Software management

Petr Ferschmann
petr@ferschmann.cz

Předpoklady k přednášce

Víte co je ...

- ... kompilace
- ... preprocesor
- ... verzovací systém (např. CVS)

Kerio Technologies

- Síťové aplikace
- C++
- Cross platform produkty (MS Windows, Linux, MacOS X, Solaris)
- Různé překladače (MSVC 6.0, MSVC .NET, GCC 2.95, GCC 2.96, GCC 3.2)

Životní cyklus

Každý software i jeho části mají životní cyklus, který se stále opakuje.

- Vývojová
- Testovaná (zmražená)
- Stabilní
- Ukončená

Životní cyklus 2.

Nejedná se o životní cyklus zákazníkům, ale o release managementu.

Cyklus může proběhnout i několikrát za den.

Mohou být vnořené – cyklem projde jen část a posleze celý produkt.

Životní cyklus – vývojová

- Probíhá hlavní vývoj.
- Po dokončení ucelených změn přejde do fáze testování.
- Najednou může probíhat více vývojových větví paralelně, které se po otestování sloučí (a přejdou společně znovu do fáze testování)

Životní cyklus – testovaná

- Nejsou přidávány žádné nové vlastnosti.
- Jsou jen opravovány chyby.
- Minimální změny rozhraní.
- Provádí se intenzivní testy.

Životní cyklus – stabilní

- Již žádné změny.
- Opravy jen velmi drobných chyb.
- U velkých chyb přejde zpátky do zkrácené vývojové a testované.
- Rozhodně by se nemělo měnit rozhraní.

Životní cyklus – ukončená

- Není již používaná žadným produktem.
- Není nutné ji dále aktualizovat.
- V podstatě je mrtvá.
- Ukončená neznamená ukončená podpora zákazníkům.

- Nejoblíbenější stav programátora :-)

Kerio

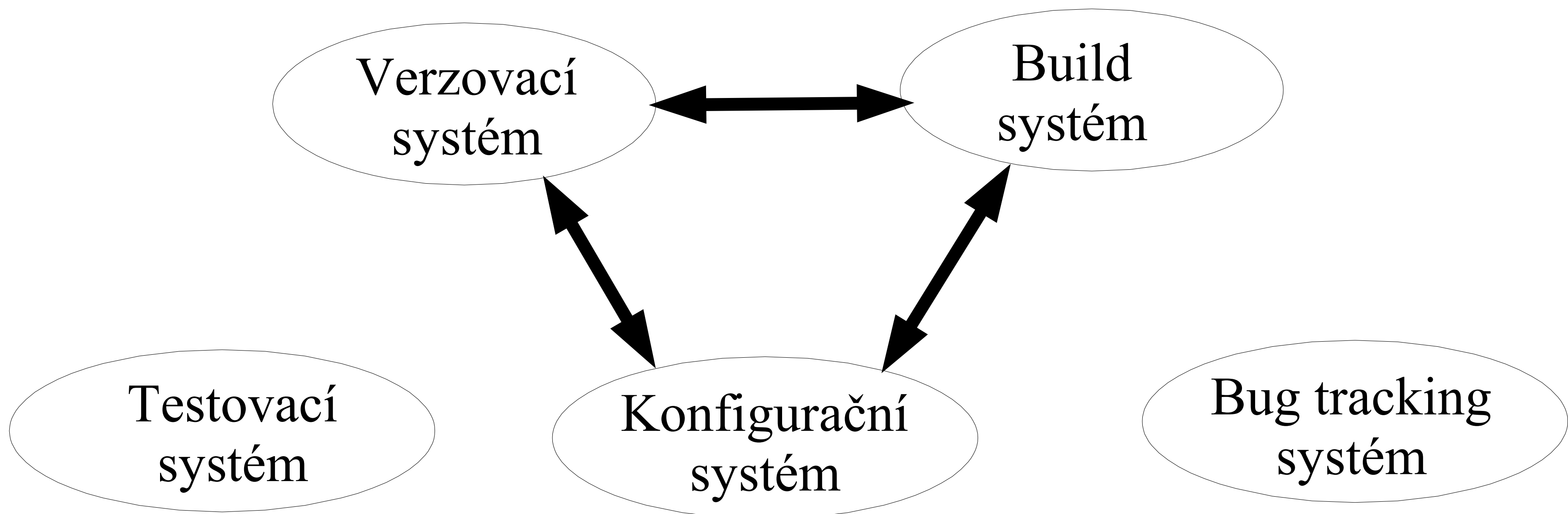
- Pod touto značkou budou vždy praktické realizace u nás.
- Produkty se skládají z komponent (knihoven)
- Knihovny i produkty vyvyjí jiný tým.
- Životní cyklus mezi knihovnami a produkty je cca 2 měsíce.
- Snažíme se množství udržovaných verzí minimalizovat (obvykle máme 1 až 2 stabilní).

Kerio 2.

- Abychom předešli exponenciálnímu množství kombinací komponent a minimalizovali čas kdy musí produkty přecházet na nové verze ->
- Vydáváme knihovny v tzv. main releasech.

Základní stavební kameny

Aby mohl cyklus probíhat musí mít kde:



Konfigurace produktu

- Produkt se skládá z komponent.
- Komponenta může záviset na jiné (a na konkrétní verzi).
- Musíme být schopni sestavit na základě této informace celý produkt.
- Tyto informace je nutné ukládat.

Příklad konfigurace produktu

- Kerio MailServer 5.6
  - WebServer (libweb) 1.0 ve variantě s podporou KStream
  - Nástroje pro práci s poštou (libmail) 1.3
  - Logovací knihovna (libtlog) 2.3
  - Implementace SASL (libtauth) 1.3
  - OpenLDAP

Kerio

- Protože CVS neumí seskupovat branche máme konfiguraci produktu uloženou v build systému, který si umí odpovídající verzi vyzvednout.
- Klíčem verze je
  - Jméno (např. libweb)
  - Branch (např. LIBWEB_1_0)
  - Verze (1.0)

Verzovací systém

- Místo kde probíhá celý cyklus.
- Udržujeme zde jednotlivé verze (v tzv branche)
- Umožňuje spolupráci více lidí.

Kerio

- Využíváme systém CVS

- Vývoj probíhá paralelně v několika větvích.
- Protože CVS není vhodné pro binární soubory, ukládáme zkompilované části na síťový disk.

Jak ukládat varianty?

- Každý produkt má různé varianty
  - Ladící verze (navíc různé testy - assert, výpisy, ...)
  - Optimalizace a úpravy pro různé zákazníky
  - Různé platformy
  - Různé varianty produktu (light, enterprise, ...)

Jak ukládat varianty?

- Můžeme uložit do verzovacího systému.
  - Pokud to umožňuje efektivně
  - Nevýhodou je exponenciální růst verzí při množství variant.
- Podmíněným překladem programu (preprocesor)
  - Při drobných rozdílech
- Podmíněným výběrem souboru k překladu
  - Při velkých rozdílech a u systému bez podpory preprocesoru

Platforma?

- Kombinace
  - Procesoru
  - Operačního systému
  - Překladače (kvůli name mangling)
  - Hlavních knihoven (libc, STL)

Např. Intel na Linux s GCC 3.3

Cross-platform knihovny

- Nejlepším způsobem jak se postavit tomuto problému je použít knihovnu:
- Příklady:
  - QT
  - ACE (Adaptive Communication Environment)
  - PWLIB
  - GTK, wxWindows

Cross-platform konfigurace

- Pro každou platformu máte variantu
- Obvykle řešeno pomocí preprocesoru.

- Při dvou platformách lze použít:

```
#ifdef _LINUX
```

- Při větším množství platforem neudržitelné
- Někdy se liší verze od verze

Cross-platform konfigurace 2.

- Zvolíme vlastnost a přidělíme jí symbol

```
#ifdef HAVE_SYS_INET_H
#  include <sys/inet.h>
#endif
```

Cross-platform konfigurace 2.

- Zvolíme vlastnost a přidělíme symbol

```
#ifdef HAVE_SYS_INET_H
#include <sys/inet.h>
#endif
```

- Vytvoříme soubor s konfigurací (config.h) kde definujeme dané symboly:
  - Pro danou platformu staticky
  - V konfigurační fázi buildu (častěji)

Build systém

- Umí přeložit a sestavit program
- Úzká spolupráce s konfiguračním systémem
  - Kde najdu knihovny
  - Jakou variantu chci kompilovat
- Úzké navázání na testovací systém

Typy build systémů

- Nativní
  - spolupracující s IDE
- Command line
  - volají command-line verze překladačů
- Meta build systémy
  - generují pro nativní build systémy

Nativní

- Jsou specifické pro systém
- Dobrá integrace se systémem (např. s IDE)
- Některé postrádají konfigurační fázi
  - MS Visual C++ 6.0 a 7.0 (.net)
  - Borland C++ Builder
  - autoconf/automake (patří i do command-line)

MS Visual C++

- Omezené možnosti
  - Chybí konfigurační fáze
  - Pracná tvorba (při velkém množství projektů – např. testy)
  - Import jen jedním směrem (6.0 do 7.0)
  - U verze 6.0 nedostatečné automatické buildy
  - Menší flexibilita (formát je velmi jednoduchý)
- Skvělá integrace s IDE
- Jen pro MS Windows
- Výborný při při tvorbě jen pro Windows

Command line

- Bývají velmi flexibilní (až moc)
- Neintegrace s IDE (některé)
- Některé jsou cross platform
- Některé umí více překladačů
  - autoconf/automake
  - ant
  - jam (a jeho varianty)

KERIO

Autoconf/automake

- Nejpoužívanější a nejlepší
- Velmi flexibilní (až moc)
- Vynikající konfigurační fáze
- Integrován testovací systém (bez statistik)
- Někdy auto-peklo (hlavně s verzemi)
- Dva oddělené programy – občas to skřípe
- Špatná podpora MS Windows a překladače z MS Visual C++
- Pomalý – pro některé změny je nutné pouštět znovu konfigurační fázi (např. přidání Makefile.am)
- Protože je používaný – snadná integrace knihoven třetích stran

ant

- Viz předchozí přednáška

- S C++ není ideální
- Hlavně Java (velmi oblíbený)
- Nemá konfigurační fázi

jam

- Nezávislý na externích programech
- Jednoduchý
- Celkem flexibilní
- Použit např. v boost.org
- Podpora různých překladačů (včetně MS Visual C++)
- Nemá konfigurační fázi

Meta

- Generují projekty pro ostatní build systémy
- Snadná integrace vygenerovaných projektů s IDE
- Nejsou omezeny cílovým build systémem
  - cmake
  - qmake (použit při tvorbě programů v QT)
  - Premake

Cmake

- Velmi flexibilní
- Konfigurační fáze (GUI)
- Umí generovat Makefile i MS Visual C++ projekty
- Dobrá integrace s auto-build a auto-testovacím systémem (dart)

Cmake

Dashboard - Fri May 23 08:15:16 EDT 2003

Friday, May 23 2003

Dartboard | Date | ◀ | T | ▶ | Updates | Tests | Build | CVS | DoxygenHome | Rollup

Nightly Builds

Site	Build Name	Update	Cfg	Build		Test				Build Date	Submit Date
				Error	Warn	NotRun	Fail	Pass	NA		
heart.convex.com	HP-UX-aCC										No submission
esat.kuleuven.ac.be	HPUX-B.11.00-gcc-3.2	0	0	0	39	0	0	26	3	Fri May 23 06:17:38 METDST 2003	Fri May 23 00:32:01 EDT 2003
esat.kuleuven.ac.be	IRIX-6.5-CC-n32	0	0	1468	0	0	0	26	3	Fri May 23 06:14:40 CETDST 2003	Fri May 23 00:48:23 EDT 2003
rapture.sci.utah.edu	IRIX64-CC	0	0	0	0	0	0	26	3	Fri May 23 04:09:55 MDT 2003	Fri May 23 06:30:52 EDT 2003
rolle.engr.utk.edu	IRIX64-CC	0	0	0	0	0	0	26	3	Fri May 23 04:09:09 EDT 2003	Fri May 23 04:27:10 EDT 2003
rapture.sci.utah.edu	IRIX64-CC-64	0	0	0	0	0	5	21	3	Fri May 23 04:38:50 MDT 2003	Fri May 23 06:45:18 EDT 2003
manifold.crd	IRIX64-CC-n32	6	0	0	0	0	0	26	3	Fri May 23 02:00:42 EDT 2003	Fri May 23 02:05:26 EDT 2003
esat.kuleuven.ac.be	Linux-2.4-gcc-3.2	0	0	0	39	0	0	27	2	Fri May 23 06:27:29 AM CEST 2003	Fri May 23 00:38:49 EDT 2003
ringworld.kitwarein.com	Linux-c++	0	0	0	0	0	0	26	3	Thu May 22 22:04:08 EDT 2003	Thu May 22 22:07:17 EDT 2003
hythloth.kitware	Linux-como4301	6	0	0	0	0	0	24	5	Thu May 22 22:30:28 EDT 2003	Thu May 22 22:34:18 EDT 2003

Cmake

Testing started on Thu May 22 23:29:20 EDT 2003

Site Name: KALGAN

Build Name: CYGWIN-c++

26 passed, 0 failed, 0 not run

Name (sort by)	Status	Time (sort by)	Detail
complex	Passed	28.366	
complexOneConfig	Passed	26.167	
conly	Passed	3.596	
curl	Passed	117.698	
dependency_w_libout	Passed	21.052	
dependency_wo_lib_out	Passed	22.279	

Auto testy

- Programy je nutné testovat (viz junit přednáška)
- Testů není nikdy dost
- Musí umět spolupracovat s build systémem

Auto testy

- Požadavkem je automatické zpracování
  - Vezme produkt a dokáže jej sestavit (komponenty mají správnou verzi)
  - Sestaví testy a spustí je
  - Výsledky dokáže zpracovat

Testovací programy

- Junit – java
- Cppunit – přepis Junitu do C++ (stejné vlastnosti)
- boost.org tests – přepis Junitu podruhé
- Expect – pro console aplikace – očekává vstup a simuluje výstup

- Klikací – simulují události v GUI

Kerio – současnost

- Využíváme CVS
- Pro Unixy využíváme autoconf/automake s rozšířením, které dokáže podle popisu produktu získat i knihovny ve správne verzi
- Pro MSVC sada projektů. Bez konfigurační fáze
- Pro testy využíváme boost.org tests.
  - Důvodem použití oproti cppunit je pouze to, že knihovny boost již používáme
- Automatické kompilace. Výstup porovnáváme diffem. Testy pouštíme v autoconf/automake pomocí "`make check`"

Kerio – budoucnost

- Zůstaneme u CVS.
- Plánujeme testovací nasazení meta builderu. Zřejmě cmake.
- Pokud se osvědčí přejdeme úplně.

Otázky

Děkuji