (za pomocą „przycisku”)
 - Automatyczne umieszczenie w środowisku
 - testowym, zapewniania jakości, produkcyjnym, ...

Ciągłe dostarczanie (ang. continuous delivery)

- Mechanizmy pozwalające na automatyczne dostarczenie wersji oprogramowania do klienta jeżeli tylko przejdzie ono wszystkie etapy zatwierdzania jakości.
 - Potencjalnie dowolna wersja systemu może być dostarczona klientowi
 - Promuje całkowitą automatyzację -> rurociąg dostarczający oprogramowanie (ang. deployment pipeline)

Canary release



Codzienne „build'y”



- Organizacja rozwijająca oprogramowanie ustala czas dostarczenia komponentów systemu (np. na godzinę 15:00)
 - Jeżeli programiści posiadają nowe wersje komponentów, które rozwijają, muszą je dostarczyć do wyznaczonej godziny.
- Nowa, kompletna wersja systemu jest budowana z tych komponentów (kompilacja, łączenie).
- Zbudowany system jest przekazywany zespołowi testującemu, który wykonuje predefiniowane testy systemowe.
- Błędy wykryte podczas testowania są dokumentowane i zwracane do programistów. Zostaną one poprawione w kolejnych wersjach komponentów.

Rozwój systemów automatyzacji procesu budowania

- linia komend
- skrypty
- dedykowane języki skryptowe (GNU Make) – początek automatyzacji
- automatyzacja czynności przed- w trakcie oraz po- budowie (wywoływanie wielu skryptów).
- Minimalizacja samodzielnego pisania skryptów
- Budowa typu - Continuous Integration – budowa przyrostowa (ang. *incremental build*) cechująca się bardzo częstymi odwołaniami do procesu komplikacji.
- rozproszenie budowy (przyspieszenie budowy oprogramowania poprzez rozdzielenie procesu komplikacji na wiele lokalizacji)
- rozproszenie przetwarzania – każdy krok procesu budowy może być wykonany na innej maszynie.
- Automatyczne wykrywanie zależności

Przykłady narzędzi automatyzacji procesu budowania



- Apache Ant (Java), Nant (.NET) – automatyzacja oparta o pliki skryptowe.
- Apache Maven (Java), Gradle (Groovy)
- Scons (Python)
- Phing (PHP)
- Rake (Ruby)
- Gulp, Grunt (JavaScript)

Narzędzia wspierające model ciągłej integracji

- Rozbudowana integracja z innymi narzędziami
 - Systemy zarządzania wersjami, systemu automatyzacji procesu budowania, IDE
- Przykłady
 - Jenkins
 - IBM uDeploy
 - Cruise Control
 - Apache Continuum
 - TeamCity
- CI jako usługa (w chmurze)
 - TravisCI, Cloudbees (Jenkins), Codebetter (TeamCity)

Zarządzanie wydaniami

Zarządzanie wydaniami

- **Wydanie** (ang. release) to dystrybucja (publiczna lub prywatna) wersji oprogramowania.
- Wydanie publiczne systemu jest to wersja oprogramowania, która została dostarczona klientom.
- W przypadku oprogramowania generycznego istnieją zazwyczaj dwa typy wydań:
 - Wydania główne (ang. major releases) które dostarczają znaczących zmian w funkcjonalności systemu
 - Wydania podzielone (ang. minor releases), których celem jest naprawa błędów oraz rozwiązywanie problemów zgłaszanych przez klientów.
- W przypadku oprogramowania dedykowanego lub linii produktowych wydania systemu mogą być produkowane niezależnie dla poszczególnych klientów oraz w ramach konkretnego klienta może istnieć (i być w użyciu) kilka wersji jednocześnie.

Śledzenie wydań

- W przypadku problemów potrzebne może być utworzenie dokładnie takiego systemu, jaki dostarczony został do konkretnego klienta.
- Kiedy tworzone jest wydanie systemu, musi ono zostać udokumentowane w celu zapewnienia reprodukcji w przyszłości.
- Jest to szczególnie istotne w przypadku systemów dedykowanych i wbudowanych o długim czasie życia, które sterują złożonymi procesami/maszynami.
 - Klienci mogą korzystać z konkretnego wydania systemu przez wiele lat i zgłosić problem długo po dacie wydania.

Reprodukcja wydania

- W celu udokumentowania wydania należy zapisać określone wersje kodu źródłowego komponentów, które zostały wykorzystane przy tworzeniu wydania.
- Należy zachowywać kopie plików z kodem źródłowym, odpowiednich plików wykonywalnych oraz wszystkie dane i pliki konfiguracyjne.
- Zapisana powinna być również wersja systemu operacyjnego, bibliotek, kompilatorów oraz innych narzędzi wykorzystanych do budowania systemu.

Planowanie wydania

- Wydanie to nie tylko strona techniczna (praca włożona w przygotowanie dystrybucji wydania),
 - przygotowanie reklam, materiałów, strategii marketingowej w celu przekonania klientów do nowej wersji.
- Harmonogramowanie wydania
 - Jeżeli wydania są zbyt częste lub wymagają aktualizacji sprzętu klienci mogą nie wyrażać zainteresowania zmianą (szczególnie jeżeli muszą za to zapłacić).
 - Jeżeli wydania są zbyt rzadkie – klienci mogą wybierać produkt alternatywny.

Komponenty wydania

- Oprócz kodu wykonywalnego na wydanie mogą składać się:
 - Pliki konfiguracyjne definiujące sposób w jaki wydanie powinno być skonfigurowane dla konkretnej instalacji,
 - Pliki z danymi (np. lokalizowane komunikaty o błędach),
 - Program instalacyjny wspomagający instalacje oprogramowania na danym systemie i sprzęcie,
 - Dokumentacja (elektroniczna, papierowa) opisująca system,
 - Pakiety i materiały promocyjne dedykowane do wydania.

Czynniki wpływające na planowanie wydań - 1



Czynnik	Opis
Jakość techniczna systemu	Jeżeli raportowane są poważne błędy, które mogą mieć wpływ na pracę wielu użytkowników systemu – wydanie poprawiające błędy może być potrzebne. Mniejsze błędy systemu mogą być wydawane w postaci „łatek” (ang. patch), które mogą być zastosowane do bieżącego wydania.
Zmiany platformy	Mожет существовать необходимость создания нового выпуска из-за появления новой версии операционной системы.
Piąte prawo Lehman'a	To „prawo” sugeruje, że jeżeli doda się dużo nowych funkcjonalności do systemu to razem z nimi wprowadzi się również błędy, które ograniczą ilość funkcjonalności w kolejnym wydaniu. Dlatego wydanie systemu, dodające dużą liczbę istotnych nowych funkcjonalności może zazwyczaj wymusić pojawienie się wydania (wydań), związanych z poprawą błędów czy zwiększeniem wydajności.

Czynniki wpływające na planowanie wydań - 2



Czynnik	Opis
Konkurencja	Konkurencyjne produkty wprowadzają nowe cechy – jeżeli nie chcemy stracić rynku musimy wprowadzić podobne cechy do systemu.
Wymagania rynku	Dział marketingu w organizacji zobowiązał się do wydania będzie dostępna w danym dniu.
Zmiany proponowane przez klientów	W przypadku systemów dopasowanych klienci mogą zgłosić potrzebę realizacji zestawu zmian i oczekwać wydania systemu jak tylko zmiany zostaną wprowadzone.

Cykl wydania

(ang. software release lifecycle)

- Wydanie publiczne danej wersji oprogramowania jest często związane z całym procesem nazywanym cyklem wydania
- Fazy cyklu wydania reprezentują stabilność porcji oprogramowania polegającej wydaniu oraz zakres postępu wykonanych do danej chwili prac rozwojowych

Przykład faz cyklu wydania



- Rozwój
 - Pre-alpha
 - alpha
 - beta
 - release candidate (RC)
- Wydanie
 - RTM (ang. "release to manufacturing" lub "release to marketing")
 - GA (ang. general availability)
 - Service release / service pack

Wersjonowanie w cyklu wydania

- Proces przypisania unikatowej nazwy (identyfikatora – numeru) do „stanu” oprogramowania
 - Konwencja mniej i bardziej znaczące numery
- Schematy wersjonowania
 - Identyfikatory bazujące na sekwencji
 - Każdemu wydaniu oprogramowania przypisany jest identyfikator składający się z kilku sekwencji znaków alfanumerycznych lub liczb.
 - 1.0, 2.3.34.1546, 9.3b, 3.4.1
 - Nie ma standardu „znaczenia” sekwencji

Wersjonowanie a znaczenie zmiany



- W niektórych schematach wersjonowania znaczenie zmiany odzwierciedlane jest przez poziom sekwencji, np:
 - major.minor[.build[.revision]]
- Schemat wersjonowania może również przenosić informacje o zakresie wykonanych testów rzeczywistych
 - 1.1b1, 1.1rc2, 1.1
 - 0.1, 0.3

Inne schematy wersjonowania

- bazujące na:
 - dacie (np. Ubuntu – 8.04, 9.10)
 - roku edycji (np. Office 2003)
 - kodach alfanumerycznych (np. Adobe Photoshop CS2)
 - ekscentryczne (np. tex)
 - niezdecydowane (np. MS Windows)
 - hybrydowe (czyt. marketingowe) (np. Java 1.5 = Java 5)

Wersjonowanie semantyczne

- <http://semver.org/>
- X.Y.Z
 - X – major, Y – minor, Z – patch
- Jakakolwiek zmiana w opublikowanym, w danej wersji oprogramowaniu wymaga zmiany wersji.
- X = zero – inicjacja projektu – wszystko może ulec zmianie
- X++ - wprowadzone niekompatybilne zmiany
- Y++ - wprowadzone zmiany kompatybilne wstecz
- Z++ - wprowadzono poprawkę (bez zmian w funkcjonalności)
- Możliwe jest dołączanie identyfikatorów
 - 1.0.0-alpha, 1.0.0-beta+exp.sha.5114f85

Podsumowanie – część 2

- Budowanie systemu jest procesem łączenia komponentów systemu do postaci wykonywalnego programu, który może być uruchomiony na docelowym systemie komputerowym.
- Oprogramowanie powinno być często budowane i testowane natychmiast po utworzeniu jego nowej wersji. Ułatwia to wykrywanie błędów i problemów, które pojawiły się od czasu ostatniego zbudowania systemu.
- Wydanie systemu składa się z wykonywalnego programu, plików konfiguracyjnych i dokumentacji. Zarządzanie wydaniami to podejmowanie decyzji o datach wydań, przygotowywanie elementów potrzebnych do dystrybucji systemu oraz dokumentowanie wydania.