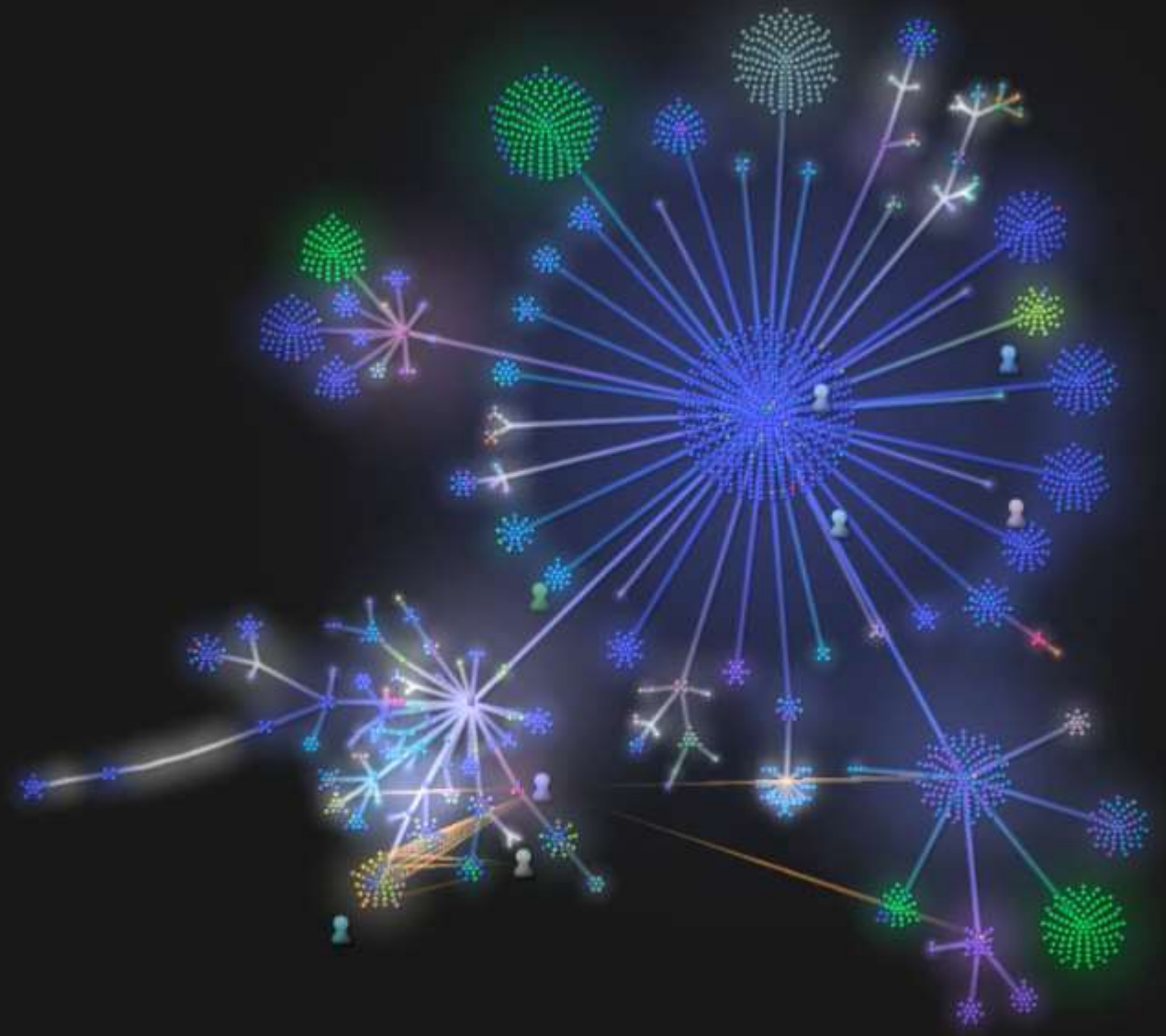
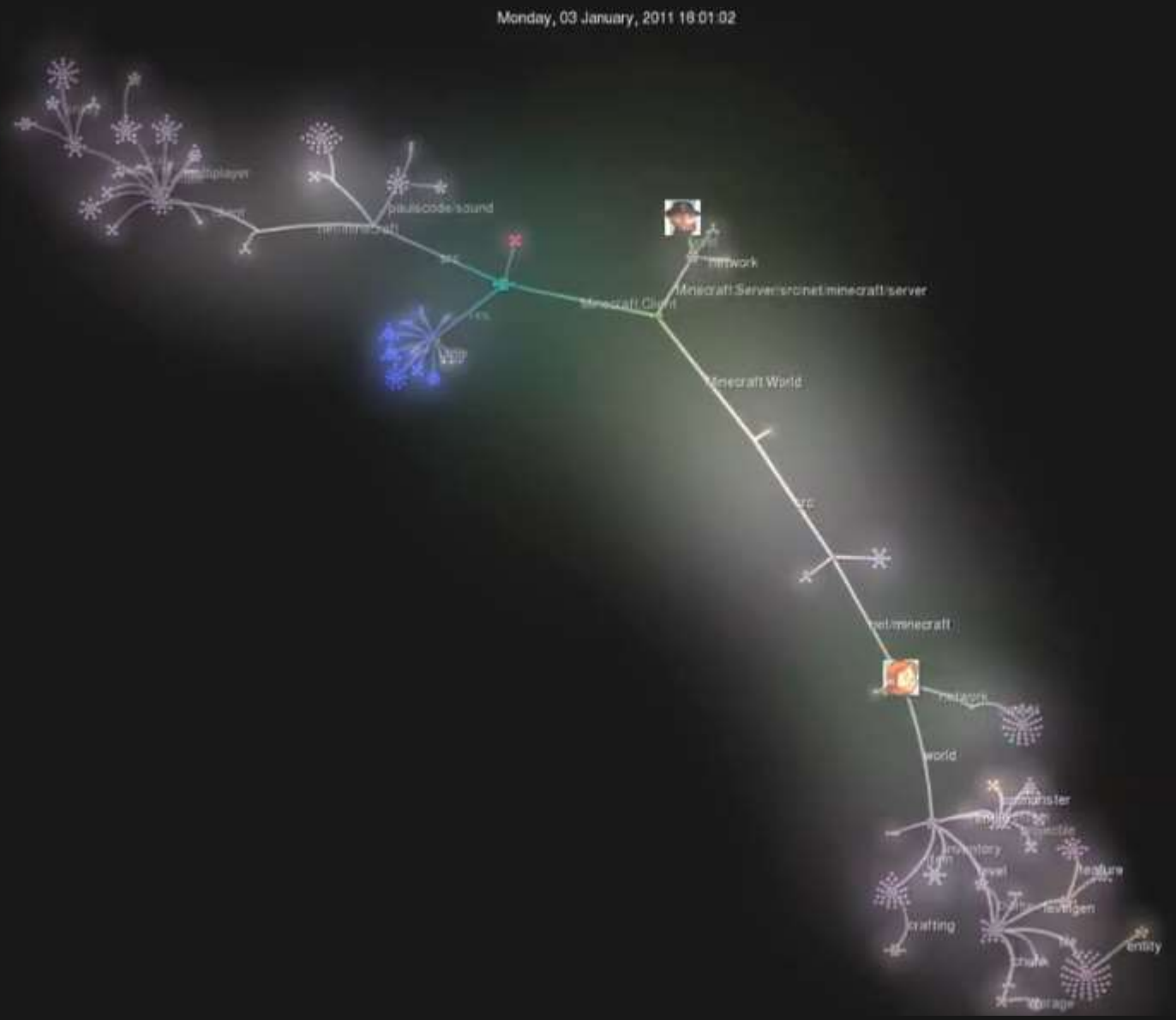
PRODUCENT OPROGRAMOWANIA tkomp

Jak programować w zespole i się nie pozabijać

czyli strategia i praca z gałęziami w repozytoriach kodu Mercurial.

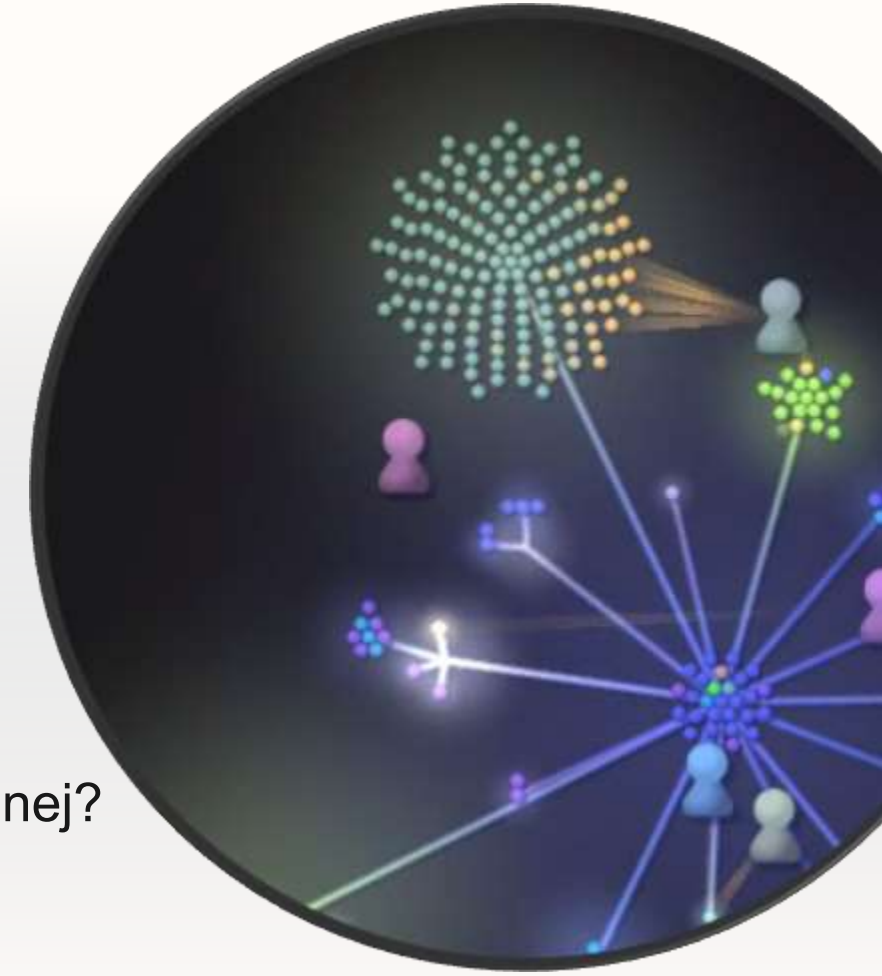
Adam Kruszewski
<adam.kruszewski@tkomp.pl>




Monday, 03 January, 2011 18:01:02
paulscode/sound
network
Minecraft.Server/src/net/minecraft/server
Minecraft World
net/minecraft
world
crafting
entity

Ocena zespołu programistów w 5 minut
Test Joela*

1. **Czy stosujesz system kontroli wersji?**
2. Czy możesz zbudować wersję w jednym kroku?
3. Czy stosujesz codzienne budowanie wersji?
4. Czy używasz system zarządzania błędami?
5. Czy usuwasz błędy zanim napiszesz nowy kod?
6. Czy masz harmonogram aktualizowany na bieżąco?
7. Czy masz specyfikację?
8. Czy programiści mają komfortowe warunki pracy?
9. Czy używasz najlepszych dostępnych narzędzi?
10. Czy masz testerów?
11. Czy kandydaci piszą programy podczas rozmowy kwalifikacyjnej?
12. Czy praktykujesz korytarzowe testy wygody użytkowania?

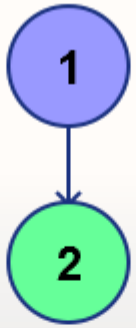


* Joel Spolsky - http://polish.joelonsoftware.com/Articles/TheJoelTest.html

Czym tak naprawdę jest system kontroli wersji?


```
int main (int argc, char **argv) {
      printf(„Witaj przygodo!\n”);
}
```

Czym tak naprawdę jest system kontroli wersji?




```
int main (int argc, char **argv) {
      printf(„Witaj przygodo!\n”);
+ return 0;
}
```

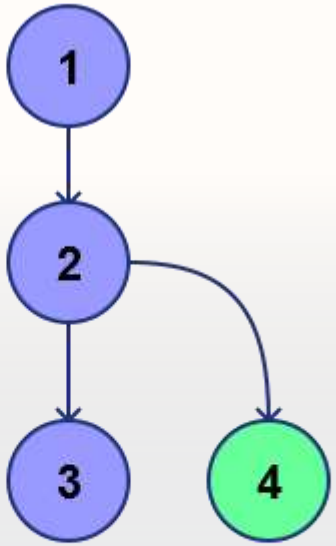
Czym tak naprawdę jest system kontroli wersji?


```
+#include <stdio.h>

int main (int argc, char **argv) {
      printf(„Witaj przygodo!\n”);
return 0;
}
```

Czym tak naprawdę jest system kontroli wersji?

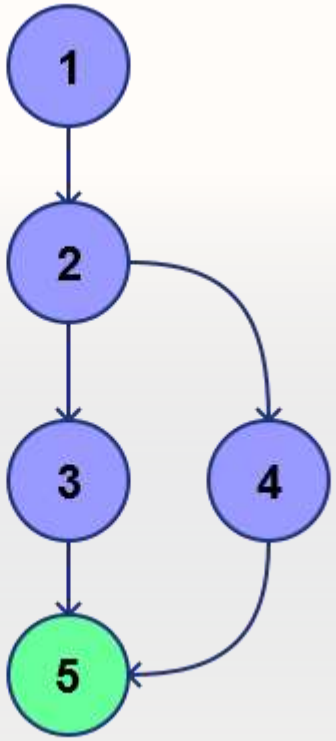

```
int main (int argc, char **argv) {
    printf(„Witaj przygodo!\n”);
return 0;
}
```

Czym tak naprawdę jest system kontroli wersji?



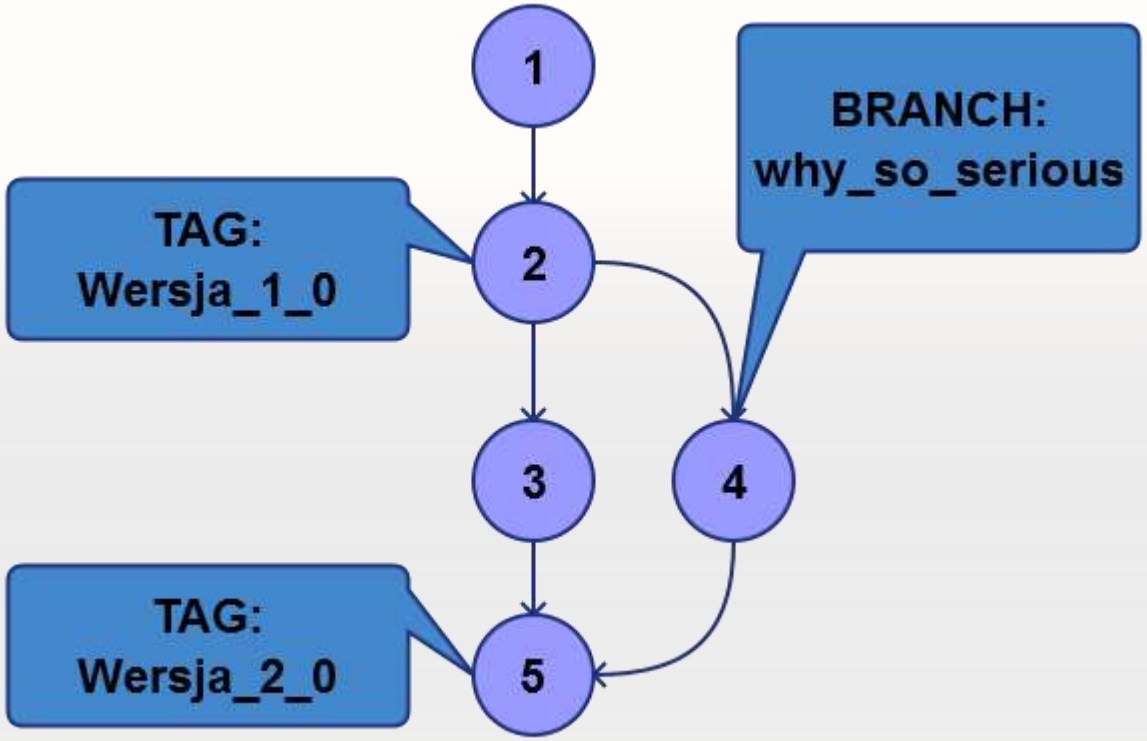

```
int main (int argc, char **argv) {
      printf(„Witaj przygodo!\n”);
-   return 0;
+ return -1;
}
```

Czym tak naprawdę jest system kontroli wersji?



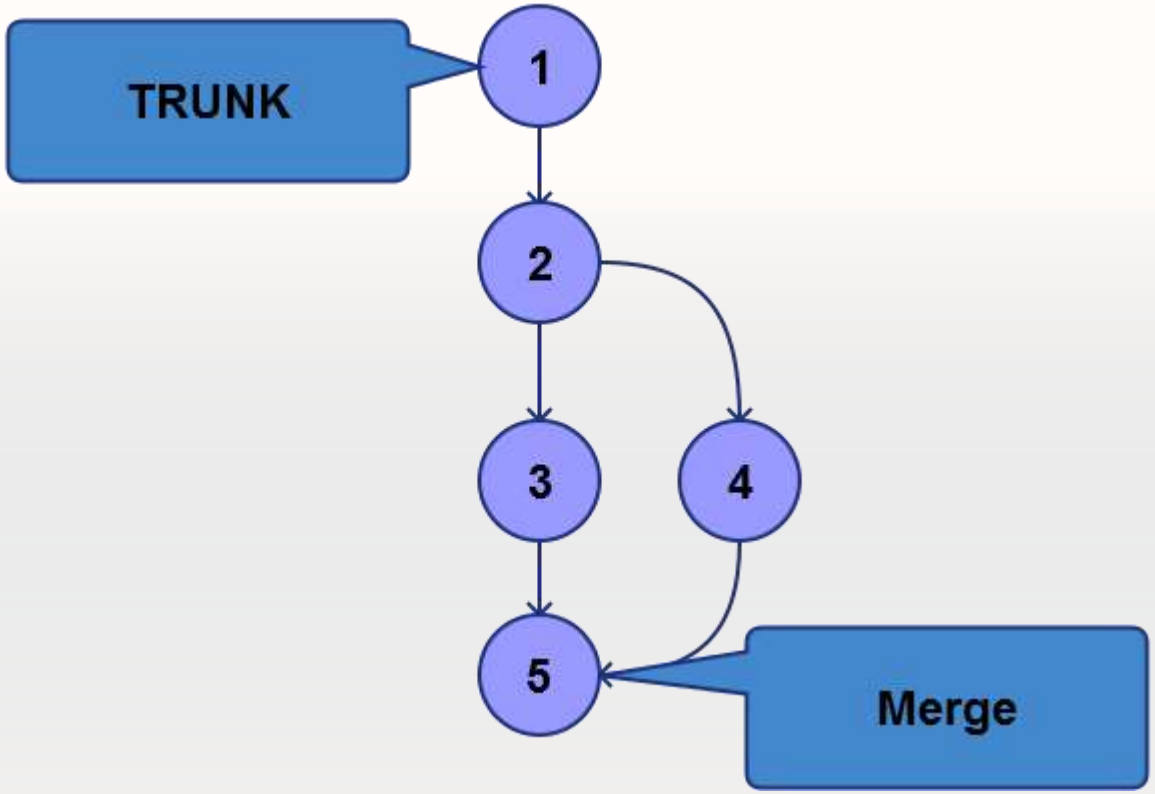

```
+#include <stdio.h>

int main (int argc, char **argv) {
      printf(„Witaj przygodo!\n”);
-  return 0;
+ return -1;
}
```

Czym tak naprawdę jest system kontroli wersji?



- Graf skierowany składający się z „łatek”
- Przechowuje dodatkowe metadane, tj.
  - TAG-i
  - Nazwane gałęzie (BRANCH)

Czym tak naprawdę jest system kontroli wersji?



- Posiada korzeń (trunk)
- Jasno określone miejsca złączeń gałęzi (MERGE)

A jak radzili sobie jaskiniowcy?

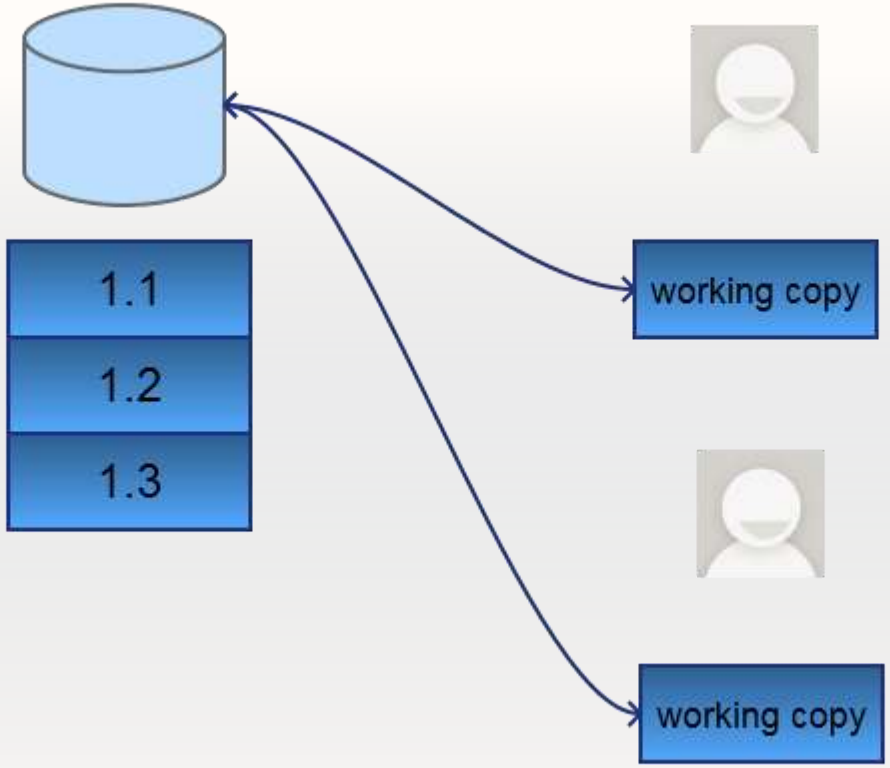
Czyli krótka historia systemów kontroli wersji.

- RCS (1982)
- CVS (1986)
- Visual Source Safe (1994; Microsoft)
- CVSNT (1998)
- BitKeeper (1998; Własnościowy)
- Subversion (2000)
- Team Foundation Server (2005; Microsoft)

Ekran początkowy gry komputerowej Prehistoric
źródło: http://en.wikipedia.org/wiki/Prehistorik

A jak radzili sobie jaskiniowcy?

Czyli krótka historia systemów kontroli wersji.



A jak radzili sobie jaskiniowcy?

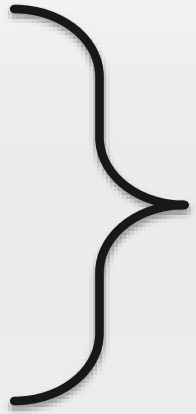
Czyli krótka historia systemów kontroli wersji.

> “ if you actually like using CVS, you shouldn't be here. You should be in some mental institution, somewhere else. ”
>
> - Linus Torvalds, 2007
> https://www.youtube.com/watch?v=4XpnKHJAok8

A jak radzili sobie jaskiniowcy?

Czyli krótka historia systemów kontroli wersji.

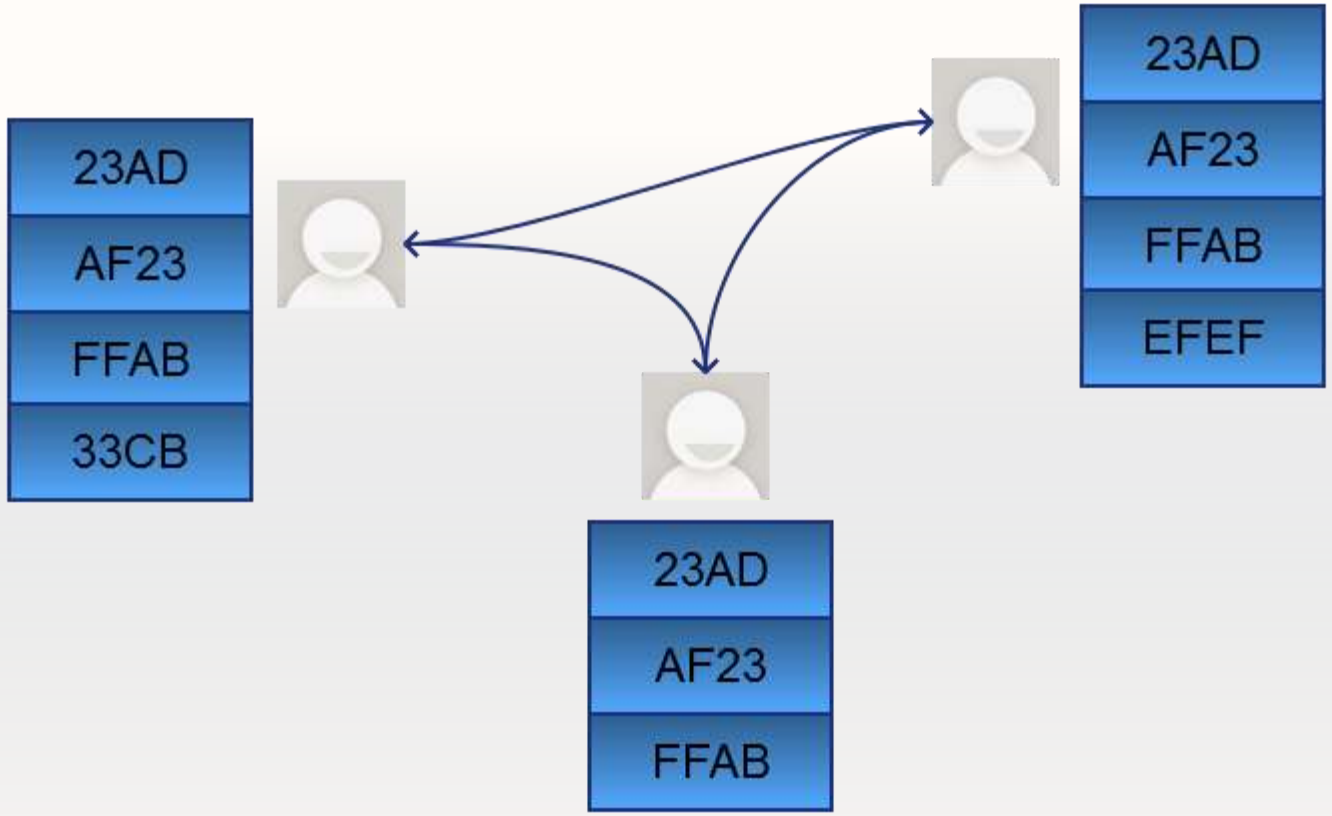
- Mercurial
- Git
- Bazaar



2005

A jak radzili sobie jaskiniowcy?

Czyli krótka historia systemów kontroli wersji.



Kto tego w ogóle używa?

- Mercurial
  - Facebook, Mozilla, Python, OpenJDK, OpenOffice
    (za http://mercurial.selenic.com/wiki/ProjectsUsingMercurial)
- Git
  - Linux Kernel, KVM, Android (Google), Fedora, GCC, Perl
    (za https://git.wiki.kernel.org/index.php/GitProjects)
- Bazaar
  - Ubuntu, MySQL, Squid
    (za http://wiki.bazaar.canonical.com/WhoUsesBzr)

Kto tego w ogóle używa?

- Mercurial
  - Bitbucket (http://bitbucket.org)
  - Kiln (https://www.fogcreek.com/kiln/)
- Git
  - GitHub (http://github.com)
  - Kiln (https://www.fogcreek.com/kiln/)

A tak „krok po kroku” to jak się tego używa?

- Mercurial
  - Joel Spolsky – HgInit (http://hginit.com/)

- Git
  - Getting Git Right (https://www.atlassian.com/git/)

Nie lubię linii poleceń, co zrobić?

- Mercurial
  - TortoiseHG (http://tortoisehg.bitbucket.org/)
  - SourceTree (http://www.sourcetreeapp.com/)
- Git
  - GitHub (http://github.com)
  - SourceTree (http://www.sourcetreeapp.com/)

Dlaczego chcesz mnie zabić?!

- commit kodu, który się nawet nie kompiluje (np. nie dodany nowy plik źródłowy)
- „force commit” / „force push” (tj. moje zmiany są lepsze od waszych)
- zmiany API „z partyzanta”
- możesz być kreatywny… ;-)

Jack Nicholson
w filmie „The Shining”

Co zrobić aby się nie pozabijać.

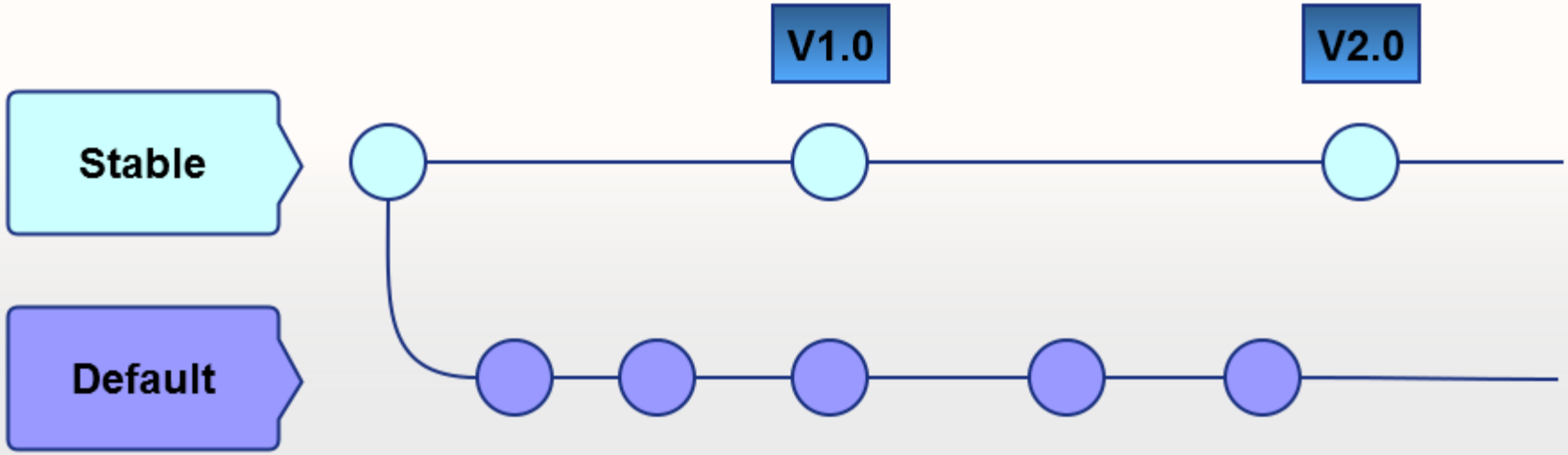
- Implementacja 12 punktów testu Joela.

- Konwencja pracy z repozytorium nazwana „Git Flow” (http://nvie.com/posts/a-successful-git-branching-model/)

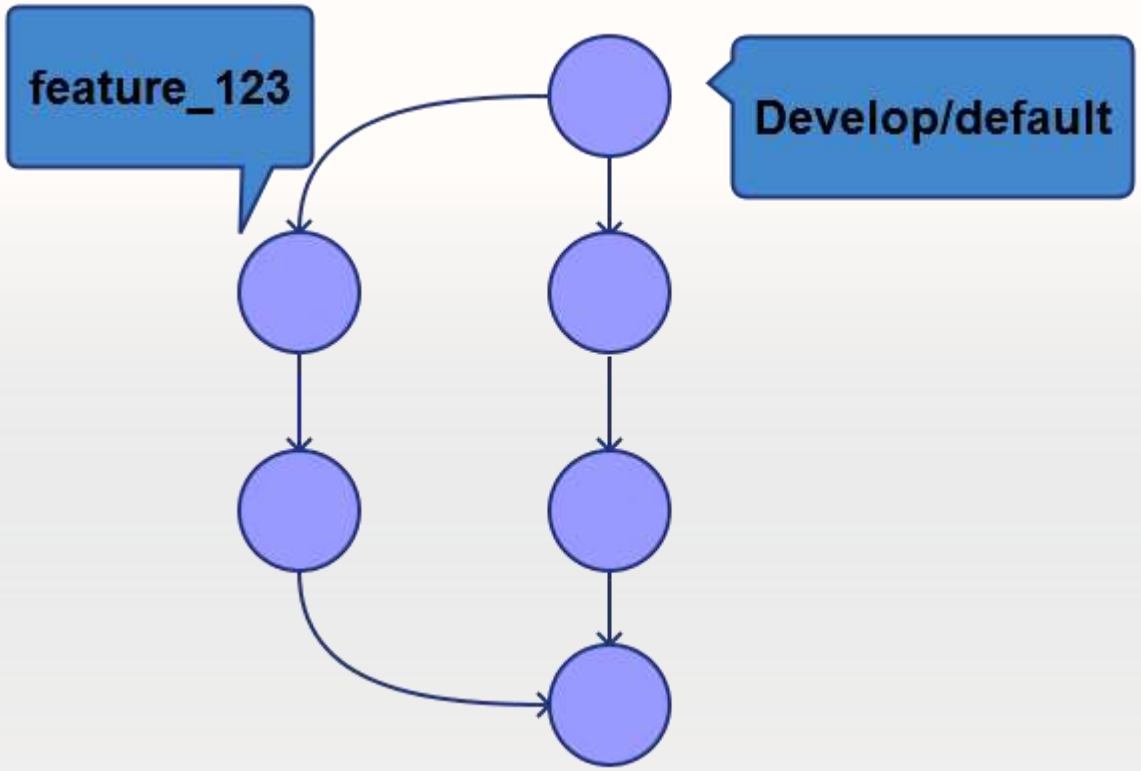
Im dalej w las tym więcej…

- Dwie stałe gałęzie:
  - Develop/default (przechowuje aktualną wersję rozwojową)
  - Master/stable (śledzi historię wydań aplikacji)

Stable i default, dwa bratanki.

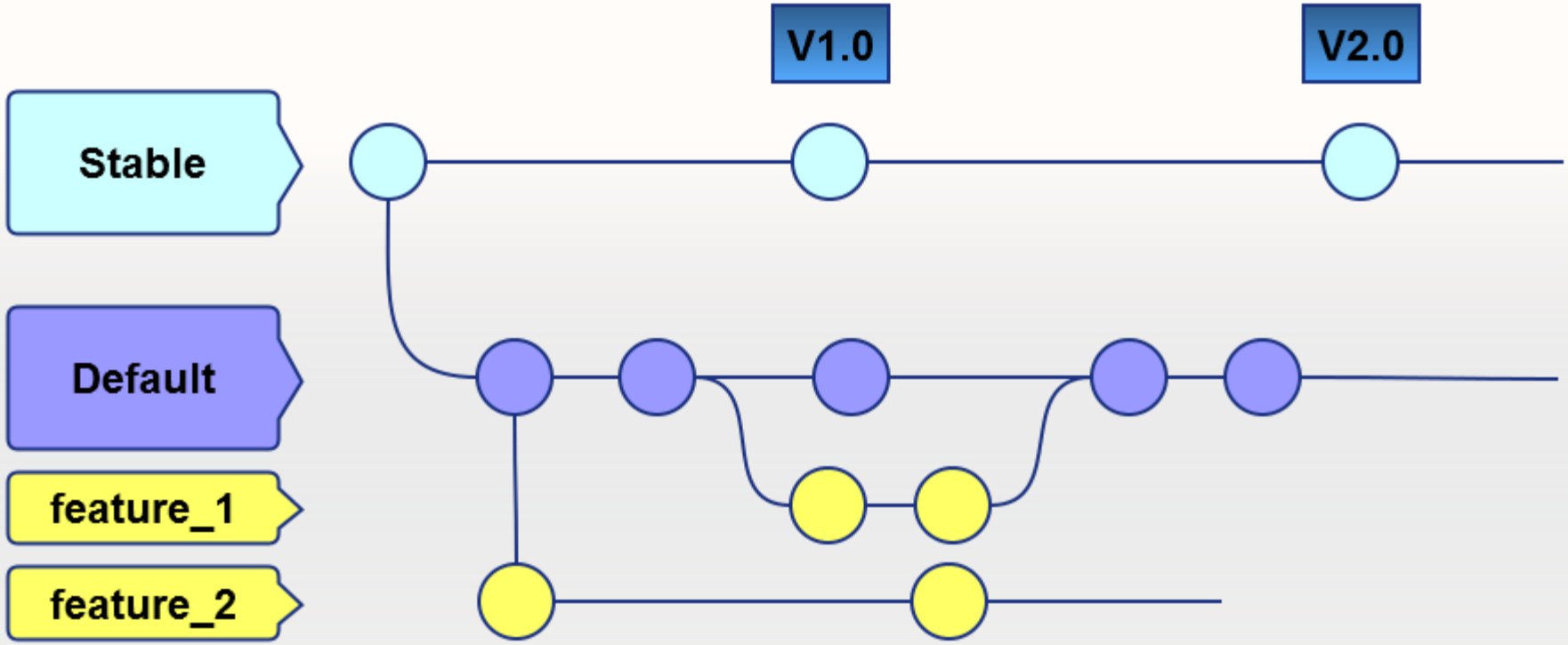


Nowe rzeczy dodawaj w swojej piaskownicy.



- Wiele gałęzi tworzonych w razie potrzeb:
  - Feature branch (tworzona na potrzeby wprowadzenia nowej funkcjonalności; zawsze pochodzi z develop/default i jest mergeowana tylko do develop/default)

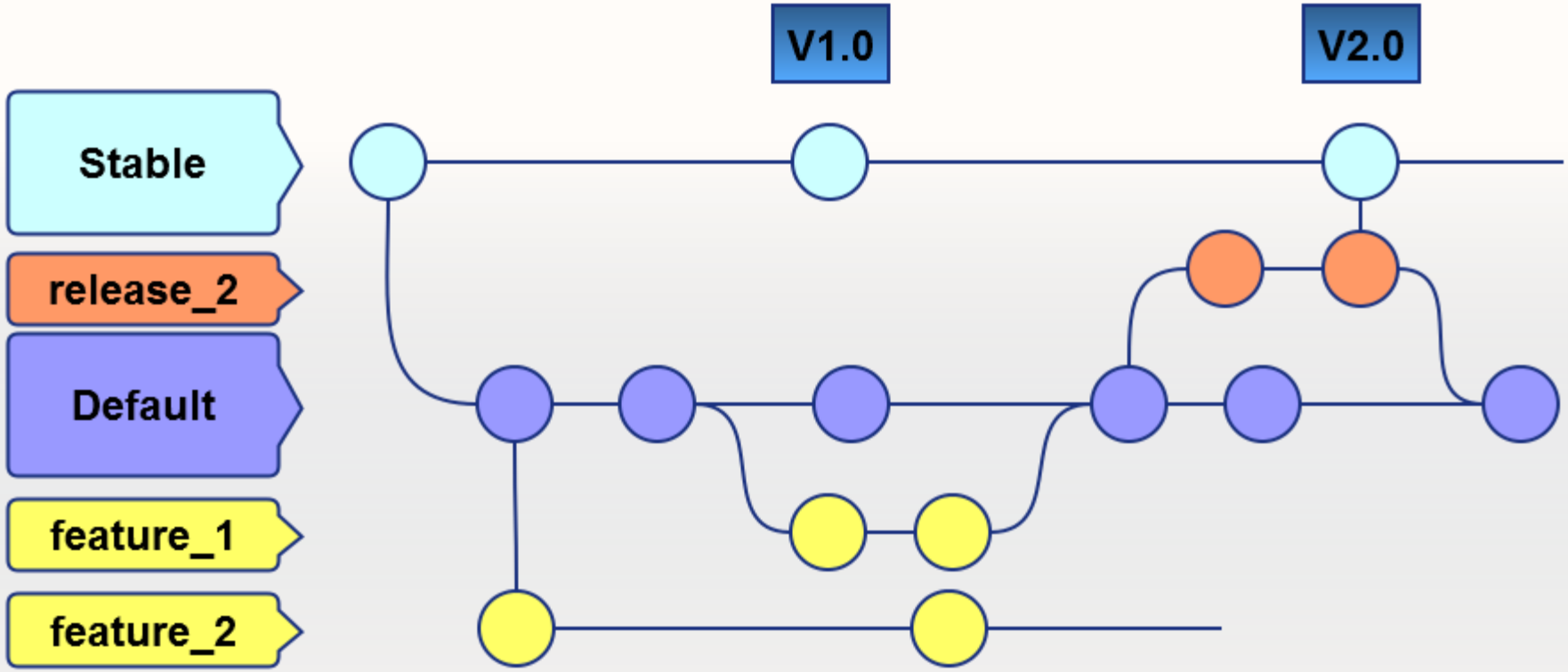
Nowe rzeczy dodawaj w swojej piaskownicy.



Stabilizacja następuje obok prac rozwojowych.

- Gdy gałąź rozwojowa będzie posiadała wystarczającą ilość nowych funkcjonalności/zmian aby móc wydać wersję, wtedy tworzy się z niej nową gałąź przeznaczoną do stabilizacji nowej wersji.
- Po ustabilizowaniu tej gałęzi, łączy się ją do gałęzi stabilnej (master/stable) oraz łączy się dodane w niej poprawki do gałęzi rozwojowej (default/develop).

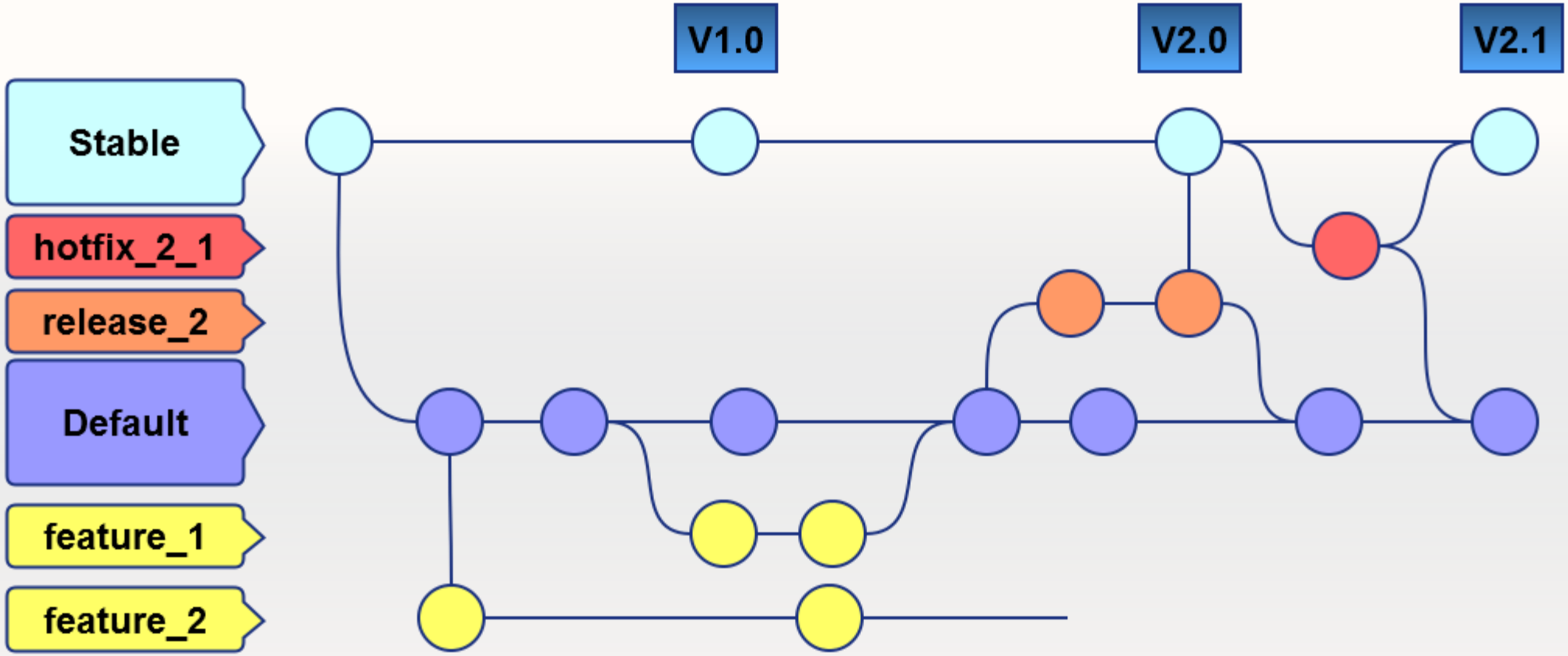
Stabilizacja następuje obok prac rozwojowych.



Co z poprawą błędów, zgłoszonych w wersji stabilnej?

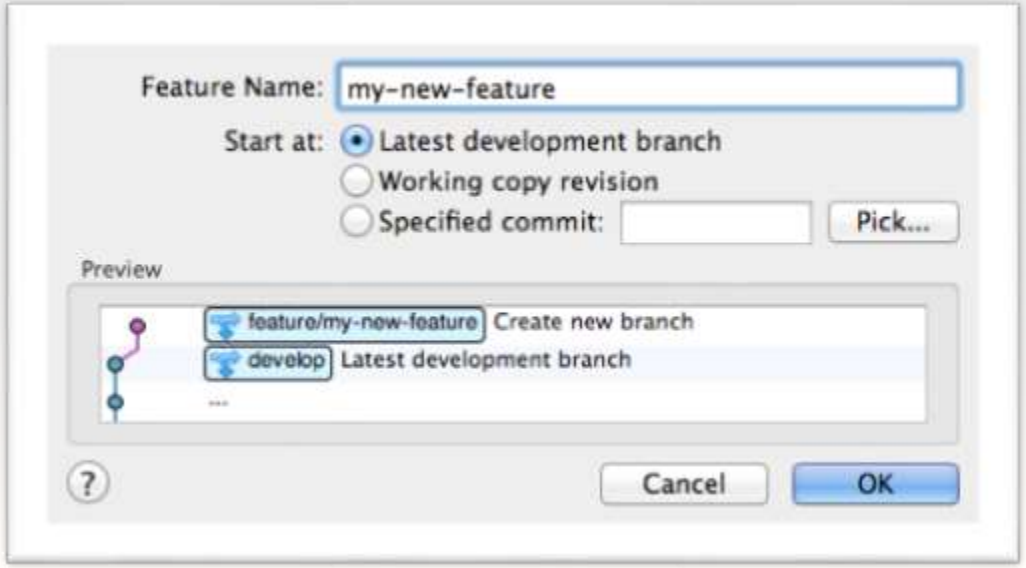
- Poprawę błędów w gałęzi master/stable dokonuje się w utworzonych w tym celu gałęzi typu „hotfix”, które „odgałęziamy” ze znacznika (tag), które wskazuje na tą dokładnie wersję, w której naprawiamy błąd.
- Poprawkę błędu łączymy ze stable oraz włączamy również do gałęzi rozwojowej (default/develop).

Co z poprawą błędów, zgłoszonych w wersji stabilnej?



Czy wymaga to zapamiętania 101 dziwnych komend?

- Mercurial posiada rozszerzenie hg flow (https://andy.mehalick.com/2011/12/24/an-introduction-to-hgflow)
- Git posiada rozszerzenie git-flow (http://yakiloo.com/getting-started-git-flow/)
- Obie wtyczki są obsługiwane przez SourceTree w graficznym GUI. (https://blog.sourcetreeapp.com/2012/08/01/smart-branching-with-sourcetree-and-git-flow/)



Dobry opis zmiany (commitu)

- W pierwszej linii, najlepiej w ramach 50 znaków, opisz krótko daną zmianę.
- Następną linię pozostaw pustą, a poniżej postaraj się odpowiedzieć na pytania: dlaczego ta zmiana była wymagana, jak rozwiązuje dany problem, czy i jakie efekty uboczne może wprowadzać („zawijaj wiersze” dla tych, którzy używają konsoli ;-)
- Na końcu dodaj identyfikator zmiany/specyfikacji, zgłoszenia błędu, etc. w celu łatwiejszej identyfikacji.

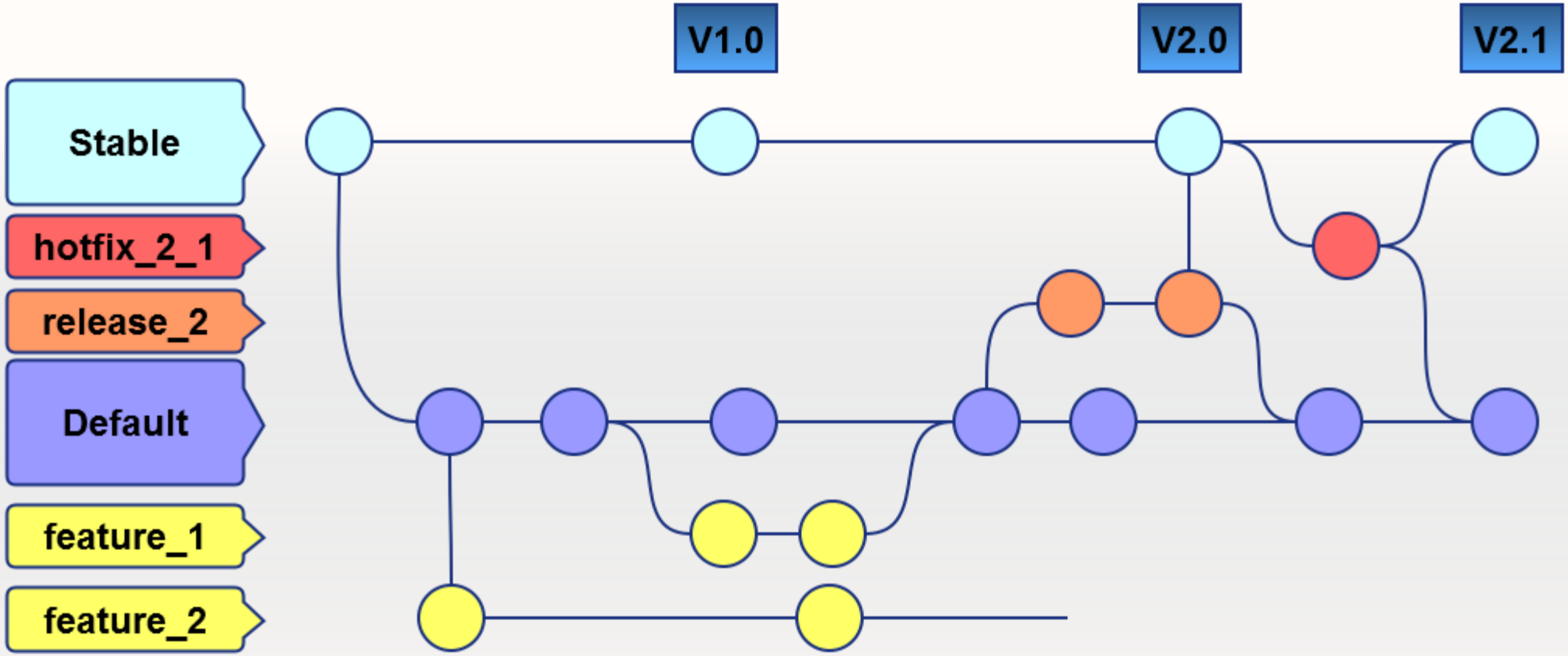
Zrobiłem „błędny” commit, co teraz?

- Póki nikt o tym nie wie, to nic się nie stało ;-)
- Git i mercurial mogą wycofywać commit, który nie został wypchnięty do innego repozytorium (undo)
- Jeśli już wszyscy o tym wiedzą, to należy przeprosić ;-)
- Commity mogą być wycofywane (backout), czyli wykonywany jest commit odwrotny do danego.

Rozwiązywanie konfliktów.

Gra Street Fighter IV,
zrzut ekranowy z serwisu 1up.com

Rozwiązywanie konfliktów.



Rozwiązywanie konfliktów.

Jan Kowalski

```
[…]

frmSimpleDbUpdateMode=Request.Form.Item(…)
If frmSimpleDbUpdateMode=„1” then
   SimpleDbUpdateMode=true
else
   SimpleDbUpdateMode=false
end if

[…]
```

Zbigniew Kowalski

```
[…]

frmAllowReplaceDepts=Request.Form.Item(…)
If frmAllowReplaceDepts=„1” then
   AllowReplaceDepts=true
else
   AllowReplaceDepts=false
end if

[…]
```

Rozwiązywanie konfliktów.

```
[…]

frmSimpleDbUpdateMode=Request.Form.Item(…)
If frmSimpleDbUpdateMode=„1” then
   SimpleDbUpdateMode=true

frmAllowReplaceDepts=Request.Form.Item(…)
If frmAllowReplaceDepts=„1” then
   AllowReplaceDepts=true
else
   SimpleDbUpdateMode=false
end if
 AllowReplaceDepts=false
end if

[…]
```

Nigdy nie ufaj automatom ;-)

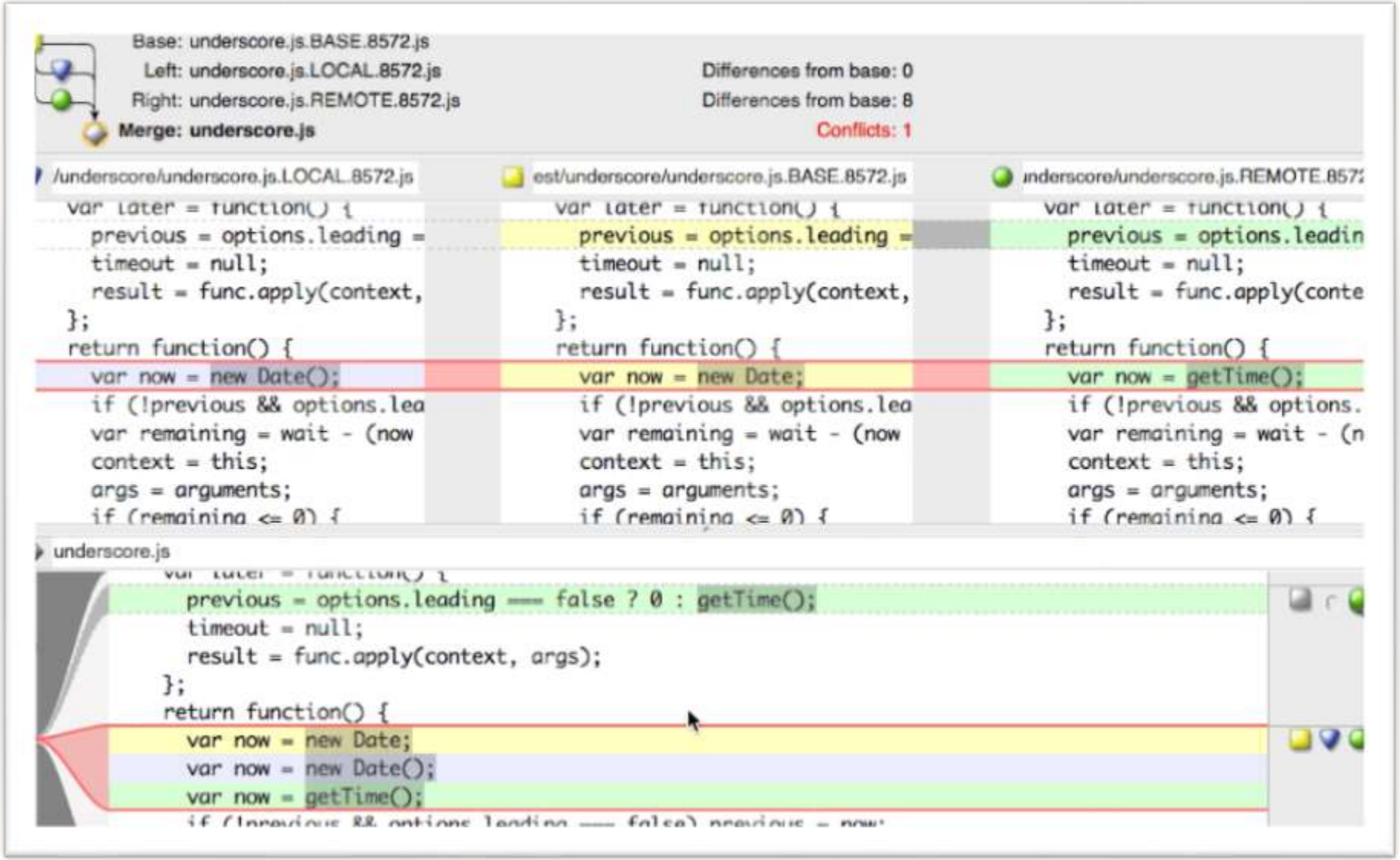
One nie rozumieją składni.

Rozwiązywanie konfliktów.

Narzędzia do trójstronnego łączenia zmian:

- KDiff3 (http://kdiff3.sourceforge.net/)
- P4Merge (http://www.perforce.com/product/components/perforce-visual-merge-and-diff-tools ; własnościowy)

Rozwiązywanie konfliktów.



Z wpisu na blogu o tytule „Jak używać p4merge z git-em” http://naleid.com/blog/2013/10/29/how-to-use-p4merge-as-a-3-way-merge-tool-with-git-and-tower-dot-app

Podumowując

- Stable i default, dwa bratanki.
- Nowe rzeczy dodawaj w swojej piaskownicy.
- Poprawki również (i nie zapominaj, że default/develop również ich potrzebuje).
- Opisz swoją pracę.
- Przy łączeniu zmian, nie ufaj automatom.

To ile może być łącznie tych gałęzi?

Tego nie wie nawet Chuck Norris ;-)