The Programmer's Toolkit

Bernhard Wagner – xmlizer.biz

Agenda

- Einführung/Motivation
- Lösungsansätze
- Aktivitäten und Tools

Software-Entwicklungs-Prozess

- Optimierungsansätze
  - Strukturierung des Software-Entwicklungs-Prozesses
  - Kultur
  - Tool-Unterstützung

  - Lassen sich kombinieren
  - Stellen Hilfen dar, aber keine Garantie

Strukturierung

- Strukturierung Gesamtprozess
  - RUP http://www.rational.com/rup
  - XP http://xprogramming.com
  - V-Modell http://www.v-modell.iabg.de/
- Customization nötig
- Methodik auch auf kleineren Granularitätsstufen sinnvoll
  - z.B. "Methode immer vor dem Einchecken dokumentieren"

Kultur

- Huhn-Ei-Problem
  - z.B. Dokumentation ("es gibt keine Doku, also schreibe ich auch keine")
- Kultur muss gezielt gefördert werden
  - Initalzündung
  - Aufrechterhaltung/Weiterentwicklung durch designierte Stelle
- Eine simple Regel

  "Dinge die häufig getan werden *müssten*, sollen einfach getan werden *können*."

Tool-Unterstützung

- Coding
- Unit Testing
- Logging
- Build-Prozess
- Versioning
- Sourcecode Dokumentation

Coding

- Tools
  - Full Blown IDEs
  - Texteditoren, um IDE-Funktionen erweitert
- Auswahlkriterien
  - Integration mit Standard-Tools (Repository, Dokumentation, Debugger)
  - Konfigurierbarkeit (Compiler, JDK)
  - Unterstützung vom Deployment
  - Erweiterbarkeit (Integrationsmöglichkeit)
  - Ergonomie (Benutzerprofile)

Coding Referenzen

- http://www.eclipse.org
- http://www.borland.com/jbuilder
- http://www.windriver.com/products/sniff_plus/
- http://www.vim.org
- http://www.emacs.org

Test Driven Development

Write test code to ask your system a question, write system code to respond to the question and keep the dialogue going until you've programmed what you need.

Test-first programming and merciless refactoring are the key practices of evolutionary design.

By learning to evolve your designs, you can become a better software designer and reduce the amount of work you over- or under-engineer.

-- Joshua Kerievsky

Unit Testing

- Umfeld
  - Extreme Programming
  - Test Driven Development
  - Agile Processes
- Vorteile
  - fördert die modulare Modellierung
  - laufend aktualisierte Dokumentation
  - fördert das Denken in Interfaces statt in Implementationen
  - frühe Fehlererkennung
  - automatisierte QA

Junit

- Junit übernimmt Testfunktionalität als Java-Framework
  - Auf- und Abbau der Fixtures (für den Test nötige Objekte inkl. Prüfling) vor respektive nach jedem einzelnen Test
  - Gruppieren von Tests, sammeln der Resultate
  - Entwickler kümmern sich nur noch um Spezifika Ihres Tests.
- Portierungen:
  - Perl: Test::Unit http://search.cpan.org/dist/Test-Unit/
  - C++: CppUnit http://cppunit.sourceforge.net/

Referenzen Unit Testing / TDD

- http://junit.sourceforge.net/doc/testinfected/testing.htm
- http://cppunit.sourforge.net
- http://www.xprogramming.com
- http://c2.com/cgi/wiki?CppUnit
- http://www.agilealliance.org
- Martin Fowler: "Refactoring", Addison-Wesley, 1999.
- Frank Müller; Immer wieder treffen; Agile Softwareentwicklung; iX 2/03, S.109
- Dierk König; Code Workout; Training mit Unit-Tests in Perl; iX 2/03, S. 124

Logging

- Logging: Poor man's Debugging ? No!
- Debugging vs. Logging
  - Debugging: Zur Entwicklungszeit
  - Logging: Zur Laufzeit
- Brian W. Kernighan und Rob Pike (Entwickler des UNIX Betriebssystems):
  - Debugger nur einsetzen, um den Wert von ein bis zwei Variablen zu überprüfen oder einen Stack-Trace anzuzeigen.

Nachteile Debugging

- gemäss Kernighan/Pike:
  - man verliert sich leicht in komplizierten Datenstrukturen und Programmfluss
  - Schrittweise durch ein Programm zu gehen ist weniger produktiv, als an wohlüberlegten Stellen im Programm output-statements einzuflechten und selbst-überprüfenden Code einzufügen
  - Log statements bleiben im Programm, während Debugging sessions flüchtig sind

Logging

- Nachteile
  - kann eine Applikation verlangsamen
  - Wenn zu wortreich eingesetzt, kann der output unlesbar werden "Man findet vor lauter Bäumen den Wald nicht"

log4j Konfiguration @ Edittime

- Entwicklungszeit (Source-Code):
  - 2 Ebenen: Loglevel und Kategorie
  - Log Request Level:
    - debug < info < warn < error < fatal
  - Kategorie:
    Frei wählbare hierarchische Kategorien.
    Erfahrungsgemäss am besten identisch zur Klassenhierarchie, pro Klasse einen statischen Logger vorsehen.

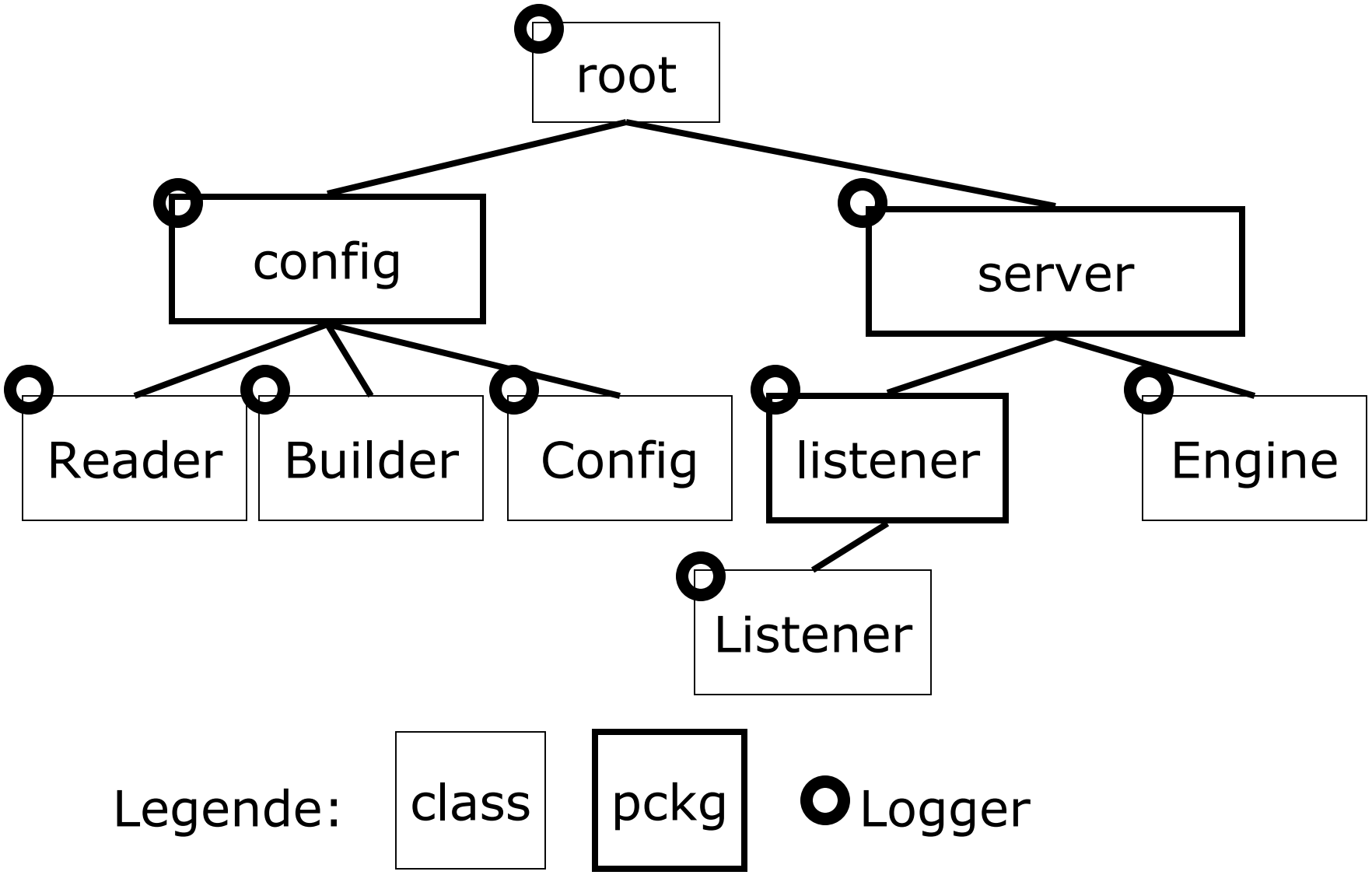
log4j Code Example

```
package com.mycompany.myproject;
import org.apache.log4j.Logger;
class MyClass {
  // ...
  public void doIt(int par){
    fgLogger.debug(“doIt("+ par +")");
          // ...
    fgLogger.debug(“exit doIt");
  }
  // ...
  private static Logger fgLogger =
    Logger.getLogger(MyClass.class);
}
```

log4j Configuration @ Runtime

- Via Konfigurationsfiles:
  - Welche logging statements
  - Wohin der output gelangt
- Für jeden Logger kann bestimmt werden:
  - Welcher Loglevel berücksichtigt wird
  - Outputformat (Layout, à la printf)
  - Output Destination (Appenders)
    - Console, File, GUI, Socket, JMS, NT event Loggers, remote UNIX Syslog
    - Eigene Appenders können entwickelt werden

log4j Categories and Inheritance



log4j Configuration File

```
# Set root logger level to DEBUG and its only
# appender to A1.
log4j.rootLogger=DEBUG, A1

# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern= \
 %d [%t] %-5p %c - %m%n

log4j.logger.com.mycompany.myproject.config=WARN
log4j.logger.com.mycompany.myproject.config.Builder=ERROR
```

Logging Practices

- Soviel wie möglich auf DEBUG-Level:
  - Methodeneintritt und –austritt
  - Bei Übergang zu Fremdsystem ist Loggen besonders nützlich um Engpässe zu identifizieren
  - Bei Fremdsystemen auch die übergebenen und erhaltenen Daten loggen

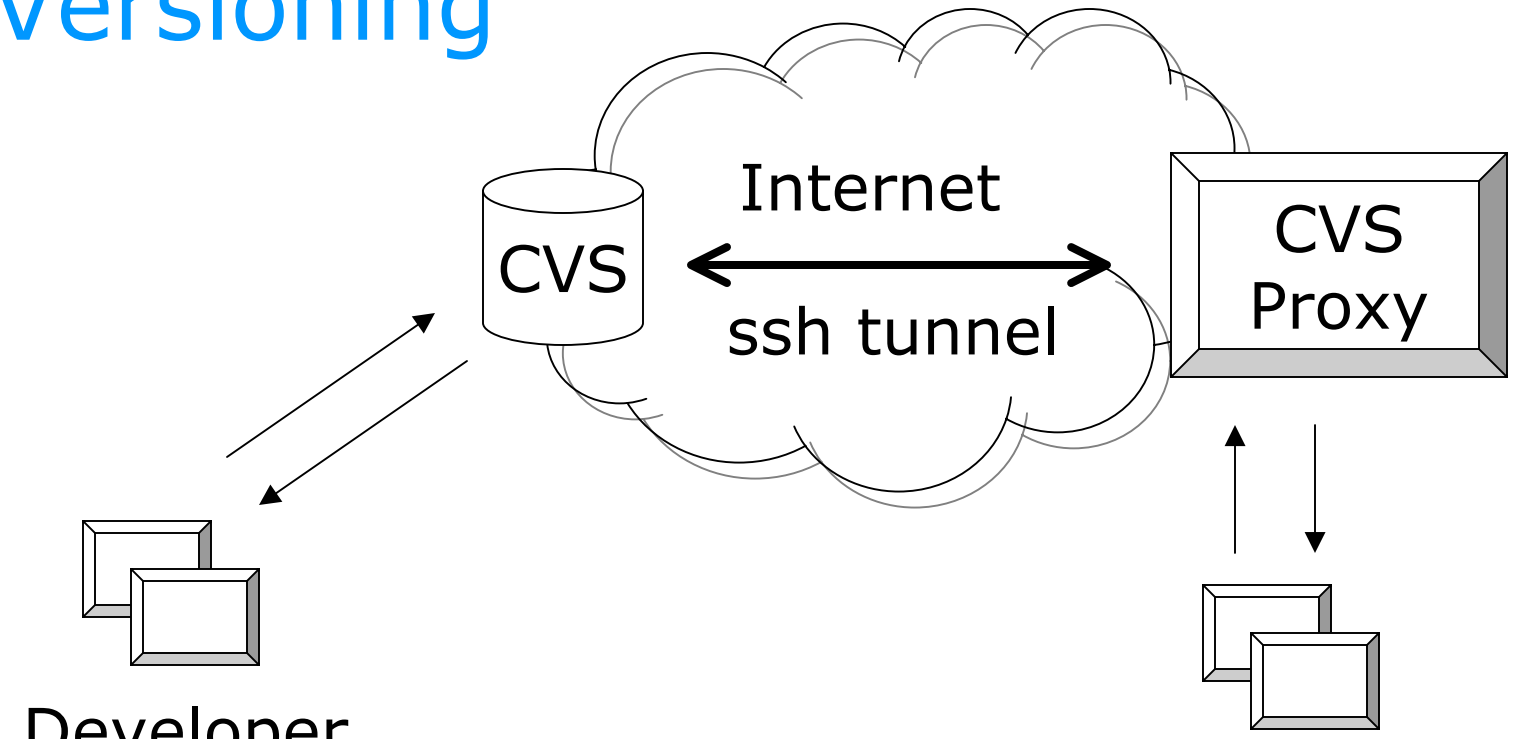
Platforms, Referenzen Logging

- Java: "log4j"
  http://jakarta.apache.org/log4j/
- Perl: "log4perl"
  http://log4perl.sourceforge.net/
- C++: "log4cpp"
  http://log4cpp.sourceforge.net/
- http://jakarta.apache.org/log4j/docs/documentation.html
- http://supportweb.cs.bham.ac.uk/documentation/tutorials/docsystem/build/tutorials/log4j/log4j.html

Versioning

- Vorteile des Einsatzes von Versioning:
  - zu jedem Zeitpunkt kann eine beliebige vorherige Version der Software rekonstruiert werden.
  - Änderungen am Code können einzelnen Entwicklern zugeordnet werden.
  - Entwicklung im Team wird sehr vereinfacht.
  - checkin/checkout können angepasst werden:
    - automatisch Mail verschicken an Mailingliste
    - automatisch Tests ausführen, checkin nur bei Erfolg

Versioning



Versioning Tools

- CVS
  - Klassiker, weit verbreitet
  - Standard bei Open Source Projekten
- Subversion
  - Soll Nachfolger von CVS werden, mit Verbesserungen, u.a.:
  - Verzeichnisse, Umbenennungen, und Datei Meta-Information werden versioniert.
  - Commits sind wirklich atomar (Transaktionen)
  - Effizientes Handling von binären Dateien
  - Standard-Protokoll WebDAV

Versioning Practices

- Jeden checkin-Kommentar beginnen mit einem Schlüsselwort:
  - ADDED, REMOVED, MOVED, CHANGED oder FIXED
- Möglichst kleine, in sich abgeschlossene Änderungsschritte einchecken.

Referenzen Versioning

- http://cvsbook.red-bean.com/
- http://www.cvshome.org/
- Karl Franz Fogel:” Open Source Development with CVS”, The Coriolis Group, October 1999
- http://subversion.tigris.org/
- Stefan Tilkov; Eins, zwei, drei; CVS-Nachvolger Subversion; iX 2/03, S.116

Sourcecode Dokumentation

- Grundregel:
  "Give them tools, not rules"
- Java: javadoc
- C++: Doxygen
- Perl: POD

javadoc: Vorteile I

- Standard: einmal erlernt, kann jeder Entwickler dokumentieren, bzw. die Dokumentation anwenden, selbst nach Projekt- oder gar der Firmenwechsel.
- konsequente Anwendung der Dokumentation fördert Programmierung gegen *Interfaces* statt gegen eine konkrete *Implementation*
- Jeder eliminierte Freiheitsgrad, sowohl bei Teamarbeit, als auch individuell, spart viel Zeit

javadoc: Vorteile II

- Vorgegebene Struktur der Dokumentation (Methoden-parameter, Rückgabewert, Exceptions) erlaubt automatisiertes Testen gewisser QA-Kritieren. Ebenso Aktualität der Dokumentation
- Standard: Tool zur Generierung von neuem Ausgabeformat der Dokumentation, z.B. latex, pdf, etc. erzeugt automatisch Zusatznutzen für alle in dieser Weise dokumentierten Projekte

Dokumentation: Doxygen

- log4j -> log4perl, log4cpp
- javadoc (Java) -> Doxygen (C++, Java)
- Vorteile v. Doxygen:
  - bietet einen Stil entsprechend javadoc, somit können Java-Programmierer ihr Wissen 1 zu 1 übertragen.
  - Kann auch zur Code-Dokumentation von Java verwendet werden.
  - Tools für den Output in verlinktem HTML und latex existieren.
  - Bietet mehr Features als javadoc (grafische Klassenhierarchie, globale und individuelle Konfiguration)

Dokumentation: POD

- Perl definierte ab Perl5 POD (Plain Old Documentation)
- vergleichbare Mächtigkeit wie HTML (Headings, Bullets, Links).
- Nachteil: Weniger strukturiert als javadoc und Doxygen bzgl. Methodenparameter, etc.
- Vorteil: Es existieren Testtools, die die Beispielcodes innerhalb der Dokumentation ausführen und die Resultate testen.

Referenzen Dokumentation

- http://java.sun.com/javadoc/
- http://www.doxygen.org/
- http://www.perldoc.com/perl5.6/pod/perlpod.html

Build-Prozess

- Tools für den Build-Prozess
  - make (Unix-Umfeld)
  - ant (Java)
  - maven (Java, Apache Jakarta)
  - jelly (Java, Apache Jakarta)

Build: make

- make
  - Das älteste der Buildtools, im Unix-Dunstkreis entstanden
  - etwas knorrig in der Syntax
  - weit verbreitet
  - nicht standardisiert
  - Perl verwendet make, baut aber eine standardisierte Umgebung darum herum

Build: make example

- Makefile excerpt

```
all: main
doku: main
   doxygen doxygen.conf
main: $(OFILES)
   $(CC) -o $@ $(OFILES)
```

- Aufruf

```
make main
```

Build: ant

- ant
  - entstand zunächst aus Schwierigkeiten mit "make"
  - Ist mittlerweile Standard build tool im Java Umfeld.
  - Alle Tools innerhalb von ant ("tasks") sind auf allen Plattformen verfügbar (ähnlicher Ansatz wie Perl)
  - XML
- Maven, Jelly
  - Weiterentwicklungen von ant

Build: ant example

- build.xml excerpt

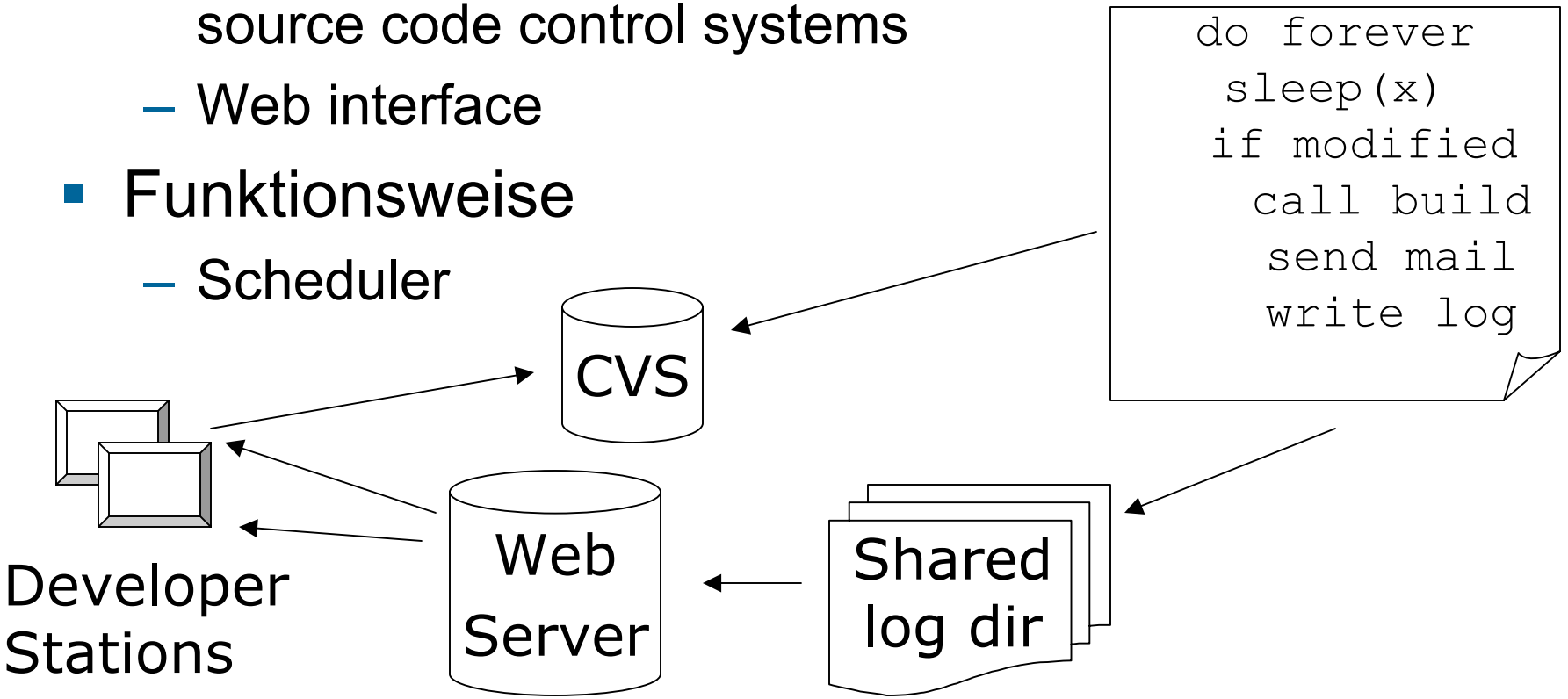
```
…
<target name=main depends="ofiles">
   <javac srcdir="${src.dir}“
      destdir="${build.classes}“
   </javac>
</target>
<target name= doku depends=main>
…
```

- Aufruf

```
ant main
```

Build: cruisecontrol

- Cruisecontrol
  - Framework für Continuous Integration
  - plugins für email notification, ant, vss, source code control systems
  - Web interface
- Funktionsweise
  - Scheduler



Referenzen Build-Prozess

- http://jakarta.apache.org/ant/
- Michael Marr; Fleißige Ameise; ant; Make-Alternative für Java; iX 2/01, S.56
- http://cruisecontrol.sourceforge.net/
- Dierk König; Tempomat; Teamarbeit mit dem Open-Source-Werkzeug CruiseControl; iX 8/02, S.94
- http://www.gnu.org/software/make/

Language vs. Tool

Aspect	Java	Perl	C++
Unit Testing	***JUnit***	Test::Unit	CppUnit
Logging	***log4j***	log4perl	log4cpp
Build	ant	make	make
Versioning	cvs	cvs	cvs
IDE	eclipse	(Komodo)	eclipse
Documentation	***Javadoc***	POD	Doxygen

Legende: ***Originator***

Ende

Vielen Dank für Ihre Aufmerksamkeit

Folien online erhältlich:
http://xmlizer.biz/talks/iex03