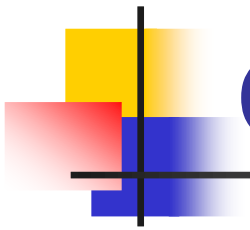
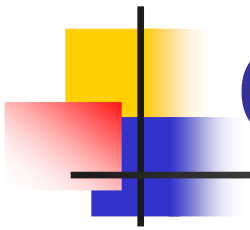


Configuration Management (CM)

Zarządzanie zmianami oraz wersjami w projekcie informatycznym

Zagadnienia

- Rola CM w procesie tworzenia oprogramowania
- Planowanie CM
- Podstawowe zadania CM
  - Zarządzanie zmianami
  - Zarządzanie wersjami
  - Pojęcia: release, version, revision
- Proces budowania systemu (*ang. build*)



CM – do czego to służy?

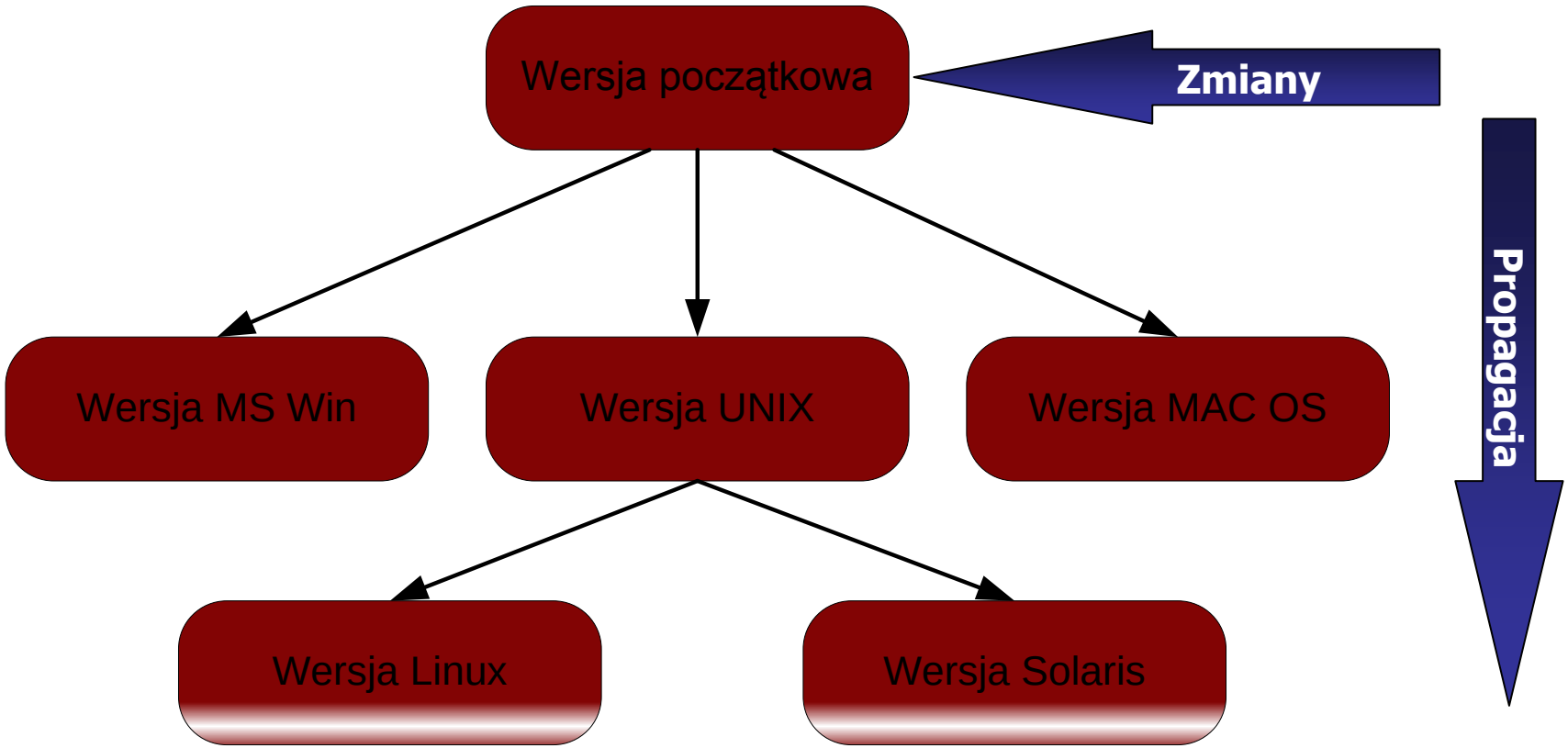
CM – do czego to służy?

- Typowa sytuacja: powstawanie kolejnych wersji systemu w cyklu jego życia
  - Porty na różne platformy sprzętowe/programowe
  - Różne wersje dostosowane do potrzeb różnych użytkowników
- Rolą CM jest zarządzanie zmianami wprowadzanymi do systemu
  - Zmiana jest czynnością zespołową
  - Kontrola kosztu (nakładu) poniesionego na implementację zmian w systemie

CM – c.d.

- Polega m.in. na tworzeniu oraz wdrażaniu procedur i standardów
  - Dotyczących sposobu implementacji zmian
- Często wchodzi w skład szerzej pojmowanego procesu zarządzania jakością (ang. *quality management*)

Wersje systemu



Standardy CM

- Podstawą CM jest zbiór standardów wdrożonych w danej organizacji
  - Nie tylko „istniejących” !
- Standardy definiują m.in.
  - Sposoby identyfikacji **CI** (ang. configuration item)
  - Sposób kontroli zmian, zarządzania wersjami
- Istnieją gotowe standardy (np. IEEE standard for CM, standard no. 828-1983)
  - Jednak w większości są one oparte o model procesu typu „waterfall” – dla procesów ewolucyjnych są mało przydatne

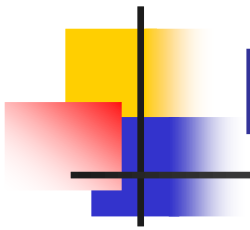
Configuration Item – def.

- „An aggregation of hardware or software or any of its discrete portions that satisfies an end use function and is designated for configuration management”
- W większości przypadków – plik
- Ale np. dokument FrameMaker-a
  - Wiele fizycznych plików
  - Jeden logiczny dokument

Jeden CI

Proces planowania CM

- Co może ulegać zmianom a więc powinno podlegać kontroli zmian?
  - Specyfikacje, dokumenty projektowe
  - Programy, testy, dane testowe
  - Podręczniki użytkownika ....
  - Ogólnie: wszystko co może okazać się przydatne dla przyszłej pielęgnacji systemu
- Dla większych projektów liczba CI jest bardzo duża
  - Ręczne zarządzanie praktycznie niemożliwe

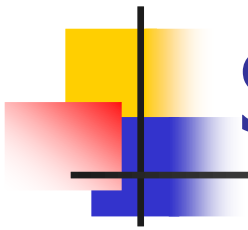


Proces planowania CM

- Rozpoczyna się we wczesnych fazach projektu
- Musi definiować CO będzie podlegać kontroli zmian (zbiór CI)

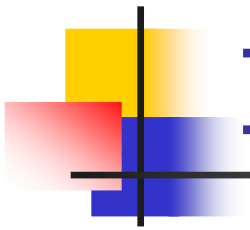
Plan CM (SCMP)

- Określa
  - Typy CI podlegających zarządzaniu
  - Konwencje nazewnictwa
- Kto jest osobą odpowiedzialną za CM w projekcie
  - Configuration Manager
- Definiuje procedury kontroli zmian, zarządzania wersjami
- Definiuje kolekcjonowane dane dotyczące CM



SCMP

- Opisuje narzędzia jakie będą wykorzystywane w procesie CM
- Procedury wykorzystywania narzędzi
- W przypadku gdy system korzysta z oprogramowania zewnętrznego
  - interfejsy
  - sposób uwzględniania zmian w oprogramowaniu zewnętrznym
- Polityka backup'ów
- Polityka release'ów



Identyfikacja CM

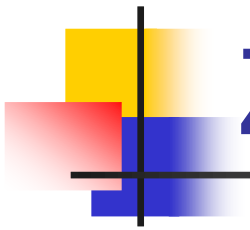
- Wszystkie dokumenty projektowe muszą być jednoznacznie identyfikowane
- Spójna konwencja nazewnictwa
- Najpopularniejsze podejście
  - Hierarchiczny schemat nazewnictwa
  - Ze względu na przeznaczenie, moduł, itp.

Hierarchia nazewnictwa

nazwa_projektu/
analiza/
    modul_1/
    modul_2/
projekt/
    modul_1/
    modul_2/
źródła/
    modul_1/
    modul_2/
testy/
    ...

Baza danych CM

- Powinna zawierać wszystkie dane dotyczące CM
- Powinna udzielać odpowiedzi na pytania takie jak:
  - Kto implementuje dany moduł?
  - Jakiej platformy wymaga dana wersja systemu?
  - Na które wersje systemu ma wpływ zamiana wprowadzona w danym komponencie?
  - Ile powstało oficjalnych wersji systemu i jakie były daty ich wypuszczenia?



Zarządzanie zmianami

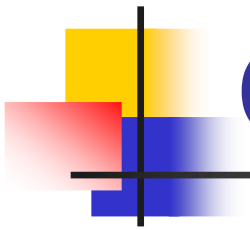
- Systemy informatyczne podlegają ciągłym zmianom. Źródła zmian:
  - Użytkownicy
  - Programiści
  - Wymagania rynku
- Zadaniem CM jest zarządzanie zmianami oraz optymalizacja kosztów wprowadzania zmian

Proces zarządzania zmianami

Zgłoszenie żądania zmiany (ang. CR – change request)
Analiza CR
if *zmiana jest uzasadniona*
  Analiza sposobu implementacji zmiany
  Ocena kosztu wprowadzenia zmiany
  Zgłoszenie CR do CCB
  if *zmiana została zaakceptowana*
    repeat
      wprowadzenie zmiany
      dowiązanie do właściwego CR-a
      zgłoszenie zmiany do akceptacji
    until *zmiana jest zaakceptowana*
    utworzenie nowej wersji systemu
  else
    odrzucenie CR-a
else
  odrzucenie CR-a

Zgłoszenie CR-a

- Postać tzw. *Change Request Form* określa SCMP
- CRF musi zawierać:
  - Opis żądanej zmiany
  - ID Zgłaszającego
  - Powód zmiany
  - Priorytet zmiany (określany przez zgłaszającego)
- Uzupełniany o:
  - Ocenę CR-a
  - Analizę wpływu na system
  - Koszt zmiany (pracochłonność)
  - Sposób implementacji



CRy a powód zgłoszenia

- Defekty
- Rozszerzenia istniejącej funkcjonalności systemu
  - Implementowane np. na życzenie klienta
- Źródła defektów
  - Problemy wykryte w czasie review
  - Problemy zidentyfikowane w czasie testów
  - Problemy zgłoszone przez użytkownika

Zgłoszenie CR-a - przykład

Projekt: TestTools **Numer:** 1122/2002

Zgłaszający: Osoba 1 **Data zgł:** 05/10/2002

Wersja: 3.0.1

Opis zmiany: Przy wyborze testu do wykonania, wybór testów powinien być początkowo zawężony do aktualnego zestawu testów

Analizujący: Osoba 2 **Data analizy:** 07/10/2002

Skojarzone moduły: test.Select, test.Execute

Ocena zmiany: Prosta w implementacji ze względu na dostępność informacji o aktualnym zestawie testów. Brak konieczności wprowadzania zmian do skojarzonych modułów

Priorytet zmiany: Niski

Zgłoszenie CR-a - przykład

Implementacja zmiany:

Estymowany wysiłek: 0,5 osobodnia

Data zgł. do CCB: 09/10/2002 **Data decyzcji:** 10/10/2002

Decyzja CCB: Zaakceptowano. Zmiana ma być zaimplementowana w wersji 3.0.2

Data zgłoszenia do testów: **Wynik testów:**

Docelowa wersja:

Rzeczywisty wysiłek:

Komentarze:

Narzędzia do śledzenia zmian

- Śledzenie statusu danego CR-a jest podstawowym problemem w procesie zarządzania zmianami
- Narzędzia tego typu (*ang. change tracking tools*) kontrolują zmiany stanu danego CR-a i automatyzują powiadamianie odpowiednich osób o zmianach stanu
- Na ogół zintegrowane z systemem emailowym – dystrybucja CR-ów bez potrzeby osobistej komunikacji
  - Przykład freeware'u: bugzilla (www.bugzilla.org)
  - Przykład komercyjnego narzędzia: Rational ClearQuest (www.rational.com)

Autoryzacja zmian a weryfikacja

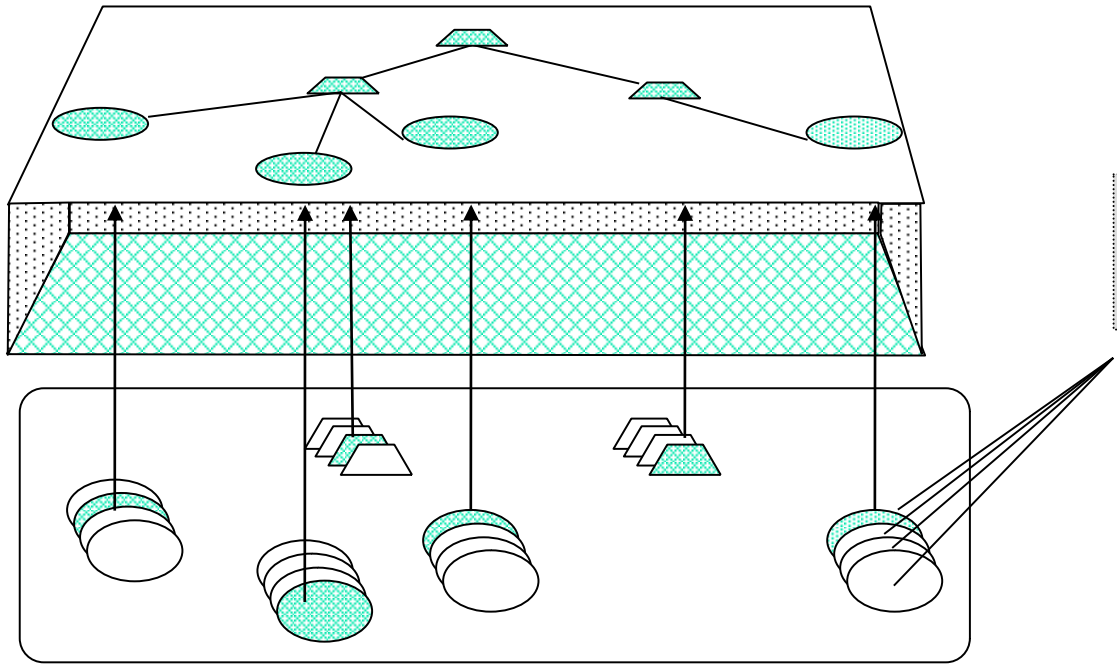
- Zawsze istnieje możliwość wprowadzenia nieautoryzowanych zmian
- Istnieją mechanizmy pozwalające zweryfikować czy bieżąca postać systemu jest tylko i wyłącznie efektem autoryzowanych zmian
- Służą do tego tzw. Configuration Status Accounting Reports (CSARs)
- Działają w oparciu o pojęcie tzw. **baseline**

Pojęcie baseline'u

- Czym jest baseline?
- Grupa configuration items (CIs) posiadających następujące właściwości:
  - Wszystkie CI przeszły proces formalnej weryfikacji
  - Są one podstawą dalszej implementacji systemu
  - Wszelkie zmiany do poszczególnych CI mogą być wprowadzane tylko za pomocą formalnych procedur

Pojęcie baseline'u (c.d.)

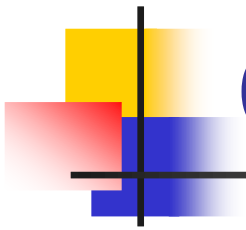
Baseline



Wersje poszczególnych elementów

Autoryzacja zmian a weryfikacja

- Configuration Status Accounting Report (CSAR)
  - Ma na celu weryfikację faktu że dana wersja systemu została stworzony zgodnie z obowiązującymi regułami
  - Identyfikacja nieautoryzowanych zmian
- Physical Configuration Audit (PCA)
  - Ma wykazać czy wszystkie CI mają poprawne wersje
  - Weryfikuje integralność danego baseline'u
  - Osoba odpowiedzialna: Configuration Manager

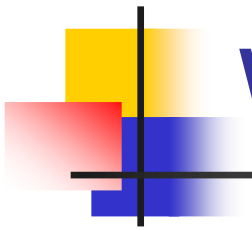


CCB

- Wszelkie zmiany wprowadzane w obrębie systemu powinny podlegać weryfikacji czy są one zasadne
  - Nie tylko z technicznego punktu widzenia!
  - Cele strategiczne i organizacyjne
- Czym jest CCB?
- CCB autoryzuje wszystkie zmiany dotyczące jakiegokolwiek CI
- W skład CCB wchodzą:
  - Project Leader
  - Configuration Manager
  - Test Leader

Zarządzanie wersjami i release'ami

- Definiuje system identyfikacji poszczególnych wersji systemu
- Planuje kiedy będą dostępne kolejne wersje systemu
- Kontroluje poprawne stosowanie procedur i narzędzi do zarządzania wersjami
- Zawiera plan dystrybucji kolejnych release'ów systemu

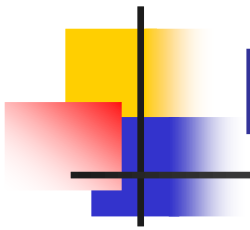


Wersja vs. wariant vs. release

- **Wersja** Postać systemu funkcjonalnie różna od pozostałych postaci systemu
- **Wariant** Postać systemu funkjonalnie identyczna ale niefunkcjonalnie różna od pozostałych wariantów systemu
- **Release** Postać systemu, która została udostępniona użytkownikom spoza zespołu tworzącego system

Release systemu

- Składa się nie tylko z plików wykonywalnych
- Zawiera również
  - Pliki konfiguracyjne systemu dla danej konkretnej instalacji
  - Pliki z danymi niezbędnymi do działania systemu
  - Programy instalacyjne dla danej platformy sprzętowej/programowej
  - Dokumentację
- Stosuje się różne fizyczne media: tasiemki, CD, dyskietki
  - Klient zawsze musi otrzymać fizyczną kopię systemu



Polityka release'ow

- Kolejny release systemu musi zawierać poprawki błędów wykrytych przez użytkowników oraz uwzględniać zmiany w konfiguracji sprzętowej
- Nowa funkcjonalność systemu
- Problem polityki release'ow: którą kolejną wersję systemu opublikować jako release?
  - Nacisk na szybkość reakcji na potrzeby zgłaszane przez użytkowników
  - Kiedy kolejny release można uznać za dostatecznie przetestowany?

Release – możliwe problemy

- Użytkownik systemu niekoniecznie chce aby instalować nową wersję
  - Czasami funkcjonalność oferowana w nowej wersji jest postrzegana jako gorsza od istniejącej w bieżącej
- Nie można zakładać że przed instalacją danego release'u wszystkie wcześniejsze zostały zaakceptowane przez klienta
  - Procedura instalacji release'u powinna być niezależna od aktualnej wersji zainstalowanej w środowisku docelowym
  - Instalacja przyrostowa – bardzo podatna na błędy

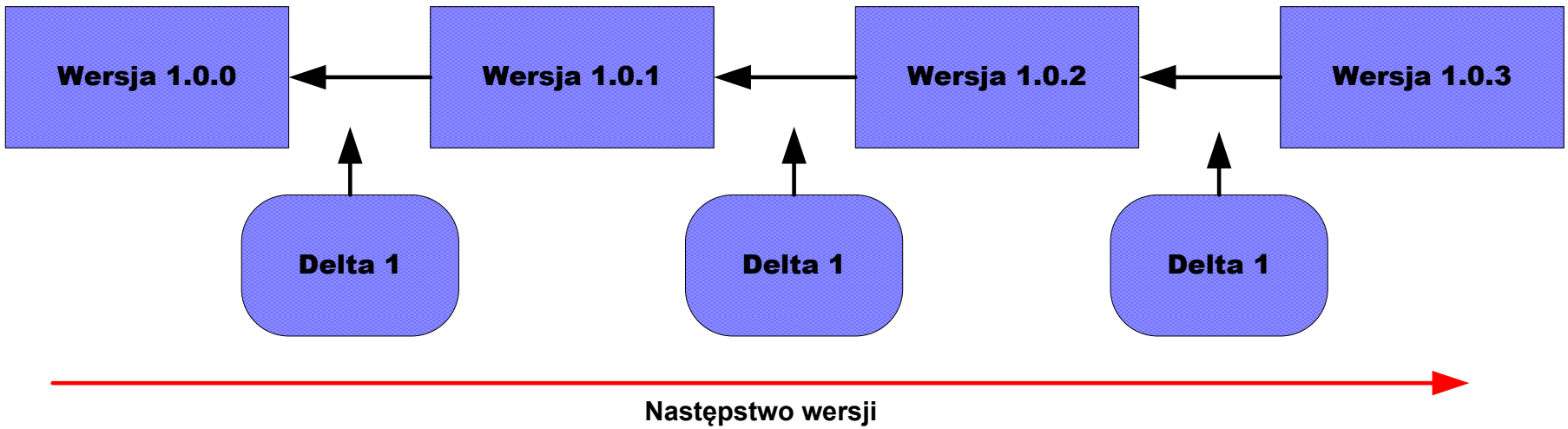
Narzędzia wspomagające zarządzanie wersjami

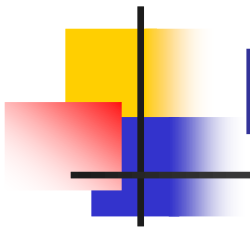
- Identyfikacja wersji i release'ow
  - Każda kolejna wersja systemu w razie potrzeby może być odtworzona
- Kontrolowane wprowadzanie zmian
  - Narzędzia umożliwiają równoległe prace na różnych wersjach systemu
  - Możliwość scalania (ang. merging) zmian pochodzących od różnych autorów
- Historia nanoszonych zmian
  - Każdy CI posiada dokumentację wszystkich wprowadzonych zmian
  - Zwykle opis danej zmiany – referencja do odpowiedniego CR-a

RCS - Revision Control System

- RCS - jedno z pierwszych narzędzi kontroli wersji
- Stosunkowo małe wymagania dot. pamięci dyskowej
  - Fizycznie zapisywane są tylko zmiany w kolejnych wersjach, nie całe wersje
- Odzyskiwanie wcześniejszej wersji
  - Wprowadzone zmiany są aplikowane do wersji najnowszej
- Umożliwia odtworzenie dowolnej wcześniejszej, nazwanej wersji
- Umożliwia niezależne, równoległe tworzenie kolejnych wersji

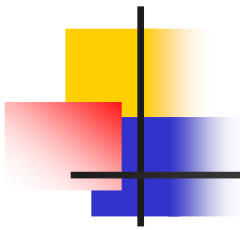
Przyrosty w RCS





RCS - ograniczenia

- RCS powstał jako system kontroli wersji kodu
  - Przeznaczony do pracy z plikami ASCII
- Nie da się go wykorzystywać do pracy z plikami binarnymi
- Tekstowy interfejs użytkownika
  - Mało praktyczna funkcjonalność porównywania poszczególnych wersji



CVS

- Repozytorium plików tekstowych
  - Umożliwia pobieranie (ang. checkout), modyfikację i zapisywanie (ang. checkin) poszczególnych elementów
  - Poprzednie wersje są cały czas dostępne
    - Każda operacja check-in tworzy nową wersję
    - Możliwość odtwarzania poprzednich wersji, porównywania (diff)
  - Możliwość oznaczania (ang. tag) kolejnych wersji w celu łatwiejszej identyfikacji
- Idea działania podobna do RCS, CMS, innych ...
- Istotna przewaga: operacja check-out nie jest wyłączna
  - W tej samej chwili więcej niż jedna osoba może pobrać (check-out) ten sam plik
  - Poszczególne osoby pracują na własnych, prywatnych wersjach kodu
  - Zmiany do repozytorium mogą być nanoszone równocześnie z różnych źródeł
  - CVS rozwiązuje (prawie) wszystkie konflikty przy nanoszeniu zmian do repozytorium

CVS – fragment manual'a (1/5)

- Simple usage: checkout and update
  - Getting a copy of the most recent contents of a package Foo:
    - cvs checkout Foo
  - Getting a copy of version (tag) V00-02-23 of a package Foo:
    - cvs checkout -r V00-02-23 Foo
  - These produce fully editable Foo directories, etc
  - To update a directory to the most recent contents:
    - cvs update -A
  - To see what an update will change, without actually changing
    - cvs -n update -A
  - Update flags:
    - U update M modified A added
    - C conflict ? unknown D deleted

CVS – fragment manual'a (2/5)

- Committing changes back to the repository
  - To put your changes back into the repository:
    - Merge in any changes since your checkout
      - cvs update -A
    - commit:
      - cvs commit
  - Many options:
    - Specify comment for logs from command line
    - Commit only one file
    - Control processing of subdirectories
  - Possible failures
    - Can't get a temporary lock on the repository
    - Conflict during update

CVS – fragment manual'a (3/5)

- Adding and removing files
  - To tell CVS a new file exists:
    - First create the file, then
      - cvs add <name>
      - cvs commit
    - Nothing changes in the repository until the commit
  - To tell CVS a file is no longer needed
    - First delete the file, then
      - cvs rm <name>
      - cvs commit
    - Nothing changes in the repository until the commit

CVS – fragment manual'a (4/5)

- Labeling particular contents for later
  - To add a particular label to certain contents:
    - Make sure that everything is in the repository
      - cvs update
      - cvs commit
    - Tell CVS to add a tag to the current contents
      - cvs tag <string>
  - Tags are an easy way to communicate with your colleagues
    - "I just fixed that in jake20010924a, give it a try"
    - This bug is back in V00-03-04, I thought it was fixed in V00-03-02"
  - Web based tools exist for seeing what changed, who changed it, etc ➔ WebCVS

CVS – fragment manual'a (5/5)

- Conflict resolution & parallel development
  - Its rare for developers to really conflict by changing the same line
    - Usually only one person working on a particular piece of functionality
    - And people working on the same thing should talk to each other!
    - Conflicts happen most often during migrations of the code
  - When it happens, CVS can't figure out how to cope
    - Marks both sets of changed lines with markers

```
<<<<<<<<<<<<<<<<
One content
================
Other content
>>>>>>>>>>>>>>>>
```

    - User has to edit this to select one or other, or combine

Rational ClearCase

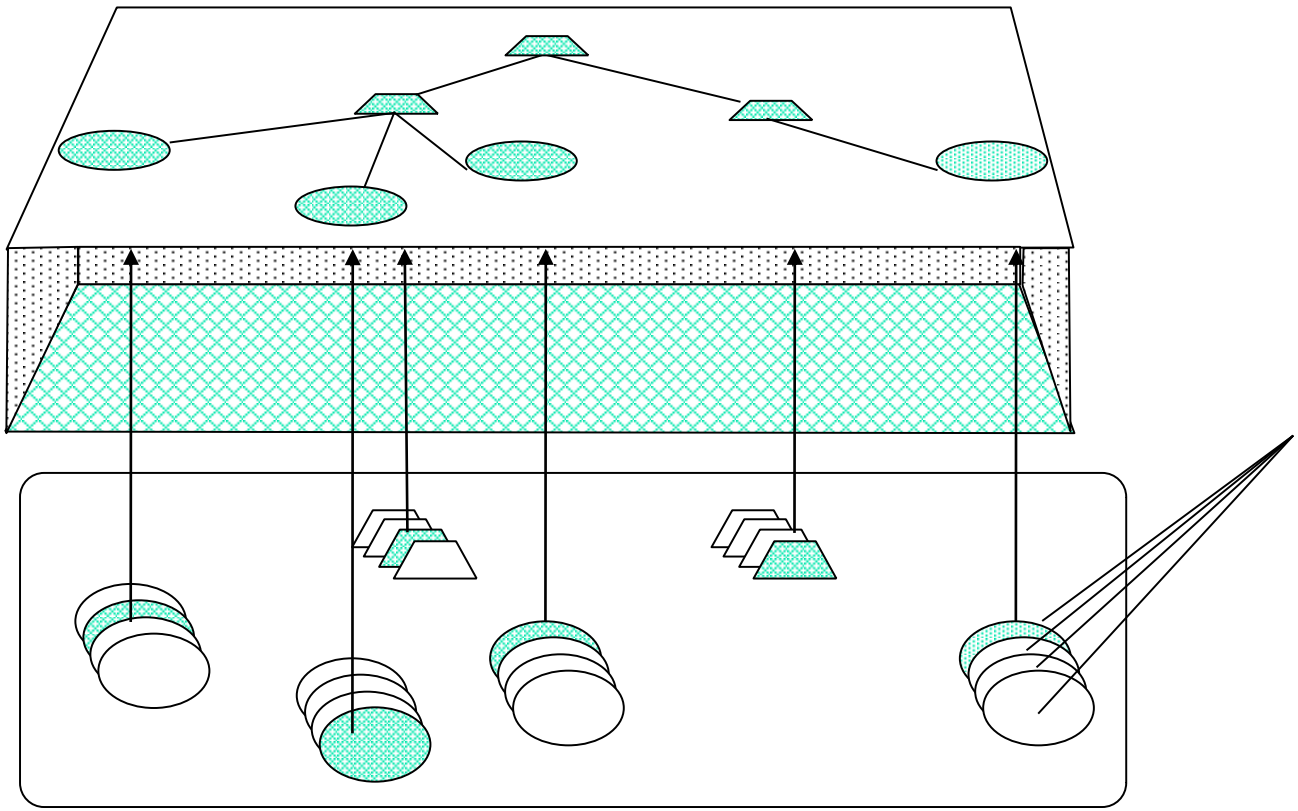
- Pojęcie VOB-a
- A Versioned Object Base (VOB) – kompletne, nieusuwalne repozytorium zawierające informacje o plikach i ich poszczególnych wersjach
- VOB zawiera w sobie informacje o efektach pracy całej grupy (pliki, katalogi oraz ich wersje). Nie ma ograniczeń dotyczących typów poszczególnych plików
- **ClearCase** zawiera w sobie historie zmian wszystkich plików i katalogów VOB-a. Cecha ta umożliwia łatwe odtworzenie dowolnego wcześniejszego środowiska (wystarczy parę poleceń)

Rational ClearCase

Pojęcie widoku (*ang. view*)

1. Widok udostępnia jedną (w pewnych przypadkach zero) określoną wersję każdego z elementów VOB-a
2. Widok stwarza nam własne, prywatne środowisko pracy
3. Widok tworzy warstwę plików wykorzystywanych tylko przez daną osobę (ang. view-private files) w oparciu o informacje o plikach zawarte w VOB-ie (dostępne dla wszystkich korzystających z VOB-a). Powyższe dwa rodzaje informacji razem są widoczne przez danego użytkownika jako elementy tworzące jedną, spójną hierarchię katalogów i plików

Rational ClearCase – VOB vs. view



Wersje poszczególnych elementów

Proces budowania systemu

- Polega na scaleniu poszczególnych części systemu w odpowiednich wersjach w wykonywalną całość
- Różne kombinacje poszczególnych wersji komponentów systemu dają w efekcie różne wersje systemu
- Proces generowania systemu może trwać od paru minut do paru dni

System build - problemy

- Czy w skład systemu wchodzą wszystkie potrzebne komponenty?
  - W normalnych warunkach wykrywane w fazie scalania (linking)
- Czy w skład budowanej wersji systemu wchodzą właściwe wersje komponentów?
  - Bardzo niebezpieczne – początkowo system może działać „poprawnie” a zacząć ujawniać defekty po wdrożeniu
  - Stąd testowanie – integralna część procesu wdrożenia!

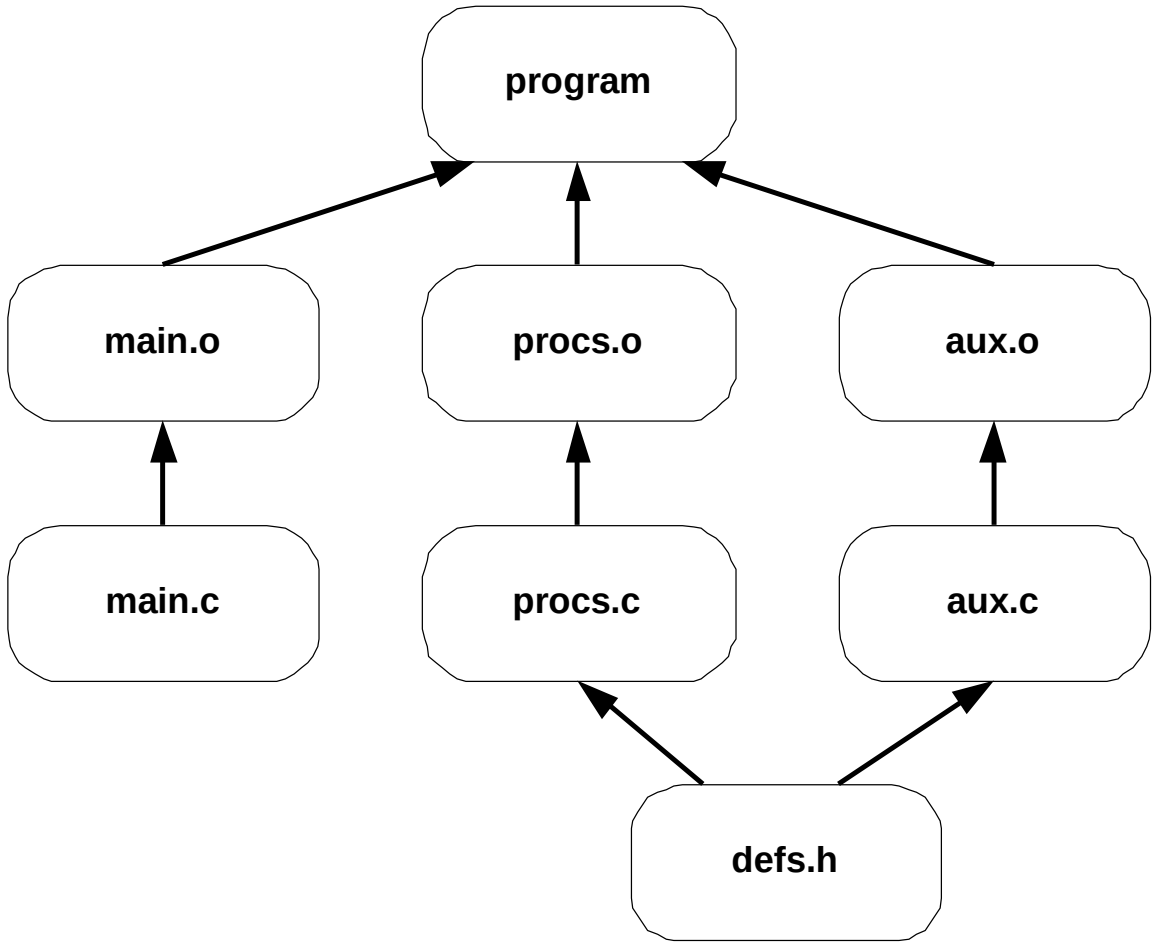
System build - problemy

- Czy wzajemne referencje w komponentach są poprawne?
  - Należy unikać używania bezwględnych ścieżek do plików w kodzie źródłowym
- Czy build systemu odbywa się na właściwą platformę?
  - Uruchomienie procesu na inną wersję tego samego OS może sprawiać początkowo wrażenie poprawnego
- Czy do procesu buildowania wykorzystujemy odpowiednie wersje narzędzi?
  - Kompilatory, linkery, itp.

System build - Make

- Najpopularniejsze narzędzie wykorzystywane pod Unix-em
- Bazuje na specyfikacji zależności pomiędzy komponentami. Powtórna kompilacja jest wymuszana w przypadku, gdy kod źródłowy został zmieniony po wygenerowaniu pliku pośredniego (*.o)
  - Porównywanie czasów modyfikacji plików

Make - zależności



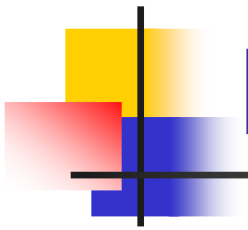
Make - problemy

- Narzędzie oparte o fizyczne a nie logiczne właściwości poszczególnych komponentów
  - Np. zmiana komentarza ➔ rekompilacja
- Specyfikacje zależności bardzo szybko rosną wraz z wzrostem liczby komponentów i stają się kosztowne w pielęgnacji
- Uciążliwa specyfikacja wersji narzędzi jakie mają zostać użyte do budowania systemu
- Brak integracji z narzędziami do kontroli wersji
  - Niektóre wersje współpracują z RCS, ale jedyna funkcjonalność ➔ checkout ostatniej wersji

clearmake – integracja z narzędziem kontroli wersji

Kolejne etapy występujące w procesie buildu

- Wywołanie clearmake, z możliwą specyfikacją obiektów docelowych (*ang. targets*)
- clearmake odczytuje zero lub więcej plików Makefile, zawierających obiekty docelowe oraz powiązane z nimi skrypty je tworzące
- clearmake posiada możliwość uzupełnienia poleceń występujących w plikach Makefile o własne wbudowane reguły
- clearmake w miarę możliwości wykorzystuje ponownie uprzednio zbudowane komponenty
- W przypadku braku możliwości wykorzystania uprzednio zbudowanego elementu, clearmake buduje (tworzy) go



Podsumowanie

- CM
  - Zajmuje się zarządzaniem zmianami wprowadzanymi do systemu
  - Jest częścią szerszego procesu zarządzania jakością
- W zakres CM wchodzi planowanie oraz zarządzanie:
  - zmianami
  - buildami systemu
  - wersjami oraz release'ami
- Proces budowania polega na scalaniu systemu z komponentów – odbywa się przy wykorzystaniu narzędzi takich jak np. Make
- Zintegrowane narzędzia CM (np. clearmake) pozwalają na połączenie procesów budowania oraz zarządzania wersjami